midi ingénierie
NEXEYA Products Division

3509, route de Baziège
31670 LABÈGE
France
Tél. : 33 (0)5 61 39 96 18
Fax : 33 (0)5 61 39 17 58
www.midi-ingenierie.com

midi ingenierie

BMAC RS485
User manual

Date : 23.11.12

Reference : bmac_v2_um_en.pdf
Ref. MI : CMN1890890_en.docx

Release : 2

Author : C.MARTY
http://www.midi-ingenierie.com

S.A.S. au capital social de 713 046€
333 373 108 RCS Toulouse
Siret 333 373 108 00013
APE 7112B
TVA FR60 333 373 108

GROUPE NEXEYA

Index

1. Presentation

1.1. Introduction

BMAC module is a digital indexer with a microstep amplifier. In is meant to drive bipolar stepper motors (4, 6 or 8 wires). The power of its DSP makes it ideal for small single-axis application as well as complex multi-axis systems.

The amplifier stage delivers 45V 2.5A_{RMS}, which is well suited for NEMA17 and NEMA23 motors. High power version BMAC-H (45V 7A) is suited for NEMA23, NEMA24 or NEMA43 motors.

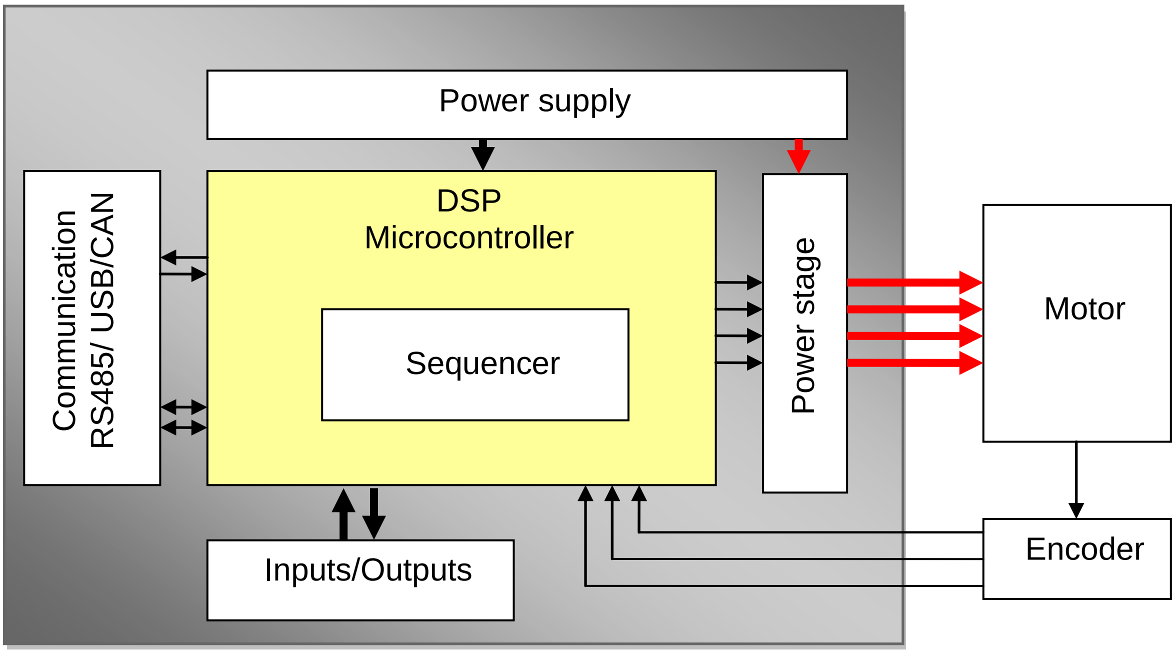
He motor can be driven in open-loop or in self-switched mode with encoder input. Self-switched motor control guarantees the motor position at any time, avoiding motor stall. BMAC cumulates the advantages of stepper and brushless motor, making it suitable for both positioning, velocity or torque control.

The built-in sequencer can record up to 500 commands. Together with the 8 optoisolated I/Os and the differential analog input, it can work as a PLC.

BMAC is a compact module with simplified connection. It can be controlled by USB (single-axis applications) or RS485 (multi-axis applications and higher noise immunity). CANopen DS402 (motion control) is available as an option.

1.2. Features

1.2.1. Architecture



1.2.2. Compatible motors

BMAC can drive hybrid bipolar stepper motors. Nominal current must be inferior to $2.5A_{RMS}$ for BMAC or $7A_{RMS}$ for BMAC-H.
The resolution is 50 microstep per step.

A motor with 200 steps/rev is best suited in order to maintain coherency of units (speeds are expressed in 1/100th RPM and positions in microsteps or 1/10000 rev).

1.2.3. Adjusting motor current

First, one should set the nominal current of the motor by writing the value (expressed in mA) in variable `#RATED_CURRENT` (max value is 2500 for BMAC and 7000 for BMAC-H). The value is specified on the motor or in its datasheet. It is the RMS current per motor phase.

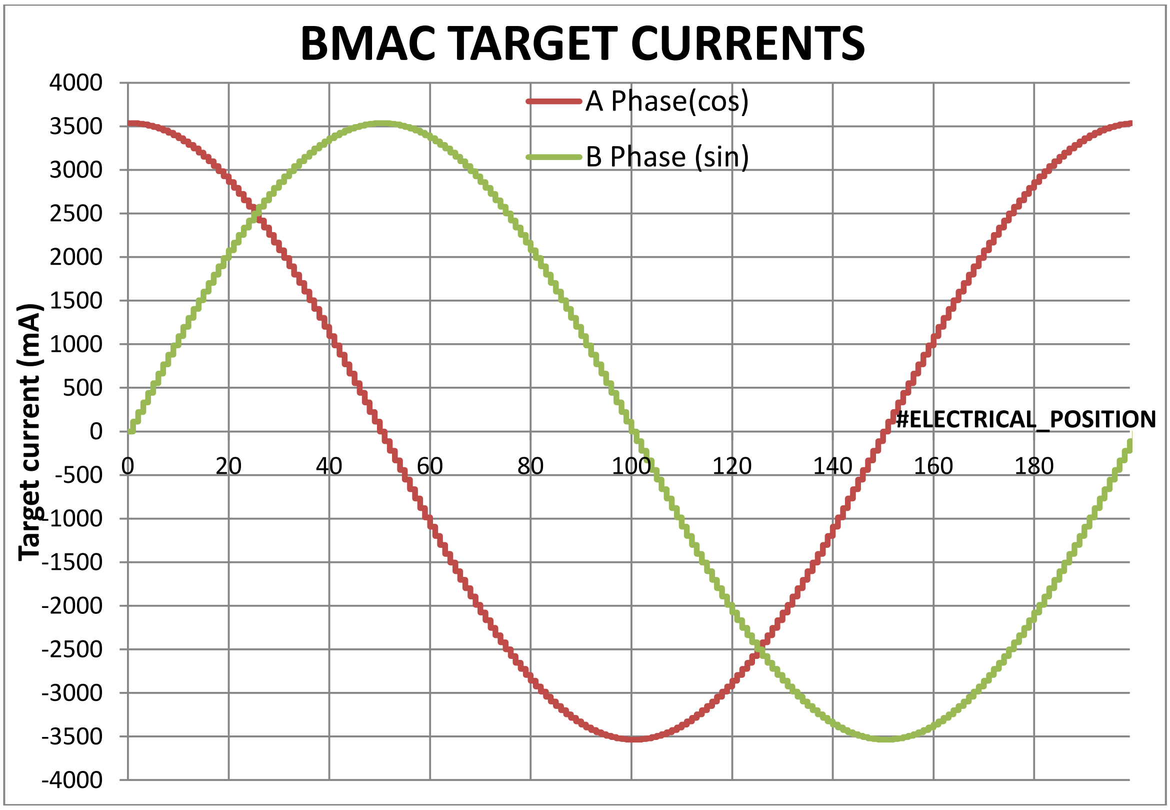
Effective current can be set to the desired value by setting variable `#CURRENT_RATIO` (0-1000) expressed in "per thousand of `#RATED_CURRENT`".
The current in mA is then given by the formula: `#RATED_CURRENT` * `#CURRENT_RATIO` /1000

The resolution of the current is 25mA for BMAC and 70mA for BMAC-H.

Target phase currents are given by :
I_{phaseA} = √2 * `#RATED_CURRENT` * `#CURRENT_RATIO` * cos (2π * (`#ELECTRICAL_POSITION` / 200))
I_{phaseB} = √2 * `#RATED_CURRENT` * `#CURRENT_RATIO` * sin (2π * (`#ELECTRICAL_POSITION` / 200))
with 0 ≤ `#ELECTRICAL_POSITION` ≤ 199
(During a clockwise movement, `#ELECTRICAL_POSITION` is incremented modulo 200)

Full steps correspond to the values 0, 50, 100 and 150 of `#ELECTRICAL_POSITION`.

Example with `#RATED_CURRENT:=2500` et `#CURRENT_RATIO:=1000`



In self-switched mode, the OPTIMIZED_CURRENT mode automatically adjusts the current needed by the motor so as to lessen thermal losses.

The motor power can be set on and off using command "POWER ON" or "POWER OFF". It is automatically set ON before any movement.

1.2.4. Encoder

BMAC can be linked to a 2-phase incremental encoder with A, /A, B, /B, I and /I signals. It must be 5V RS422 compliant. The 3 termination resistors are included in the BMAC.

The encoder can be directly powered by +5VCOD generated by the BMAC provided it needs less than 80mA.

The variable #ENCODER indicates the position of the incremental encoder expressed as 4 times the resolution. For example with a 10000 points per revolution encoder, there will be 40000 increments of #ENCODER per revolution.

The index input (I, /I) can be used for the homing procedure (please consult REFERENCE INDEX command).

In order to use the BMAC in close-loop self-switched mode, the encoder has to be 500 points per revolution and the motor has to be 200 steps per revolution. Both must have the same direction: in open loop, a clockwise rotation must result in an increase of #ENCODER.

1.2.5. Open Loop or Self-switched (closed-loop)

Select the mode using CLOSE_LOOP ON or CLOSE_LOOP OFF.
Bit #STATUS.28 in set if the BMAC is in self-switched mode.
The mode is memorized when the module is reset.
Default value is open loop.

	CLOSE_LOOP OFF (open-loop)	CLOSE_LOOP ON
Encoder	**With or withour encoder.** Encoder can have any resolution and any direction. The ratio between motor angle and encoder angle can vary.	Encoder has to be 500 pts/rev. It must be coherent with motor rotation.
Self-switching	Inactive	Enabled
Variable #POSITION	Indicates electric set-point, expressed in microstep with a 50µstep/step resolution. The variable is independent from #ENCODER.	Indicates physical position of the motor with a resolution of 10.000 positions per revolution and a precision of 1/1000th of a revolution.
Movements	According to electric position #POSITION.	According to variable #POSITION.
OPTIMIZE_CURRENT	Inactive	Enabled

1.2.6. Moves

Three kinds of movement can be performed:

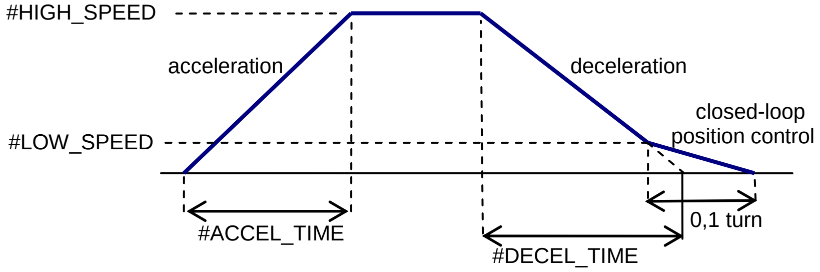
- Velocity mode (MOVE_SPEED) in which the motor turns at a constant speed.
- Position mode (MOVE_TO and MOVE_ON) in which the motor moves to a target position defined as absolute or relative.
- Interpolation mode (MOVE_INTERPOL) for synchronized multi-axis movement.

In Velocity and Position mode, trapezoidal or sinusoidal (S) velocity profile can be applied to optimize load acceleration and to fluidify movement. Acceleration and deceleration time can be adjusted by the user to suit every application.

In Position mode, the module performs closed-loop position control to reach target position with a precision of 1/10000th of a rev. This position control is performed at variable speed, proportionally to #LOW_SPEED and to position error.

A complete movement is thus decomposed into four distinct phases:

- ✓ Acceleration from current velocity (can be zero) to velocity setpoint.
- ✓ Constant speed movement.
- ✓ Deceleration from velocity setpoint to #LOW_SPEED.
- ✓ Position control from #LOW_SPEED to target position.



1.2.7. End-stops

BMAC has four end-stops:

- Two hardware end-stops using digital inputs IN7 and. User can connect sensors, dry contacts, etc…
- Two "virtual" software end-stops handled by the firmware, preventing the motor to go beyond `#POSITIVE_END` and `#NEGATIVE_END` positions. When enabled, if the motor tries to go beyond one of these end-points, movement is stopped and only reverse motion can be executed.

End-stops status is notified in variable `#STATUS`.

1.2.8. Sequencer

BMAC can record and execute up to 500 commands, allowing user to develop automation scripts that can be executed in standalone (without any computer or PLC).
Writing sequences is easy and intuitive. Memorized commands are executed at the rate of one per millisecond. Specific commands (`IF`, `JUMP`, `CALL`, `WAIT`, etc.) allow to control the sequence flow according to external events (digital and analog inputs, position or any internal variable).

1.2.9. Inputs / Outputs

BMAC has 8 isolated digital inputs/outputs. They can be used with the sequencer, for instance to launch predefined moves. Two inputs can be used as regular inputs when end-stop feature is disabled.

One differential analog inputs can be used to connect a sensor or a potentiometer. Just like digital inputs, analog input can be used with the sequencer, for instance to control axis velocity or position with a potentiometer.

1.1. Safety guidelines

1.1.1. General rules

- Modules are IP30. Protect the motor from oil-mist, cutting oil, metal chips and paint fume, etc. Otherwise it may result in failure of electric circuits of the Driver Unit.
- Avoid projection of solvent, acids, bases.
- Avoid exposure to radiations.
- Do not remove the cover from a module. Internal voltage can be dangerous.
- Do not touch a powered module: risks of getting burnt or electric shock.
- Do not touch the motor shaft : risk of harm.

1.1.2. Storing

- Modules must be stored or moved in their original package or a suitable conditioning.
- Protect the modules from direct sunlight and humidity.
- Keep ambient temperature within –20°C to +40°C.

1.1.3. Proper use

- **Warning ! Motor temperature can reach 85°C. Do not touch the module or the motor.**
- **Always power off and wait at least 20s before servicing a module or its connectors.**
- Damages may occur if pinning specification is not strictly observed.
- The module shall not be installed in a confined enclosure and ambient temperature shall be kept between –10°C et +40°C.
- The cable shall not be submitted to repeated bending.
- The module shall be installed on a fixed, stable chassis, otherwise it may fall and be damaged or harm someone.
- Mechanical ground of the module should be connected to the mechanical ground of the chassis.
- Do not insert anything in the modules holes.

2. Technical specifications

2.1. Power supply

	Supply voltage	Max consumed current
BMAC	12 to 45Vdc	2A
BMAC-H	24 to 45Vdc	6A

Warning! Consumed current increases as the supply voltage decreases.

Needed current depends on the total mechanical power needed: P = T * ω (P is the power in Watts, T is the torque in Nm, ω is the motor rotation speed in rad/s).

The higher the voltage, the higher is the high-speed torque.

Standstill or low-speed torque does not depend on the supply voltage.

If the voltage drops below 12V, motion is stopped and the modules enter undervoltage lockout mode.

Note *:* when motor decelerates, kinetic energy is sent back to the power supply. The power supply must then be able to handle reverse current. Voltage can increase (output capacitor charging). BMACs have overvoltage detection that suspends deceleration/braking when supply voltage goes above 46V. Additional protection disables motor power if voltage goes above 49V. A fault is then notified and the module must be reset (power off or MODULE_RESET command) or the fault must be acknowledged (#ERROR:=0).

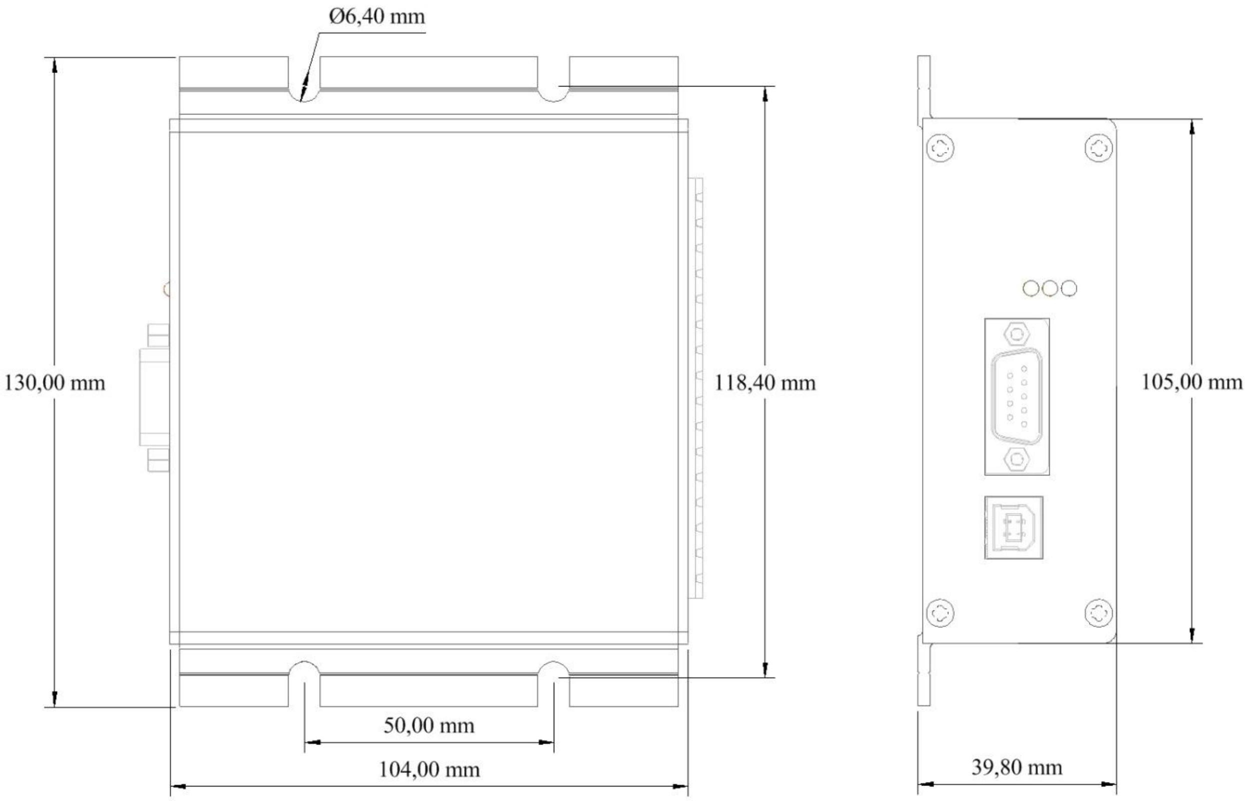
Whatever supply power is used, voltage can reach 49V

Optional ballast can dissipate extra energy in a resistor, so as not to reach the threshold.
If the supply voltage cannot handle nominal recovery voltage, one must insert a diode (75V/5A) in series between the module and the supply voltage.

2.2. Mechanical specification

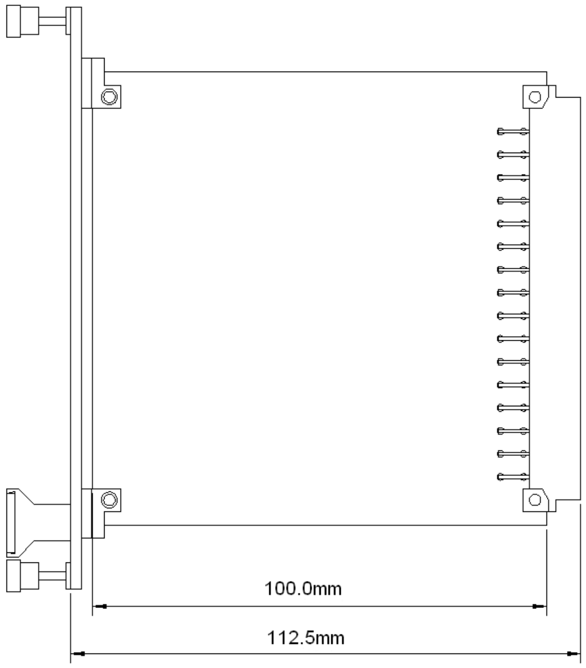
2.2.1. BMAC module

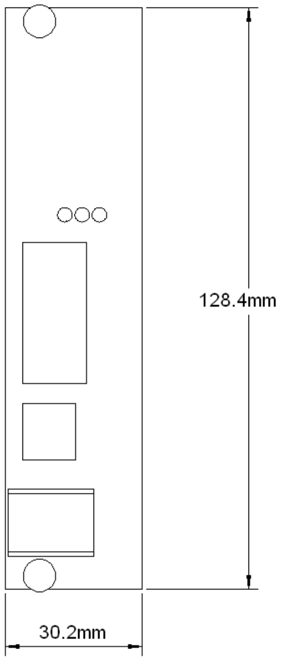
Weight: 470g
Product outline:



2.2.2. BMAC rack

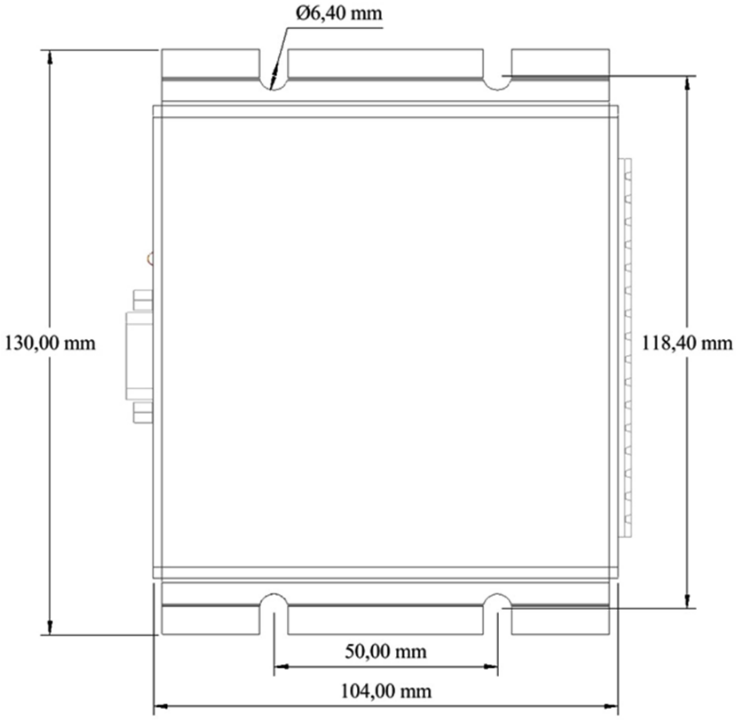
Product outline:

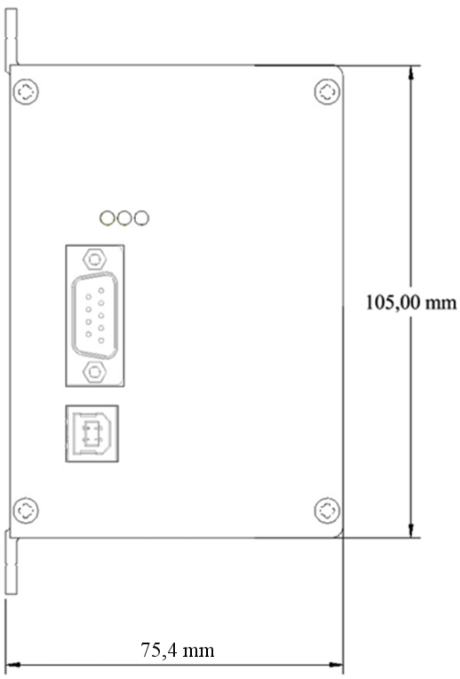




2.2.3. BMAC-H

Weight: 745g
Product outline:

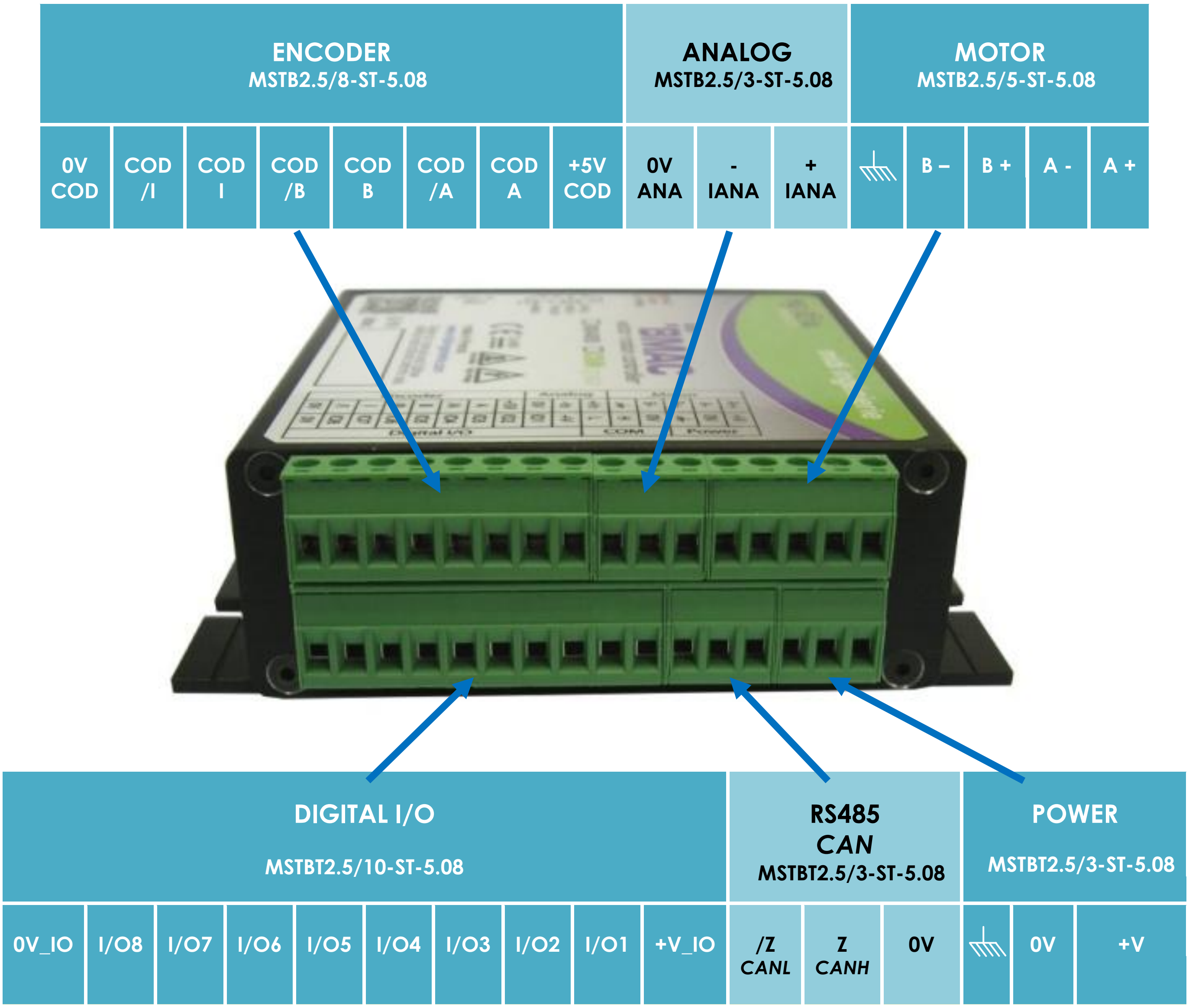




3. Pinning and configuration

3.1. Connector description

3.1.1. BMAC module

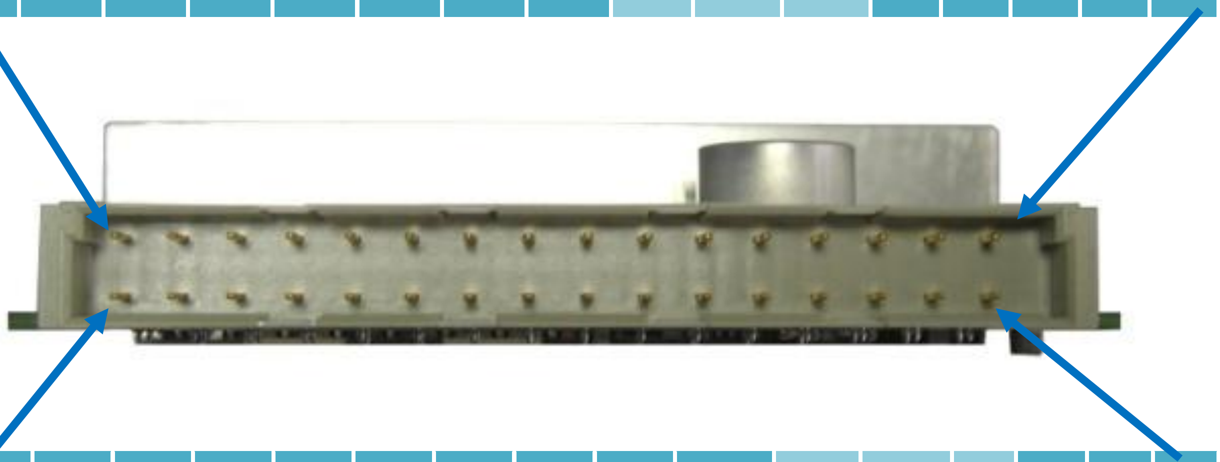


SubD9 Male : RS485 or CAN bus					
1	Reserved	**4**	Reserved	**7**	Z CANH
2	/Z CANL	**5**	⏚	**8**	Reserved
3	0V485 CAN	**6**	Reserved	**9**	Reserved

3.1.2. BMAC rack

Connector : DIN41612 D male :

ENCODER								ANALOG			MOTOR				
0V COD	COD /I	COD I	COD /B	COD B	COD /A	COD A	+5V COD	0V ANA	- IANA	+ IANA	⏚	B –	B +	A -	A +
32C	30C	28C	26C	24C	22C	20C	18C	16C	14C	12C	10C	8C	6C	4C	2C



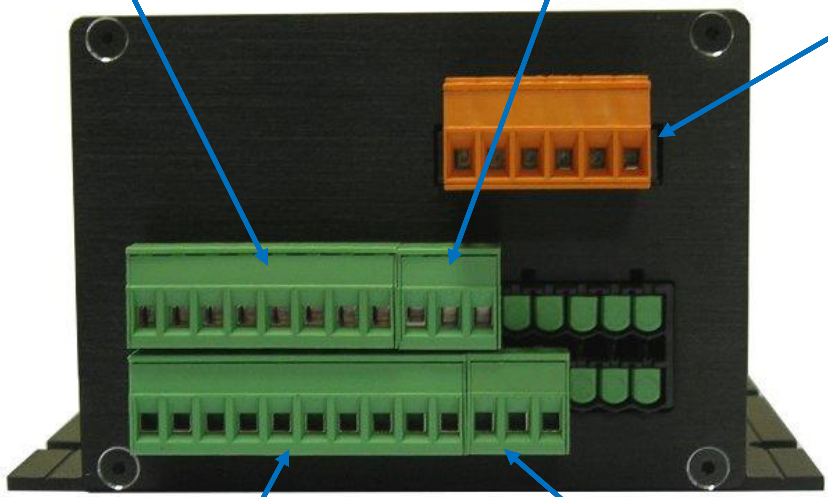
32A	30A	28A	26A	24A	22A	20A	18A	16A	14A	12A	10A	8A	6A	4A	2A
0V_IO	I/O8	I/O7	I/O6	I/O5	I/O4	I/O3	I/O2	I/O1	+V_IO	/Z CANL	Z CANH	0V	⏚	0V	+V
DIGITAL I/O										RS485 CAN			POWER		

SubD9 Male : RS485 or *CAN bus*					
1	Reserved	4	Reserved	7	*Z CANH*
2	/Z *CANL*	5	⏚	8	Reserved
3	0V485 *CAN*	6	Reserved	9	Reserved

3.1.3. BMAC-H

ENCODER MSTB2.5/8-ST-5.08								ANALOG MSTB2.5/3-ST-5.08		
0V COD	COD /I	COD I	COD /B	COD B	COD /A	COD A	+5V COD	0V ANA	- IANA	+ IANA

BLZ 5.08/02/180 SN (Weidmuller)					
POWER		MOTOR			
0V	+V	A-	A+	B-	B+



DIGITAL I/O MSTBT2.5/10-ST-5.08										RS485 CAN MSTBT2.5/3-ST-5.08		
0V_IO	I/O8	I/O7	I/O6	I/O5	I/O4	I/O3	I/O2	I/O1	+V_IO	/Z CANL	Z CANH	0V

SubD9 Male : RS485 or CAN bus					
1	Reserved	4	Reserved	7	Z CANH
2	/Z CANL	5	(ground symbol)	8	Reserved
3	0V485 CAN	6	Reserved	9	Reserved

For mechanical ground, it is recommended to use a screw + cable lug + star washer on one of the four fixation screw of the cover.

3.2. IOs features

I/Os can be used as GPIOs (General Purpose Input or Output) or alternate functions.

Each one of the eight IOs can be used as an input or as an output.

-To use it as an output, the user shall set the bit to the desired value (for instance `#OUTPUT.4:=1`) The corresponding input will be seen as the same level (`#INPUT.4` read as 1)

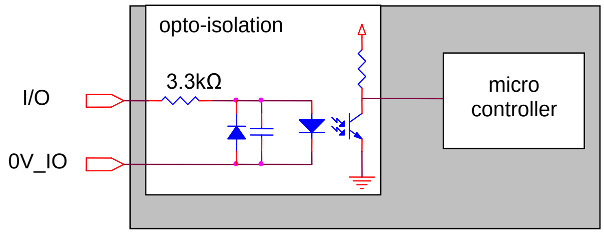
-To use it as an input, the user shall set the bit to 0 (for instance `#OUTPUT.7:=0`). The input can then be read in the `#INPUT` variable.

Some advanced functions are multiplexed with IOs. The user shall configure `#OUTPUT_CONFIG` to use either regular IO or advanced functions.

INPUT/OUTPUT	FUNCTION
I/O 1	general purpose input OR general purpose output or BUSY output (according to `#OUTPUT_CONFIG.1`) OR interpolation synchro output (if synchronized interpolation mode active)
I/O 2	general purpose input OR general purpose output or FAULT output (according to `#OUTPUT_CONFIG.2`)
I/O 3	general purpose input OR general purpose output
I/O 4	general purpose input OR general purpose output
I/O 5	general purpose input and capture input or reference input OR general purpose output
I/O 6	general purpose input or interpolation synchro input OR general purpose output
I/O 7	general purpose input or Positive end-stop input OR general purpose output
I/O 8	general purpose input or Negative end-stop input OR general purpose output

3.3. IOs specification

3.3.1. Optically isolated digital inputs

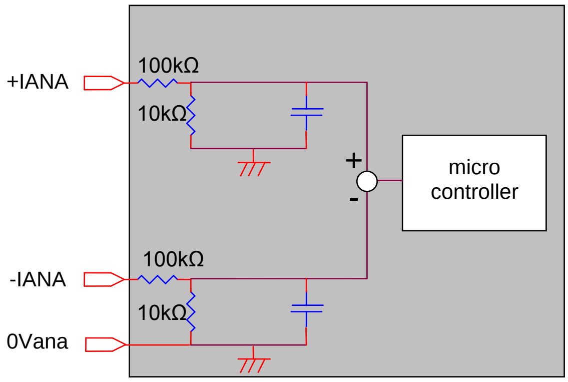


	min	max
VI_L (inactive) VI_H (active)	-30V 4V	1V +30V
II_L II_H	 1.1mA	25µA

Nominal voltage 5VDC to 24VDC
Signal voltage (inactive) 0VDC to 1VDC
Signal voltage (active) 4VDC to 30VDC
Admissible voltage ±30VDC
Galvanic insulation 50V
Reference voltage 0VIO is common for all inputs.
Digital inputs status can be read using `READ #INPUT`.

3.3.2. Analog input

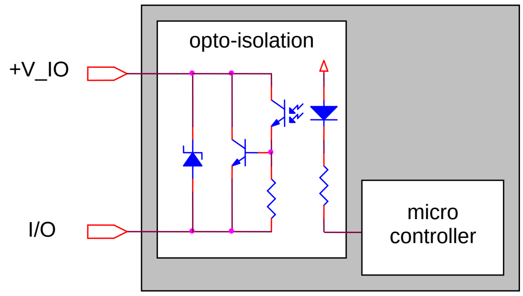
Functional diagram:



`#INPUT_ANALOG` indicates differential voltage between pins +IANA and –IANA (expressed in mV)

Input voltage range / 0Vana	0 to 35V
Differential input voltage range	±35V
Cutoff frequency	1KHz
Differential input impedance	220KOhms
Precision	±3%
Resolution	8.6mV

3.3.3. Optically isolated outputs

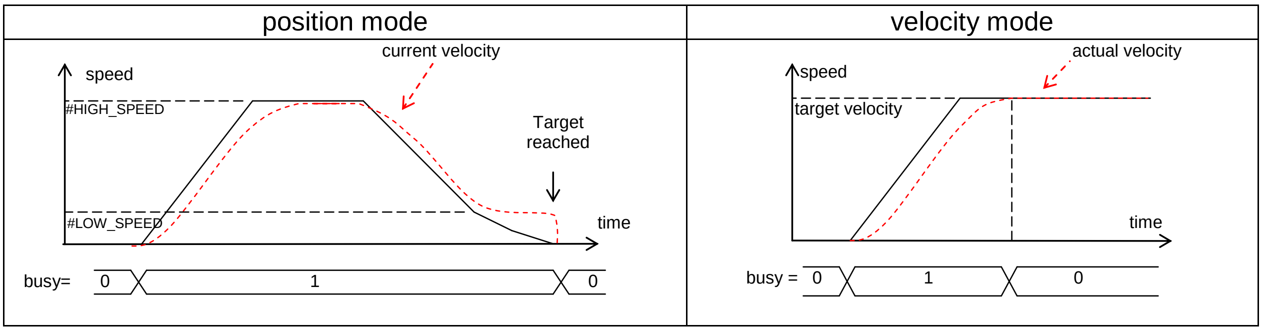


	min	max
Io_{off}		0.1mA
Vo_{on}		0.6V @1mA 1.1V @5mA 1.3V @50mA

Max output current 50mA
Max output voltage 30V
+VIO voltage is common to all digital outputs.
Logical 1 (`#OUTPUT:=1`) optocoupler is ON (driving).
Logical 0 (`#OUTPUT:=0`) optocoupler is OFF (released).

Logical output can be set using #OUTPUT and read using READ #OUTPUT

OUT1 can reflect BUSY signal if corresponding bit of #OUTPUT_CONFIG is set to 1. BUSY signal is active as long as target is not reached (target velocity in velocity mode or target position in position mode). See diagram:



OUT2 can reflect FAULT signal if corresponding bit of #OUTPUT_CONFIG is set to 1. FAULT signal is active when one of the following events occurs:

- Motor overcurrent disjunction
- Differential disjunction
- Overvoltage disjunction
- Undervoltage disjunction
- Thermal disjunction (motor or electronics above 85°C or CPU above 120°C)

3.4. Visualization

Three leds indicate BMAC internal status :

- Yellow "Busy" led shows motor activity: current motion or sequence.
- Red "Fault" led indicates fault state: overvoltage, undervoltage, thermal disjunction or motor disjunction. Moreover, this led flashes at module power-up or reset.
- Green "Power" led shows that BMAC is powered-up. Warning: the led does not indicates that supply voltage is inside the tolerance range (it could either be too low or too high). Turns orange in case of a short-circuit on +5V COD output.

3.5. Configuring COM port

BMAC implements both USB and RS485.

USB is to be used with single-axis applications in a low-noise environment. RS485 protocol is to be used with multi-axis applications and offers better noise immunity.

An application note on the USB driver installation is available on www.midi-ingenierie.com

3.5.1. Setting-up the board

BMAC should be setup with the following parameters:
`SET_BAUDRATE` (transfer speed in bauds 9600, 19200, 38400 or 115200)
`SET_ADDRESS` (module address 0 to 63)
`#LINE_DELAY` (RS485 turn-around delay in microseconds, 100 to 3000)

Factory default values are:

```
SET_BAUDRATE 38400
SET_ADDRESS 0
#LINE_DELAY:=3000
```

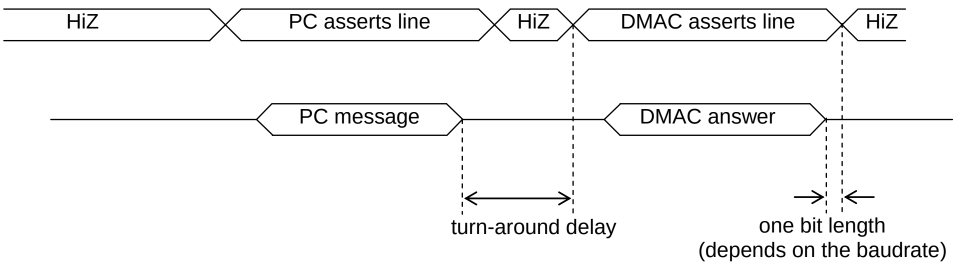
3.5.2. RS-485 protocol

What is RS485?

RS-485 allows multiple devices (up to 32) to communicate at half-duplex on a single pair of wires, plus a ground wire, at distances up to 1200 meters (Both the length of the network and the number of nodes can easily be extended using repeaters).

How does the hardware work?
Data is transmitted differentially on two wires twisted together, referred to as a "twisted pair", which provides the modules with high noise immunity and long distance. Emitters and receivers are all connected on the same bus. Only one device can drive the line at a time, so drivers must be put into a high-impedance mode (tri-state) when they are not in use.

How does the software work?
The half-duplex tri-state mode necessitates insertion of a delay between asserting the line and transmitting the data on the line, as well as between end of transmission and line release. Those delays are handled by BMAC firmware and control software (PC or PLC). Line handling is directly controlled by software when using drvMI dll or Winsim2.



How to communicate with the module?

WinSim2 Windows-based GUI application for BMAC and Midi Ingenierie products. Offers various visualization and control tools.
DrvMi.dll Dynamic library. Can be used within user applications (C/C++, VisualBasic, Delphi, etc…)

4. Commands

Command frames are sent to the module via a character string composed of the address, the command and parameters.

Entire command name (e.g.: `MOVE_TO`) or its mnemonic (e.g.: `MTO`) may be used.

Module address is given by the first two characters of the frame.
If the address is omitted, the command is "global" and shall be executed by all the modules. It is acknowledged only by the module at address 0.

Several commands can be sent within a single frame, separated by comas
(e.g.: "`#HIGH_SPEED:=10000, MOVE_TO 1234`").
Total frame length should not exceed 256 characters.

A space character should be inserted between command and parameter.

Some commands can be used only within a sequence or only in "live" (not within a sequence). Some others can be used in both modes.

In this manual, the following symbols are used:
@ represents module address
[ ] represents optional parameter

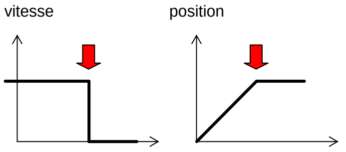
4.1. CALL / RETURN

Syntax:	CALL parameter RETURN	
Mnemonic:	CAL RET	
Parameters:	Number of the first line of the function 1 < parameter < 500	
Description:	Call of a subroutine. A subroutine must end with the RETURN command. Sequence then jumps back to line following CALL command.	
Notes:	Up to 5 subroutines calls can be nested.	
Example:	`:4  CALL 12`	Call of the subroutine at line 12
	`    NOP`	
	`    NOP`	
	`:12 NOP`	Line 12: start of the subroutine
	`    NOP`	
	`    RETURN`	Returns to line 5 (following CALL command)

4.2. CLOSE_LOOP

Syntax:	[@]CLOSE_LOOP parameter
Mnemonic:	CLO
Parameter:	ON (close loop, self-switched) or OFF (open loop)
Description:	Select the BMAC functioning mode. By default (output factory or command MODULE_RESET ALL), BMAC is in open loop. This parameter is reflected in bit #STATUS.28 **This parameter is stored in non-volatile memory.**
Example:	`03CLOSE_LOOP ON` Active the close loop of module 03.

4.3. HALT

Syntax:	[@]HALT [parameter]
Mnemonic:	HAL
Parameter:	None, MOUV ou SEQ
Description:	Module stops without deceleration ramp. Barquing torque is maximum, deceleration is determined by holding torque and load inertia. Position tracking is still active.  HALT command without parameters stops both movement and sequence. HALT MOUV stops only movement but keeps sequence. HALT SEQ stops the sequencer but has no effect on the movement.
Example:	`05HALT` Axis 05 stops immediately. If a sequence is running it is also stopped.

4.4. HARD_ENDS

Syntax:	[@]HARD_ENDS parameter
Mnemonic:	HEN
Parameter:	ALL (enables hardware end-stops) or OFF (disables hardware end-stops) POS (enables positive end-stop only) NEG (enables negative end-stop only)
Description:	Enables or disables hardware end-stops. Positive end-stop stops and prevents forward (CW) motion. Negative end-stop stops and prevents backward (CCW) motion. **This parameter is automatically stored at shut-down.**
Notes:	Polarity inversion (INVERSE_POLARITY command) applies to hard-ends input as well as regular inputs.
Example:	`03HARD_ENDS ALL` Enables hardware end-stops on module 03.

4.5. IF

Syntax:	IF test JUMP parameter IF test CALL parameter IF test JUMP_REL parameter
Mnemonic:	IF
Parameters:	"test" is a logical operation having TRUE or FALSE result. "parameter" is the line number to jump to if test result is TRUE (otherwise, sequencer goes on incrementing line number). [0 ; 500]
Description:	Conditional jump. Linear execution of the sequence is disrupted. Jump type can be one of `JUMP`, `CALL` or `JUMP_REL` (see commands details)
Example:	`:10 IF #POSITION > 1234 JUMP 56` If current position is greater than 1234, sequencer will execute line 56. Otherwise, execution will continue at line 11.

4.6. INVERSE_POLARITY

Syntax:	[@]INVERSE_POLARITY parameter
Mnemonic:	IPO
Parameter:	OFF (inputs and outputs are not inverted) ALL (inputs and outputs are inverted) IN (inputs are inverted) OUT (outputs are inverted)
Description:	Specifies if physical state is inverted vs logical state of `#INPUT` et `#OUTPUT` variables. Inputs: Standard: active physical state ⇔ `#INPUT` is 1 Inverted: active physical state⇔ `#INPUT` is 0 Outputs: Standard: `#OUTPUT` is 1 ⇔ active physical state Inverted : `#OUTPUT` is 0 ⇔ active physical state
Example:	`03INVERSE_POLARITY OUT` From now on, physical outputs will be inverted vs `#OUTPUT`.

4.7. JUMP / JUMP_REL

Syntax:	JUMP param1 JUMP_REL param2	
Mnemonic:	JUM JRE	
Parameter:	0 <= param1 < 500 -500 < param2 < 500	line number to execute to JUMP line number to jump to JUMP_REL
Description:	Sequence line jump. Linear execution is disrupted to jump to a given line. `JUMP` command jumps to a given line whatever the current line is, whereas `JUMP_REL` jumps to an offset from current line. `JUMP_REL` can be either positive or negative (backward jump).	
Notes:	`JUMP 0` blocks program execution.	
Example:	`JUMP 10`	Jump to line 10.
	`:15 JUMP_REL -1`	Loop to previous line (line 14).
	`JUMP_REL +2`	Jumps two lines forward.

4.8. MODULE_RESET

Syntax:	[@]MODULE_RESET [parameter]
Mnemonic:	MRE
Parameter:	None or ALL => reset factory settings.

Description:	Motor is powered off and then stopped (short-circuited) until complete stop.
	MODULE_RESET command is then similar to power-off/ power-on.
	Outputs are forced inactive. Volatile variables are (#V1 to #V32) are set to 0. If a sequence was configured to be executed at reset, it is launched.
	
	Parameter ALL restores factory settings (except serial line configuration):
	`#HIGH_SPEED := 60000`
	`#LOW_SPEED := 6000`
	`#CURRENT_RATIO := 500`
	`#ACCEL_TIME := 1000`
	`#DECEL_TIME := 1000`
	`#POSITIVE_END := +10000`
	`#NEGATIVE_END := -10000`
	`#POSITION := 0`
	`#OUTPUT_CONFIG := 3`
	`#ON_RESET := 0`
	Memorized variables (#M1 to #M8) are set to 0.
	Standard I/Os polarity is restored (`INVERSE_POLARITY OFF`).
	End-stops are disabled (`HARD_ENDS OFF; SOFT_ENDS OFF`).
	Motion configuration is restored `(S_CURVE OFF,OPTIMIZED_CURRENT OFF).`
	
	Note : Serial line configuration (baudrate, address and `#LINE_DELAY`) are not affected by MODULE_RESET ALL command

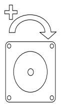
Example:	`02MODULE_RESET` Resets module 2.

4.9. MOVE_INTERPOL

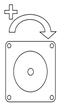
Syntax:	[@]MOVE_INTERPOL parameter
Mnemonic:	MIN
Parameter:	Position offset (relative) (in motor increments: 10^{-4} rev.) to be executed during `#INTERPOL_TIME` . [-2 147 483 648; +2 147 483 647]
Description:	In interpolation mode, each axis describes a trajectory defined by a succession of "segments". A segment is defined by the position offset to be executed in a given time period (`#INTERPOL_TIME`). At each `MOVE_INTERPOL` command, one segment is stored in a FIFO buffer (which size is defined by `#INTERPOL_FIFOSIZE` variable) Movement is launched by global command "`SYNCHRO INTERPOL`". All axes execute previously stored segments in a synchronized way. (see `#INTERPOL_MODE` variable for details on multi-axis synchronization). When FIFO is full, `MOVE_INTERPOL` command is rejected by emission of XON_ERROR (17h) character. When FIFO is empty (the entire trajectory has been described) movement is stopped and "`SYNCHRO INTERPOL`" command is necessary to start another interpolated movement. During the movement, user is in charge of supplying the modules with segments at a higher rate than trajectory execution.
Notes:	MIShell (PC application) can handle automatic command re-send in case of FIFO-full error. To enable this feature, one should add underscore character before the command: `_01MOVE_INTERPOL 200`
Example:	Sample commands for 3 axis interpolation:

```
#INTERPOL_FIFOSIZE:=64    ;FIFO contains 64 segments
#INTERPOL_TIME:=100       ;one segment per 100ms
#INTERPOL_MODE:=0         ;SimpleSYNC mode

00MOVE_INTERPOL 100       ;1st segment
01MOVE_INTERPOL 750       ;(stored in FIFO but not
02MOVE_INTERPOL 100       ;launched yet)

SYNCHRO INTERPOL          ;synchronized start

00MOVE_INTERPOL 100       ;2nd segment
01MOVE_INTERPOL -50
02MOVE_INTERPOL 100

…                         ;3rd segment…
```

4.10. MOVE_ON

Syntax:	[@]MOVE_ON parameter
Mnemonic:	MON
Parameter:	Position offset in motor increments (10^{-4}rev.) [-2 147 483 648 ; + 2 147 483 647]
Description:	Motor rotates by a given amount of increments (position offset). Motion follows pre-defined velocity profile (#ACCEL_TIME, #DECEL_TIME and #HIGH_SPEED variables). Movement direction is defined by parameter sign, positive parameters resulting in clockwise rotation (front view). 
Notes:	MOVE_ON forces motor power ON.
Example:	03MOVE_TO -5000 Module 03 moves by half a rev. counter-clockwise.

4.11. MOVE_SPEED

Syntax:	[@]MOVE_SPEED parameter
Mnemonic:	MSP
Parameter:	Target velocity expressed in 0.01RPM. [-400000 ; 400000]
Description:	Motor rotates at a given speed. Movement direction is defines by parameter sign, positive parameters resulting in clockwise rotation (front view). Velocity profile goes from current velocity to target velocity in a time defined proportionally to #ACCEL_TIME and #DECEL_TIME variables.
Notes:	 Target velocity is limited (in absolute value) to #HIGH_SPEED variable. Parameters superior to #HIGH_SPEED will be executed at #HIGH_SPEED velocity. MOVE_SPEED forces motor power ON.
Example:	00MOVE_SPEED 50000 Module 00 rotates at 500RPM clockwise.

4.12. MOVE_TO

Syntax:	[@]MOVE_TO parameter
Mnemonic:	MTO
Parameter:	Target position (10^{-4}rev.) [-2 147 483 648 ; + 2 147 483 647]
Description:	Motor moves from current position to target position. Velocity profile is defined by `#ACCEL_TIME`, `#DECEL_TIME` et `#HIGH_SPEED`.
Note:	If motor is already at target position, no movement is performed. This command forces motor power-on.
Example:	`00MOVE_TO 123456` Motors 00 goes to position +123456

4.13. OPEN_SEQ / CLOSE_SEQ

Syntax:	[@]OPEN_SEQ [@]CLOSE_SEQ
Mnemonic:	OSE CSE
Parameter:	None
Description:	OPEN_SEQ enable sequence edition. CLOSE_SEQ disables sequence edition. Bit STATUS.16 is set to 1 to indicate that sequence edition is enabled. In sequence edition, commands are not executed but stored in the module for future execution. Sequence starts at line number 1 (default) but user can force edition of a given sequence line by preceding command by ":n" (n being the sequence line number) `OPEN_SEQ` command erases any previously memorized sequence. Sequence edition remains enabled until `CLOSE_SEQ` command is sent to the module.
Example:	`OPEN_SEQ` Enable Edit mode. `NOP` `NOP` Commands are memorized in the sequencer `NOP` `CLOSE_SEQ` Disable Edit mode.

4.14. OPTIMIZED_CURRENT

Syntax:	[@]OPTIMIZED_CURRENT parameter
Mnemonic:	OCU
Parameter:	ON (optimized mode) ou OFF (default mode)
Description:	Enables or disables "optimized current" feature. When enabled, motor current is automatically adjusted to provide the necessary torque. This feature lessens thermal losses when motor does not run continuously. **This parameter is automatically stored at shut-down.**
Notes:	This mode needs autocommutation, it has no action if BMAC is set on open loop.
Example:	`06OPTIMIZED_CURRENT ON` Enables optimized current feature for module 06.

4.15. POWER

Syntax:	[@]POWER parameter
Mnemonic:	POW
Parameter:	ON,OFF,SC (short-circuiting motor coils)
Description:	POWER ON applies power to motor coils. POWER OFF implies no current and no torque. POWER SC generates low-speed resisting torque.
Notes:	`POWER ON` is automatically performed before any MOVE command.
Example:	`02POWER ON` Enable power for module 2.

4.16. READ

Syntax:	@READ parameter
Mnemonic:	REA
Parameter:	Variable name (variables are described next)
Description:	Sends variable value over serial line toward PC (or PLC). Use 'H' prefix in front of variable name to read hexadecimal value. Use 'B' prefix in front of variable name to read binary value. H et B can either be capital letters or small characters.
notes:	Hexadecimal and binary values are 4 bytes signed. `READ` command cannot be used within a sequence. Character string sent back by the module is composed of the module address (2 chars), the mnemonic of the variable, the '=' character and the value of the variable.

Example:	`00READ #POSITION`
	reads module 00 current position:
	module sends: `00#POS=12345`
	
	`01READ h#OUTPUT`
	reads module 01 digital outputs (hexadecimal format):
	module sends: `01#OUT=h00000007`
	=> Outputs OUT1, OUT2 and OUT3 are active, OUT4 is inactive (provided `INVERSE_POLARITY` is OFF).
	
	`02READ b#INPUT`
	reads module 02 digital inputs (binary format):
	module sends: `02#INP=b00000000 00000000 00000000 00001010`
	=> Inputs IN2 and IN4 are active, inputs IN1, IN3,IN5 are IN6 inactive (depending on `INVERSE_POLARITY`).

4.17. READ_SEQ

Syntax:	[@]READ_SEQ parameter
Mnemonic:	RSE
Parameter:	address of sequence line to read [1 ; 500]
Description:	Reads one line of a sequence. Useful for sequence programming. Warning, syntax can be slightly different from the one used in sequence edition.
Example:	`00READ_SEQ 3` Reads line 3 Module sends: `00:003 MTO +2000` Line 3 contains command "`MOVE_TO 2000`"

4.18. REFERENCE

Syntax:	[@]REFERENCE parameter
Mnemonic:	REF
Parameter:	ON (reference enabled on IN5 input) INDEX (reference enabled on encoder index input) OFF (reference disabled)
Description:	Enable/disable "reference mode". REFERENCE ON Reset position on IN5 input: `#POSITION` and `#ENCODER` are set to zero if motor is running clockwise and a rising edge (0 to 1) is detected on IN5 input. Reference mode (bit `#STATUS.30`) is automatically disabled once position has been initialized. Warning: IN5 must be activated during at least 1ms minimum for this operation. REFERENCE INDEX Reset position on encoder index input: `#POSITION` and `#ENCODER` are set to zero if a rising edge is detected on encoder index input. Reference mode (bit `#STATUS.30`) is automatically disabled once position has been initialized. Warning: index input must be activated during at least 50µs for this operation. ⇨ Reference searching at 600rpm max for a 500 points encoder. ⇨ Reference searching at 30rpm max for a 10000 points encoder.
Example:	`06REFERENCE INDEX` Enable index reference mode on module 06.

4.19. REQUEST_VERSION

Syntax: [@]REQUEST_VERSION

Mnemonic: RV OU RVE

Parameter: None

Description: Reads module identification string.
Module answers :
@EV vV.RR *CODE* "MIDI-INGENIERIE_*product_serial_manufacturing-date_revision-date*" PHASE:XX BOOT:vX.Y
where V = software version,
RR = software release,
CODE = software ID
dates format: dd/mm/yy.

Software code detail:

CODE	SOFTWARE
9189	RS485
J189	clock and dir
K189	CAN

Example: `04 REQUEST_VERSION`
sample module answer:
04EV v3.00 9189 "MIDI-INGENIERIE_BMAC_9189-0014_25/09/11_12/10/11" BOOT:v2.0
Module is a BMAC
Serial number is 9189-0014
Manufacturing date: 25/09/11
Revision date : 12/10/11.
Firmware : 9189 standard
Version release : 3.00

4.20. SET_ADDRESS

Mnemonic:	SAD parameter
Parameter:	New module address [0 ; 63] Factory default value = 0.
Description:	Modifies module address. Global commands are executed by all the modules but only the module at address 0 can send an answer. **Module address is stored in non-volatile memory.**
Notes:	Two modules cannot have the same address (conflict). It is the responsibility of the user to handle correct module addressing. SET_ADDRESS command should not be sent in a global way (without specifying targeted module). WARNING: Factory default address is 0 for all modules. One should set address of a module BEFORE connecting it to a bus.

Example:	`04SET_ADDRESS 3`
	module 4 becomes module 3
	
	`04SET_ADDRESS 0`
	module 4 becomes module 0. From now on, it answers global commands.
	
	`SET_ADDRESS 6`
	Warning: global command! All modules are set to address 6. Conflict!

4.21. SET_BAUDRATE

Mnemonic:	SBA parameter
Parameter:	defines communication speed in bauds (bits per seconds) 115200, 38400, 19200, 9600 (all other values will not be executed) Factory default value = 38400.
Description:	Specifies communication speed between host (PC or PLC) and the modules. **Baudrate is stored in non-volatile memory.**
Notes:	Baudrate should be the same for all modules. `SET_BAUDRATE` command should be sent in a global way (without specifying targeted module). New baudrate is applied immediately and does not require a reset.
Example:	`SET_BAUDRATE 38400` From now on, modules will communicate at 384000 bauds.

4.22. S_CURVE

Syntax:	[@]S_CURVE parameter
Mnemonic:	SCU
Parameters:	ON (S-curve ramp) or OFF (trapezoidal ramp)
Description:	Enable/Disable S-curve velocity ramp. Compared to trapezoidal velocity ramp, S-curve results in a smoother movement, minimizing acceleration and torque discontinuity. **This parameter is stored in non-volatile memory.**
Notes:	To disable velocity ramps ("start-stop"), one should set `#ACCEL_TIME and #DECEL_TIME` to zero.
Example:	`05S_CURVE OFF` Module 5 will use trapezoidal ramps for all movement.

4.23. SOFT_ENDS

Syntax:	[@]SOFT_ENDS parameter
Mnemonic:	SEN
Parameters:	ALL (enable end-stops) or OFF (disable end-stops)
Description:	Enable/Disable software "virtual" end-stop. `#POSITIVE_END` defines maximal position (CW direction). `#NEGATIVE_END` defines minimal position (CCW direction). Beyond these two positions, all movement is stopped. **This parameter is stored in non-volatile memory.**
Example:	`02SOFT_ENDS OFF` Disable software end-stops for module 02.

4.24. START_SEQ

Syntax:	[@]START_SEQ [parameter]
Mnemonic:	SSE
Parameters:	Number of the first line to be executed (default: 1).
Description:	Execution of the sequence from specified line. If no parameter is given, execution starts at line 1.

Example:	`START_SEQ 15`
	All modules start sequencer from line 15.
	
	`02START_SEQ`
	Module 2 start sequencer from line 1.

4.25. STEP

Syntax:	[@]STEP [parameter]
Mnemonic:	STE
Parameters:	Sequencer line number to be executed.
Description:	Sequencer debug mode: If parameter > 0, specified sequencer line will be executed. If parameter is not specified, following sequencer line will be executed (#LINE is automatically incremented).
Example:	`04STEP 25` Module 4 executes sequencer line 25.

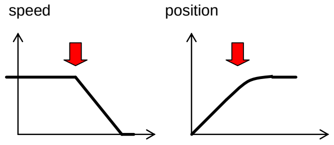
4.26. STOP

Syntax: [@]STOP [parameter]

Mnemonic: STO

Parameters: None, MOUV or SEQ

Description: Module stops, going from current speed to zero speed. Deceleration time is proportional to `#DECEL_TIME`.



Command STOP without any parameter stops both movement and sequence.
"STOP MOUV" stops only the movement but keeps sequence running.
"STOP SEQ" stops only the sequence but continues current movement.
Note: In interpolated mode, STOP command is equivalent to HALT (immediate stop without deceleration) and interpolation FIFO is emptied.

Example: `04STOP`
Module 04 decelerates until complete stop. If a sequence is running, it is stopped.
`03STOP MOUV`
Module 03 decelerates until complete stop. If a sequence is running, it is not stopped.

4.27. SYNCHRO

Syntax:	SYNCHRO [parameter]
Mnemonic:	SYN
Parameters:	ON, OFF, TOP or INTERPOL
Description:	This command allows multi-axis synchronization. `SYNCHRO ON` enables synchro mode (useless for interpolation mode) `SYNCHRO OFF` disables synchro mode `SYNCHRO TOP` launch movement (except interpolation) `SYNCHRO INTERPOL` launch movement (interpolation) Note: Power is automatically switched on at the beginning of a movement. For better synchronization, it is recommended to switch it on before starting the movement (`POWER ON`).

Example:	<u>1) standard movement (non interpolated)</u>
	
	`SYNCHRO ON` ; enable synchro mode
	`00MOVE_SPEED 4000`
	`03MOVE_SPEED -12500`
	`SYNCHRO TOP` ; launch movements
	`SYNCHRO OFF` ; disable synchro mode
	
	<u>2) interpolation</u> (see MOVE_INTERPOL command at chapter 4.4)

4.28. WAIT

Syntax:	WAIT parameter
Mnemonic:	WAI
Parameters:	waiting time in milliseconds [-3600000 ; 3600000]
Description:	Execution of a sequence is suspended until delay is over. If parameter is 0 (e.g. "`WAIT 0`"), execution is suspended until current movement is completed. If parameter is negative (e.g. "`WAIT -1000`"), execution is suspended until current movement is completed, with a timeout specified by the parameter.

Exemple:	`WAIT 2000`
	Sequencer waits 2 seconds before going on to next line.
	
	`WAIT 0`
	Sequencer waits until movement is completed before going on to next line.
	
	`WAIT -2000`
	Sequencer waits for movement to be completed for a maximum of 2 seconds before going on to next line.

5. Internal variables

5.1. Syntax

Standardized syntax is as follows:

Writing a variable:	[@]#VARIABLE[.BIT]:= [- ! H B]VALUE
Reading a variable:	[@]READ [H B]#VARIABLE
Module answer:	@MNEMONIC=VALUE

All BMAC variables are stored in a "32 bit signed" form.

5.1.1. Decimal form

By default, values are expressed in decimal form.
Character "+" before positive values is optional but always specified in modules answers.

Examples:

```
00#ACCEL_TIME:=123
00#V20:=-40
00READ #ACCEL_TIME → 00#ATI=+123
```

5.1.2. Hexadecimal form

A preceding "H" or "h" character shall be used to type and read hexadecimal values.
Negative values are represented in 4 bytes two's complement form (e.g. -10 = hFFFFFFF6).
Leading zeros are optional, but all values are read as 4 bytes (8 digits).

Examples:

```
00#ACCEL_TIME:=H100
00#V20:= HFFFFFFD8
00READ h#ACCEL_TIME → 00#ATI=h00000100
```

5.1.3. Binary form

A preceding "B" or "b" character shall be used to type and read binary values.
Negative values are represented in 4 bytes two's complement form.
Leading zeros are optional, but all values are read as 4 bytes (separated by a space).

Examples:

```
00#ACCEL_TIME:=B1100100
00READ b#ACCEL_TIME → 00#ATI=b00000000 00000000 00000000 01100100
```

5.1.4. Opposite and complement

A preceding "-" character shall be used to access the opposite of a variable.
A preceding "!" character shall be used to access the binary complement of a variable.
This notation is especially useful for writing sequences.

Examples:

```
00#V21:=-#V1
00#V23:=#V15 & !#STATUS
00MOVE_TO -#POSITION
```

5.1.5. Bit form

To read or write one bit of a variable, the name of she variable should be followed by a point ant the number of the bit (1 being the LSB, 32 being the MSB).
Bit value is then considered as a variable that can have value 1 or 0.
Bit form can be used in a command or a test.

Examples:
```
00#V10.3:=1                         ;sets third bit of #V10 to 1
00#V10.12:=0                        ;sets 12th bit of #V10 to 0
00READ #STATUS.5 → 00#STA.5=1       ;reads 5th bit of #STATUS
00#V12.1:=#STATUS.12 & #ERROR.2     ;logical AND between two bits
00IF #V1.24 = 1 JUMP 3              ;test if bit 24 of #V1 is set
```

5.1.6. Operations on variables

Several operations can be performed using variables:
[@]#VARIABLE:=OPERAND1 OPERATOR OPERAND2

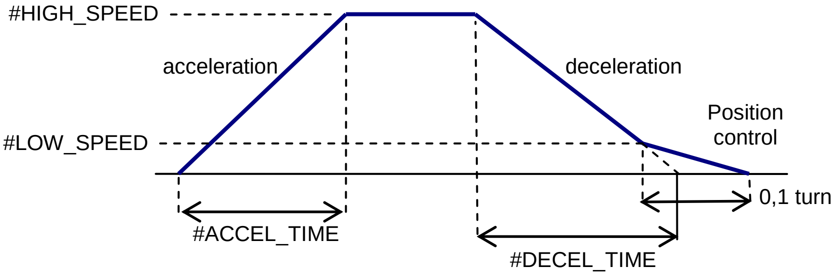
There shall be at least one space character before and after the operator.
An operands can be a variable or a constant (decimal, hexadecimal, binary or bit form) .

The operand must be one of the following::

<table>
<tr><th>operator</th><th>operation</th><th>example</th></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td>+</td><td>addition</td><td>#V1:=#POSITION + 12000</td></tr>
<tr><td>-</td><td>subtraction</td><td>#LOW_SPEED:=#HIGH_SPEED - #V3</td></tr>
<tr><td>*</td><td>multiplication</td><td>#V2:=3 * -#SUPPLY_VOLTAGE</td></tr>
<tr><td>/</td><td>integer division</td><td>#M3:=#POSITION / 10000</td></tr>
<tr><td>&</td><td>bitwise AND</td><td>#V2:=#STATUS & H00000200</td></tr>
<tr><td>|</td><td>bitwise OR</td><td>#M1:=!#V4 | H00000240</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td>></td><td>test superior</td><td>#V1:=#POSITION > 123</td></tr>
<tr><td><</td><td>test inferior</td><td>#V1:=#POSITION < 123</td></tr>
<tr><td>>=</td><td>test superior or equal</td><td>#V1:=#POSITION >= 123</td></tr>
<tr><td><=</td><td>test inferior or equal</td><td>#V1:=#POSITION <= 123</td></tr>
<tr><td>!=</td><td>test different</td><td>#V2:=#POSITION != 0</td></tr>
<tr><td></td><td></td><td></td></tr>
</table>

Result of test operations is 1 if the test is TRUE and 0 if the test is FALSE. Test operations are to be used within sequences for conditional jumps (IF…JUMP).

5.1. #ACCEL_TIME, #DECEL_TIME

Mnemonic	#ATI, #DTI
Parameter:	Acceleration and deceleration time, expressed in milliseconds. [0 ; 12000]
Description:	#ACCEL_TIME: time from standstill to #HIGH_SPEED #DECEL_TIME: time from #HIGH_SPEED to standstill.  **This parameter is stored in non-volatile memory.**
Notes:	Variable `#ACCEL_TIME` represents the time to go from standstill (stopped) to target velocity `#HIGH_SPEED`. Variable `#DECEL_TIME` represents the time to go from target velocity `#HIGH_SPEED` to standstill. If current velocity or target velocity are different from standstill or #HIGH_SPEED, acceleration and deceleration time is automatically computed proportionally to speed difference, so as to obtain constant accelerations. $\text{acceleration time} = \#ACCEL\_TIME \times \frac{\lvert \text{Target velocity - Current velocity} \rvert}{\#HIGH\_SPEED}$ $\text{deceleration time} = \#DECEL\_TIME \times \frac{\lvert \text{Target velocity - Current speed} \rvert}{\#HIGH\_SPEED}$ Example: `#ACCEL_TIME:=1000` (1s) `#HIGH_SPEED:=60000` (600tr/min) `MOVE_SPEED 30000` → computed acceleration time: 0.5s Note: Those parameters set the module deceleration.
Example:	`00#ACCEL_TIME:=1000` Module will go from standstill to `#HIGH_SPEED` in one second.

5.2. #CAPTURE (read only)

Mnemonic	#CAP
Parameter:	Value of #POSITION at last IN5 edge.
Description:	At each rising edge (0→1) of IN5 digital input, current position is stored in #CAPTURE variable.
Example:	03READ #CAPTURE → 03#CAP=27895 Module 3 was at position 27895 at IN5 rising edge.

5.3. #CURRENT_RATIO

Mnemonic	#CRA
Parameter:	Motor current, (expressed in 1/1000) of the nominal current #RATED_CURRENT [0 ; 1000] Injected current in mA is : #RATED_CURRENT * #CURRENT_RATIO /1000
Description:	This variable allows setting the injected current in the motor. A high value gives a higher holding torque, a lower value allows reducing power consumption and heat losses in the motor. **This parameter is stored in non-volatile memory.**
Notes:	The user should configure #RATED_CURRENT prior to specifying #CURRENT_RATIO. Max performance (speed and power) require #CURRENT_RATIO set to 1000. The current is generated with steps of 25mA for BMAC and 70mA for BMAC-H In self-switched mode, the OPTIMIZE_CURRENT command enable the user to minimize the current in the motor if possible, so as to lessen thermal losses and power consumption.
Example:	00#CURRENT_RATIO:=750 Motor connected to module 0 has 750/1000, that is 75% of its nominal current.

5.4. #CPU_TEMPERATURE (read only)

Mnemonic	#CTE
Parameter:	Internal microcontroller temperature (units = 0,1°C)
Example:	`02READ #CPU_TEMPERATURE` → `02#CTE=520` CPU has a temperature of 52°C on module 2.

5.5. #DRIVER_TEMPERATURE (read only)

Mnemonic	#DTE
Parameter:	indicates BMAC power stage temperature (units = 0,1°C)
Exemple:	`02READ #DRIVER_TEMPERATURE` → `02#DTE=650` Motor on BMAC at address 2 is 65°C hot.

5.6. #ELECTRICAL_POSITION (read only)

Mnemonic	#EPO
Parameter:	Indicates the electrical position [0 ; 199]
Description:	`#ELECTRICAL_POSITION` is the electrical target position in microsteps (BMAC resolution is 50µstep/step). Target phase currents are given by : $I_{phaseA} = I_{peak} * \cos(2\pi * (\#ELECTRICAL\_POSITION / 200))$ $I_{phaseB} = I_{peak} * \sin(2\pi * (\#ELECTRICAL\_POSITION / 200))$ With $I_{peak} = \sqrt{2} * \#RATED\_CURRENT * \#CURRENT\_RATIO$
Notes:	**Module electrical position is stored in non-volatile memory.** Full steps correspond to the values 0, 50, 100 and 150 of `#ELECTRICAL_POSITION`.
Example:	`00READ #ELECTRICAL_POSITION` → `00#EPO=0` ⇨ A phase current is maximum, B phase current is 0.

5.7. #ENCODER

Mnemonic	#ENC
Parameter:	New position (1/2000 turn if encoder is 500points) [-2 147 483 648 ; + 2 147 483 647]
Description:	Initialize the encoder follow position. Command particularly used to define the mecanic system origin, giving 0 as parameter.
Example:	`04#ENCODER:=2000` Encoder value set to +2000.

5.8. #ERROR

Mnemonic #ERR

Parameter: None

Description: BMAC error register:

MSB	bit 32	
	bit 31	BMAC-H amplifier error (power supply < 20V or short-circuit disjunction or differential or thermal)
	bit 30	Differential disjunction (short-circuit between phase and 0V)
	bit 29	Short-circuit disjunction of a motor phase
	…	
	bit 25	Thermal disjunction power stage > 85°C
	…	
	bit 12	Command or Variable is undefined (syntax)
	…	
	bit 9	
	bit 8	Computational error (divide by zero…)
	bit 7	Limit (parameter beyond limit)
	bit 6	
	bit 5	Overvoltage
	bit 4	Under voltage
	bit 3	Short-circuit (detail in bits 29 and 30)
	bit 2	Thermal disjunction
LSB	bit 1	

Error acknowledgement is done by writing 0 in `#ERROR`

Example:

```
01READ b#ERROR
→  01#ERR=b00000000 00000000 00000000 00010000
```

Reading error register of module 1 (binary form) → overvoltage detected.

5.9. #HIGH_SPEED

Mnemonic	#HSP
Parameter:	Target velocity (expressed in 0.01RPM) [0 ; 400000]
Description:	Target velocity for position mode movement (`MOVE_TO` and `MOVE_ON`) and maximal velocity for speed mode (`MOVE_SPEED`). **This parameter is stored in non-volatile memory.**
Notes:	If movement length is too short, target velocity may not be reached.
Example:	`04#HIGH_SPEED:=20000` Next movements will be done at 200RPM for module 4.

5.10. #INPUT (read only)

Mnemonic	#INP
Parameter:	Value representing digital inputs (binary active if 1, inactive if 0)
Description:	Digital inputs from IN1 (LSB) to IN6 or IN10 (MSB, depending on model). If polarity is inverted (command `INVERSE_POLARITY` represented by bit `#STATUS.13`), the logical state of `#INPUT` is inverted versus the physical state.

Example:	`BMAC at address 05` (with `INVERSE_POLARITY OFF`)
	
	`05READ #INPUT` → `05#INP=19`
	`05READ h#INPUT` → `05#INP=h13`
	`05READ b#INPUT` → `05#INP=b00000000000000000000000000010011`

19 (decimal) =	0	0	0	1	0	0	1	1	(binary)
	OFF	OFF	OFF	ON	OFF	OFF	ON	ON	
	IN8	IN7	IN6	IN5	IN4	IN3	IN2	IN1	

5.11. #INPUT_ANALOG (read only)

Mnemonic	#IAN
Parameter:	Analog input differential voltage in mV.
Example:	`02READ #INPUT_ANALOG` → `02#IAN=-3200` Module at address 02 has a voltage of -3.2V on its analog input.

5.12. #INTERPOL_COUNT (read only)

Mnemonic	#ICO
Parameter:	Number of segments in FIFO memory [0 ; #INTERPOL_FIFOSIZE]
Description:	This read only variable shows the number of memorized segments in FIFO memory. When FIFO is empty, #INTERPOL_COUNT=0 Movement is stopped after execution of the last segment. (see command MOVE_INTERPOL for detailed information on interpolation mode)
Notes:	Available space in FIFO is given by the difference: `#INTERPOL_FIFOSIZE - #INTERPOL_COUNT`
Example:	`02READ #INTERPOL_COUNT` pourrait renvoyer `02#ICO=23` => 23 segments are currently stored in axis 02.

5.13. #INTERPOL_FIFOSIZE

Mnemonic	#IFI
Parameter:	Size of the FIFO memory, expressed in segments [1 ; 64]
Description:	Parameter FIFO size between 1 and 64 segments. ((see command MOVE_INTERPOL for detailed information on interpolation mode)
Notes:	Setting a high value allows tolerates a certain latency between MOVE_INTERPOL messages. Setting a low value will be useful if a low response time is needed (for example when trajectory is computed in real-time).
Example:	`#INTERPOL_FIFOSIZE:=64` ; max value for all axis

5.14. #INTERPOL_MODE

Syntax: [@]#INTERPOL_MODE parameter

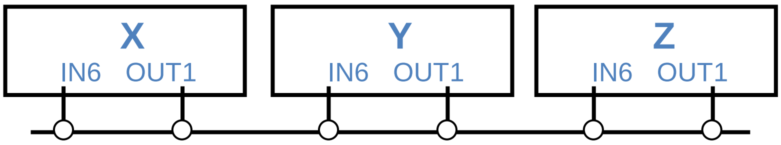
Mnemonic: #IMO

Parameter: 0 (SimpleSYNC) or -1 (UltraSYNC)

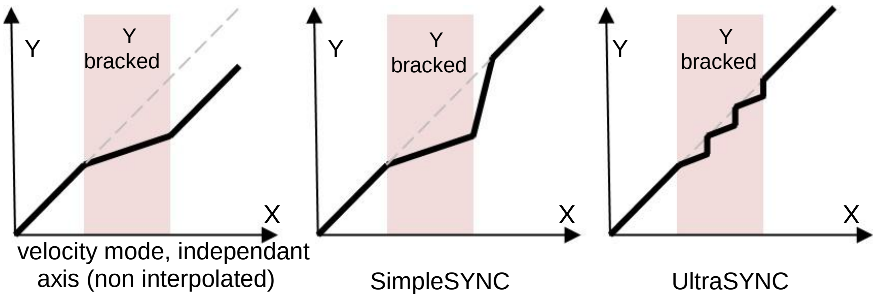
Description: This variable can choose between the two synchronization types for interpolation mode.

SimpleSYNC mode (`#INTERPOL_MODE:=0`)
Multi-axis synchronization is done by simultaneous start of the movement on all the modules (global command `SYNCHRO INTERPOL`). The duration of each segment is then assured by a precision internal timer.

UltraSYNC mode (`#INTERPOL_MODE:=-1`)
Multi-axis synchronization is done by chaining digital IOs IN6 and OUT1 for all the modules. If one of the axis goes slower than others (mechanical friction…) all the other axis are automatically informed and wait for the segment to be finished on all modules.



Example: Example on a 2 axis configuration X-Y:



5.15. #INTERPOL_TIME

Mnemonic	#ITI
Parameter:	Duration of one interpolation segment in ms [2 ; 138]
Description:	New length will be applied on all next `MOVE_INTERPOL` commands. (see command MOVE_INTERPOL for detailed information on interpolation mode) In most cases, all axis should use the same value.
Notes:	To keep the trajectory synchronized, #INTERPOL_TIME should be modified at the same time on all axis.
Example:	`#INTERPOL_TIME:=50` ; all future segments will last 50ms.

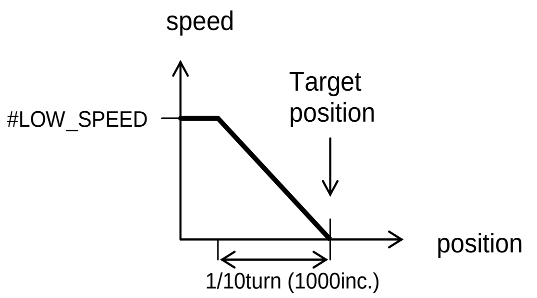
5.16. #LINE_DELAY

Mnemonic	#LDE
Parameter:	Turn-around delay in µs [100 ; 3000] Factory default value = 3000.
Description:	RS485 delay between the end of a command from host (PC or PLC) and the beginning of the answer from the DMAC. **This parameter is stored in non-volatile memory.**
Notes:	#LINE_DELAY should be set to the same value for all the axis.
Example:	`#LINE_DELAY:=1000` Module will wait for 1ms after having received a command before sending the answer.

5.17. #LINE

Mnemonic	#LIN
Parameter:	Sequencer line currently executed [0 ; 500]
Description:	Variable `#LINE` is the line executed by the module.
Notes:	When sequencer is not started, `#LINE`=0. Resetting this variable (`#LINE:=0`) stops the sequencer.
Example:	`READ #LINE` → `00#LIN:=182` Module at address 0 is executing line 182 of the sequence.

5.18. #LOW_SPEED

Mnemonic	#LSP
Parameter:	Approach speed in 100th rev./min [0 ; 400000]
Description:	#LOW_SPEED is used at the end of position mode movement (`MOVE_TO` and `MOVE_ON`). Velocity profile during position control loop is defined by `#LOW_SPEED`:  **This parameter is stored in non-volatile memory.**
Notes:	This variable permits fine tuning of the response time and stability of the position closed-loop control. Response time is shorter as #LOW_SPEED is set to a high value. On the other hand, if #LOW_SPEED is set too high, an oscillation can appear if load inertia is important, with little friction.
Example:	`04#LOW_SPEED:=2000` In position mode, Position control will go from 20RPM to standstill at the end of any movement.

5.19. #M1 to #M8

Mnemonic	#M1 to #M8
Parameter:	Value between -2^{31} and 2^{31} -1 [-2 147 483 648 ; + 2 147 483 647]
Description:	User variables: can be used to store any kind of value. `#M1` to `#M8` content is stored in non-volatile memory and restored at power-on, unlike `#V1` to `#V32` which content is reset.
Example:	`#M2:=H100` Store hexadecimal value 0x100 in variable `#M2`.

5.20. #NEGATIVE_END

Mnemonic	#NEN
Parameter:	Virtual negative end-stop (10^{-4}rev.) [-2 147 483 648 ; + 2 147 483 647]
Description:	All movement going beyond this position is stopped. #STATUS.20 bit is set and only CW movement is enabled. **This parameter is stored in non-volatile memory.**
Notes:	`#NEGATIVE_END` value is not modified when `#POSITION` is modified.
Example:	`03#NEGATIVE_END:=-80000` Module at address 3 can only go 8 rev. CCW away from home position.

5.21. #ON_RESET

Mnemonic	#ORE
Parameter:	Number of sequencer line to be executed on reset [0 ; 500]
Description:	Sequencer will automatically be launched, starting from specified line, on reset. To disable auto-start of the sequencer, set value to zero: `#ON_RESET:=0`.
Example:	`#ON_RESET:=1` Module executes sequencer, starting from line 1 at reset.

5.22. #OUTPUT

Mnemonic	#OUT
Description:	Digital outputs. If the polarity is inverted (command INVERSE_POLARITY represented by bit #STATUS.12), the logical state of #OUTPUT is inverted versus output physical state.
Notes:	All 32 bits of variable #OUTPUT can be written, but only LSB corresponds to physical outputs (#OUTPUT.1 to #OUTPUT.8). OUT1 and OUT2 can be used for alternate functions Busy and Default according to #OUTPUT_CONFIG variable. The physical state of the output is then determined by module status instead of #OUTPUT variable.
Example:	00#OUTPUT:=4 Activates OUT3, Deactivates all others

4 (decimal) =	0	0	0	0	0	1	0	0	(binary)
	OUT8	OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1	

5.23. #OUTPUT_CONFIG

Mnemonic	#OCO
Description:	Digital outputs multiplexing with alternate functions. Outputs 1 et 2 can be used as GPIO or as alternate functions: OUT1 can represent BUSY signal OUT2 can represent FAULT signal If #OUTPUT_CONFIG bit is 0, output is used as a GPIO. Output state is determined by corresponding #OUTPUT bit. If #OUTPUT_CONFIG bit is 1, output is used as an alternate function. Output state is determined by the status of the module. **This parameter is stored in non-volatile memory.**
Notes:	Default factory value for #OUTPUT_CONFIG is 3 (OUT1 and OUT2 represent BUSY and FAULT signals).
Example:	00#OUTPUT_CONFIG:=0 All outputs are used as GPIOs 00#OUTPUT_CONFIG:=1 OUT1 = BUSY, OUT2 = GPIO 00#OUTPUT_CONFIG:=2 OUT1 = GPIO, OUT2 = FAULT 00#OUTPUT_CONFIG:=3 OUT1 = BUSY, OUT2 = FAULT

5.24. #POSITION

Mnemonic	#POS
Parameter:	New position (10^{-4}rev.) [-2 147 483 648 ; + 2 147 483 647]
Description:	Force module position. Can be used to set home position (`#POSITION:=0`).
Notes:	**Module position is stored in non-volatile memory.** **The value may not be correct if the motor has moved when not powered.** Warning: do not set a position outside end-stops range (defined by #NEGATIVE_END and #POSITIVE_END) if they are enabled.
Example:	`04#POSITION:=0` Set new "home" reference position for module at address 4.

5.25. #POSITIVE_END

Mnemonic	#PEN
Parameter:	Virtual positive end-stop (10^{-4}rev.) [-2 147 483 648 ; + 2 147 483 647]
Description:	All movement going beyond this position is stopped. #STATUS.19 bit is set and only CCW movement is enabled. **This parameter is stored in non-volatile memory.**
Notes:	`#POSITIVE_END` value is not modified when `#POSITION` is modified
Example:	`#POSITIVE_END:=100000` Module can only go 100 rev. CW away from home position.

5.26. #RATED_CURRENT

Mnemonic	#RCU
Parameter:	Nominal current of the motor, expressed in mA RMS per phase. [0 ; 2500] for BMAC [0 ; 7000] for BMAC-H
Description:	With this variable, the user defines the maximum current that will be sent in the motor. The actual current is defined according to #CURRENT_RATIO variable: Current (mA) = #RATED_CURRENT * #CURRENT_RATIO /1000 **This parameter is stored in non-volatile memory.**
Notes:	Refer to the motor specification sheet or identification mark to setup #RATED_CURRENT
Example:	`00#RATED_CURRENT:=1200` Setup BMAC to drive a 1.2A_{RMS} motor

5.27. #SPEED, #PROFILE_SPEED (read only)

Mnemonic	#SPE, #PSP
Parameter:	Axis velocity in 100th of RPM. Signed
Description:	Variable `#SPEED` is the current motor speed (measured). Variable `#PROFILE_SPEED` is the computed speed of the module.
Notes:	The value of `#SPEED` is determined by sampling module position. Instantaneous value can then be slightly different from real speed.
Example:	`04READ #SPEED` → `04#SPE=-20000` Reading speed of module 4. Module 4 rotates at 200RPM in CCW direction.

5.28. #STATUS (read only)

Mnemonic #STA

Parameter: Aucun

Description: BMAC status word:

MSB	bit 32	Movement stopped abnormally
	bit 31	Error (check #ERROR for details)
	bit 30	Reference mode
	bit 29	Busy
	bit 28	Closed Loop (1) or Open-Loop (0)
	bit 27	Position control
	bit 26	Movement
	bit 25	Power ON (1) or OFF (0)
	bit 24	
	bit 23	Synchro mode
	bit 22	
	bit 21	Brake active (1) or inactive (0, see BRAKE command)
	bit 20	Soft negative end-stop
	bit 19	Soft positive end-stop
	bit 18	Hard negative end-stop
	bit 17	Hard positive end-stop
	bit 16	Sequencer Edition mode
	bit 15	Sequencer run
	bit 14	Output polarity standard (0) or inverted (1)
	bit 13	Input polarity standard (0) or inverted (1)
	bit 12	Velocity profile « S » (1) or trapezoidal (0)
	bit 11	
	bit 10	
	bit 9	
	bit 8	Inverted polarity of hard end-stops inputs
	bit 7	Enable soft end-stops
	bit 6	Enable hardware negative end-stops
	bit 5	Enable hardware positive end-stops
	bit 4	Optimized current mode
	bit 3	
	bit 2	
LSB	bit 1	

Example: `03READ h#STATUS` → `03#STA=h13000800`
Reading status in hexadecimal form.

5.29. #SUPPLY_VOLTAGE (read only)

Mnemonic	#SVO
Parameter:	Measure of supply voltage in mV
Example:	`02READ #SUPPLY_VOLTAGE` → `02#SVO=32000` Module 2 supply voltage is 32V.

5.30. #TIMER_1 to #TIMER_3

Mnemonic	#T1 to #T3
Parameter:	Tempo en milliseconds [0 ; + 2 147 483 647]
Description:	Variables `#TIMER_1` to `#TIMER_3` are to be used within a sequence to insert a delay. Starting from a value defined by the user, those three variables are decremented every millisecond down to zero. They can be read (`READ #TIMER_1`) or tested (`IF #TIMER_1=0 JUMP…`) to know if the delay is over.
Example:	`#TIMER_3:=2000` Variable `#TIMER_3` is loaded with value 2000. It will be decremented every millisecond for the next two seconds.

5.31. #V1 to #V32

Mnemonic:	#V1 to #V32
Parameter:	4 bytes signed value [-2 147 483 648 ; + 2 147 483 647]
Description:	Variables `#V1` to `#V32` can be used to store values, parameter or math results. They can be used within a sequence. They are initialized to zero at reset or power-up.
Example:	`00#V13:=1234` Stores value 1234 in variable `#V13` of module 0.

6. Sequencer

6.1. Features

BMAC can store and then execute command lines.
Each line is represented by its "line number" (1 to 500).
When sequencer is at line zero (#LINE=0), it is not running.
When sequencer is running (command "START_SEQ n" where n is the first line to be executed), the module executes one line every millisecond and goes on to next line (automatically incrementing #LINE).

6.2. Lines storage

To store sequence lines in non-volatile memory:

1. Enable "Edit mode" using command OPEN_SEQ.
2. Type commands just as if they were to be executed in "Live mode". The line number is automatically incremented or can be forced by preceding the command with " :n " where n is the line number.
   (Example: "00:120 MOVE_TO 15000")
3. Disable "Edit mode" with command CLOSE_SEQ

Example: Storage of a sequence that makes one rev every two seconds:

```
OPEN_SEQ            ;enable Edit mode
MOVE_ON 10000       ;1 rev CW
WAIT 2000           ;delay
JUMP 1              ;go back to line 1
CLOSE_SEQ           ;disable Edit mode
```

The following command lines will be stored:
Line 1 : MOVE_ON 10000
Line 2 : WAIT 2000
Line 3 : JUMP 1

6.3. Execution of the sequencer

Use START_SEQ command to launch sequencer execution. First line number can be specified as an argument if needed.

To stop sequencer execution, use command STOP (or STOP SEQ to keep motor movement).
Line 0 corresponds to sequencer stop. When the module is at line 0, it stays there until it receives a START_SEQ command.

The sequencer can be configured to start execution at module reset if #ON_RESET variable contains a value.

Example: To start sequencer from line 12 on reset, type " #ON_RESET:=12 ".
To disable this feature, type #ON_RESET:=0

6.1. Sample sequences

6.1.1. Example 1

Digital output 4 is ON if supply voltage is in the range 15V - 20V.

```
OPEN_SEQ                                ;enter Edit mode
IF #SUPPLY_VOLTAGE < 15000 JUMP 5  ;output OFF if V < 15Volts
IF #SUPPLY_VOLTAGE > 20000 JUMP 5  ;output OFF if V > 20Volts
#OUTPUT.4:=1                            ;output ON otherwise
JUMP 1                                  ;loop on line 1
#OUTPUT.4:=0                            ;output OFF
JUMP 1                                  ;loop on line 1
CLOSE_SEQ                               ;quit Edit mode
START_SEQ 1                             ;Sequencer start
```

6.1.2. Example 2

Speed control loop: The BMAC velocity is determined by analog input with gain and offset.
(this example use mnemonics instead of full-size commands)

```
OSE                                     ;enter Edit mode (OPEN_SEQ)
:50 #V1:=#IAN - 15000                   ;line 50: #INPUT_ANALOG - 15000 in V1
#V1:=#V1 * 5                            ;gain of 5
MSP #V1                                 ;movement (MOVE_SPEED)
JRE -3                                  ;loop on 50 (JUMP_REL)
CSE                                     ;quit Edit mode (CLOSE_SEQ)
SSE 50                                  ;Sequencer start (START_SEQ)
```

6.1.3. Example 3

Digital input 2 triggers 1 rev clockwise and digital input 3 triggers 1 rev counter-clockwise.

```
OPEN_SEQ                          ;enter Edit mode
IF #INPUT.2 = 1 JUMP_REL +3       ;test input 2
IF #INPUT.3 = 1 JUMP_REL +4       ;test input 3
JUMP 1                            ;loop on line 1
MOVE_ON 10000                     ;1 rev CW
JUMP_REL 2                        ;jump to "wait for the end of the movement"
MOVE_ON -10000                    ;1 rev CCW
WAIT 0                            ;wait for the end of the movement
JUMP 1                            ;loop on line 1
CLOSE_SEQ                         ;quit Edit mode
START_SEQ 1                       ;Sequencer start
```

6.1.4. Example 4

Call subroutine to reset position if input 3 is OFF or if supply voltage is above 30V.

```
OPEN_SEQ                                ;enter Edit mode at line 1
:150 IF #INPUT.3 = 0 CALL 160           ;Line 150: test input 3
IF #SUPPLY_VOLTAGE > 30000 CALL 160     ;test supply voltage
JUMP 0                                  ;stop sequencer

:160 #POSITION:=0                       ;Line 160: reset position
RETURN                                  ;back to CALL
CLOSE_SEQ                               ;quit Edit mode
START_SEQ 150                           ;Sequencer start
```

7. ANNEXES

7.1. Summary of commands

Command	Mnemo.	Parameter	Direct	Sequence	Factory value (MRE ALL)
CALL	CAL	seq line		◊	
CLOSE_LOOP	CLO	ON / OFF	◊	◊	OFF
CLOSE_SEQ	CSE	-	◊		
HALT	HAL	- / SEQ / MOUV	◊	◊	
HARD_ENDS	HEN	OFF / ALL / POS / NEG	◊	◊	OFF
IF	IF	test		◊	
INVERSE_POLARITY	IPO	OFF / ALL / IN / OUT	◊	◊	OFF
JUMP	JUM	seq line		◊	
JUMP_REL	JRE	seq line		◊	
MODULE_RESET	MRE	- / ALL	◊		
MOVE_INTERPOL	MIN	relative position	◊		
MOVE_ON	MON	relative position	◊	◊	
MOVE_SPEED	MSP	speed	◊	◊	
MOVE_TO	MTO	absolute position	◊	◊	
OPEN_SEQ	OSE	-	◊		
OPTIMIZED_CURRENT	OCU	ON / OFF	◊	◊	OFF
POWER	POW	ON / OFF / SC	◊	◊	OFF
READ	REA	variable	◊		
READ_SEQ	RSE	seq line	◊		
REFERENCE	REF	INDEX / ON / OFF	◊	◊	OFF
RETURN	RET	-		◊	
REQUEST_VERSION	RV	-	◊		
SET_ADDRESS	SAD	address	◊		0
SET_BAUDRATE	SBA	baudrate	◊		38400
S_CURVE	SCU	ON / OFF	◊	◊	OFF
SOFT_ENDS	SEN	ALL / OFF	◊	◊	OFF
START_SEQ	SSE	seq line	◊		
STEP	STE	seq line	◊		
STOP	STO	- / SEQ / MOUV	◊	◊	
SYNCHRO	SYN	ON / OFF /TOP / INTERPOL	◊		OFF
WAIT	WAI	time		◊	

7.2. Summary of variables

Variable	Mnemo.	Memorized	Read only	Factory value (or MRE ALL)
#ACCEL_TIME	#ATI	◊		1 000
#CAPTURE	#CAP		◊	
#CPU_TEMPERATURE	#CTE		◊	
#CURRENT_RATIO	#CRA	◊		500
#DECEL_TIME	#DTI	◊		1 000
#DRIVER_TEMPERATURE	#DTE		◊	
#ELECTRICAL_POSITION	#EPO	◊	◊	0
#ENCODER	#ENC			0
#ERROR	#ERR			
#HIGH_SPEED	#HSP	◊		60 000
#INPUT	#INP		◊	
#INPUT_ANALOG	#IAN		◊	
#INTERPOL_COUNT	#ICO		◊	
#INTERPOL_FIFOSIZE	#IFI			64
#INTERPOL_MODE	#IMO			0
#INTERPOL_TIME	#ITI			100
#LINE_DELAY	#LDE	◊		3 000
#LINE	#LIN			0
#LOW_SPEED	#LSP	◊		6 000
#M1 to #M8	#M1 to #M8	◊		0
#NEGATIVE_END	#NEN	◊		-100 000
#ON_RESET	#ORE	◊		0
#OUTPUT	#OUT			0
#OUTPUT_CONFIG	#OCO	◊		3
#POSITION	#POS	◊		0
#POSITIVE_END	#PEN	◊		+100 000
#PROFILE_SPEED	#PSP		◊	
#RATED_CURRENT	#RCU	◊		2500
#SPEED	#SPE		◊	
#STATUS	#STA		◊	
#SUPPLY_VOLTAGE	#SVO		◊	
#TIMER_1 to #TIMER_3	#T1 to #T3			0
#V1 to #V32	#V1 to #V32			0

7.3. Related documents

Those related documents can be downloaded on www.midi-ingenierie.com

- Application Note – Installation du driver USB FTDI
- Application Note – Liaison calculateur : protocoles et syntaxe
- User Manual DMAC & BMAC CANopen