Motion Drive

Digital drive for Brushless motor
MD serial

User manual

Read manual before installing and respect all indications with this icon: 

SERAD SA
271, route des crêtes
44440 TEILLE – France
☎ +33 (0)2 40 97 24 54
🖷 +33 (0)2 40 97 27 04
💻 http://www.serad.fr
✉ info@serad.fr

Table of Contents

1- Introduction

1-1- Warning

Only suitable qualified personnel should undertake the mounting, installation, operation and maintenance of the equipment.

It is important that all safety instructions are strictly followed. Personal injury can result from a poor understanding of the safety requirements.

A bad shield connection can damage drive electronic composants.

The following safety regulations should be followed:

- VDE 0100 — Specification for the installation of power systems up to 1000 V
- VDE 0113 — Electrical equipment of machines
- VDE 0160 — Equipment for power systems containing electronic components.

- ***Never open the equipment.***
- ***Dangerous high voltages exist within the equipment and on the connectors. Because of this, before removing any of the connectors, it is necessary to remove the power and wait at least 5 minutes to allow the capacitors to discharge.***
- ***Never connect or disconnect the drive with power applied.***
- ***Some of the drive's surfaces can be very hot.***

Some of the drive's components are susceptible to damage from electrostatic discharges. Always handle the equipment using appropriate anti-static precautions.

We reserve the right to make changes to all or part of the specification without prior notice.

1-2- MD series drive description

1-2-1- General :

The MD Series intelligent brushless drives are specially adapted for high dynamic performance.

They contain an integrated power supply, mains filter and braking resistor.

They can be used to control motor torque, speed or position depending on their operating mode.

Various field bus configurations are available such as MODBUS, CANopen and PROFIBUS DP that allow for the use of the drives in networked systems.

Thanks to their easy-to-program Basic language, multi-tasking kernel, MOTION control features and integrated PLC functions, they are well suited to a wide range of applications.

1-2-2- Technical data :

Supply :

MD 230 M : 230V AC ±10% single phase
MD 230 T : 230V AC ±10% three phase
MD 400 T : 400V AC ±10% three phase

Auxiliary supply :

24 V DC ±10%, 0.5A typical (0,7A max with encoder O/P)

Supply filter :

Integral

Switching frequency :

6.25 kHz sine-wave PWM

DC bus voltage :

310 V for MD 230 series, 560V for MD 400 series

Leakage current

2,2 mA for MD 230 series, 1 mA for MD 400 series

Braking resistance :

Integral : MD 230 : 110 ohms 30W

MD 400 : 180 ohms 30W

Facility to add an external resistor:

Type	Min. value	Max.cont. power	Max imp. power
MD230/1 ou /2	60 Ω	1000W	2300W
MD230/5 ou /7	30 Ω	1800W	4600W
MD 400	80 Ω	2800W	7000W

Protection :

Short circuit between phases, phase to earth, over current, I2t

Over voltage, under voltage

Motor feedback fault

Motor feedback :

Resolver (16 bit resolution)

Precision absolute resolver ± 0,7°

Incremental encoder (option)

Master encoder input :

Incremental : A, /A, B, /B, Z, /Z

Maximum frequency : 800 kHz

Encoder emulation :

Incremental : A, /A, B, /B, Z, /Z 1024 points per rev

Diagnostic display :

7 segments LED

Communication :

RS 232 MODBUS RTU

RS 422 (point to point), RS 485 MODBUS RTU (option)

CANopen (option)

Digital inputs :

4 inputs standard

12 additional inputs with expansion module

Type: PNP, 24V DC, 12mA per input

Logic 0: Between 0 and 5 V

Logic 1: Between 10 and 30 V

Digital outputs :

2 outputs as standard

S1 : Relay, 48V dc / 48V ac, 3A max

S2 : NPN (open collector) 24V dc, 100mA

8 additional outputs with expansion module

Type : PNP 24V dc, 100mA max per output

Protected against short circuit and over temperature.

Analogue inputs :

2 inputs :

Input voltage : ±10 V

Maximum voltage : ±12 V

Input impedance : 20 kΩ

Resolution : 10 bits

Analogue output :

1 output :

Output voltage : ±10 V

Maximum current : 5 mA

Resolution : 8 bits

Architecture :

Processor : 40 MHz DSP

Memory : FLASH for programs and parameters

RAM for data

Real-time, multi-tasking kernel

Control loops :

Current loop : 160 µs

Speed loop : 320 µs

Position loop : 640µs

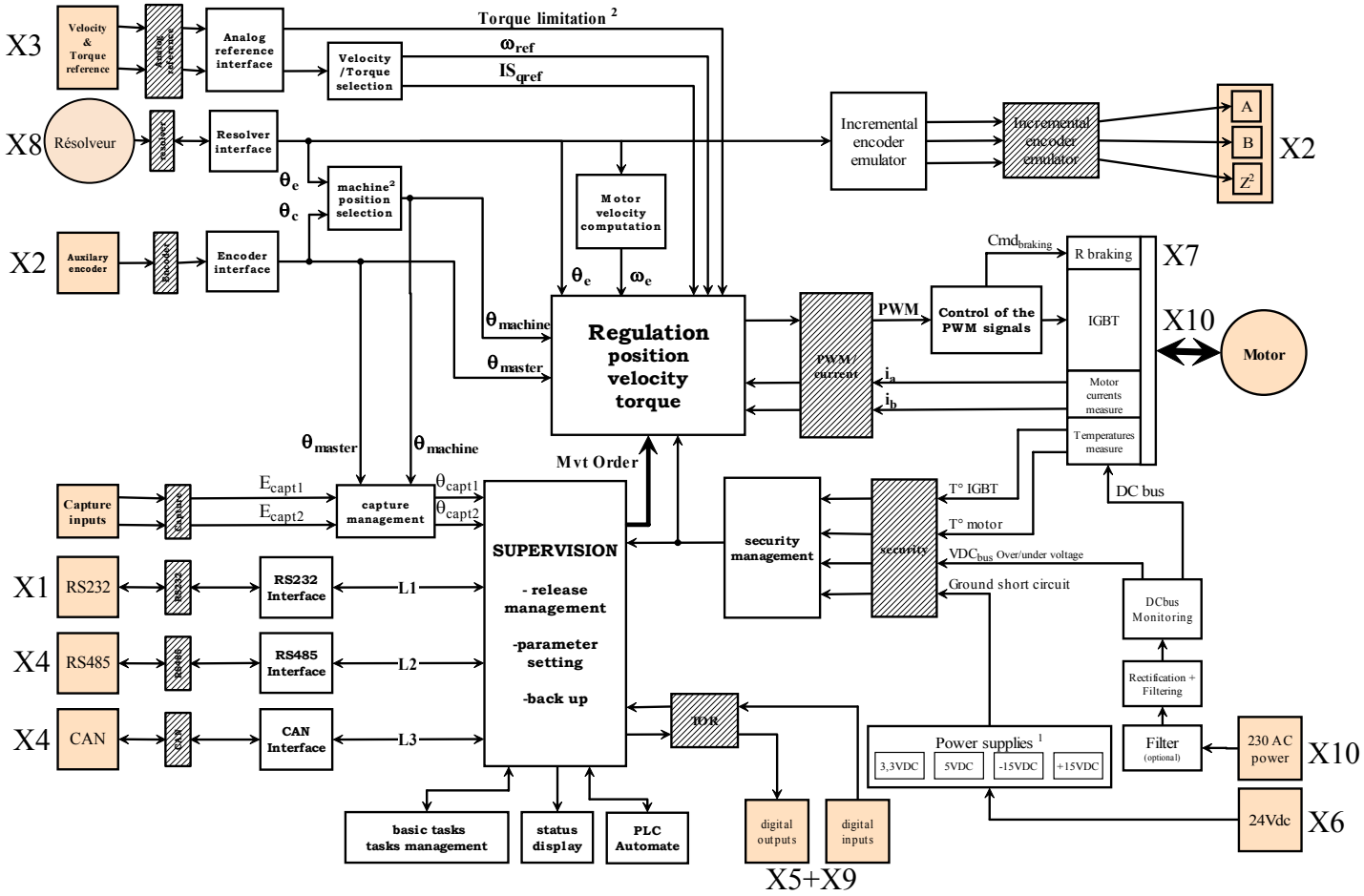
Operating modes :

Torque mode

	Speed mode
	Position mode
	Motion control
Operating temperature :	0 to 40°C
Storage temperature :	-10 to 70°C
Degree of protection :	IP 20

Drive	Rated current	Peak current (2s)	Rated power	Dimensions w x h x d
MD 230 / 1	1,25 Aeff	2,5 Aeff	0,35 kVA	67 x 215 x 203
MD 230 / 2	2,5 Aeff	5 Aeff	0,7 kVA	67 x 215 x 203
MD 230 / 5	5 Aeff	10 Aeff	1,5 kVA	67 x 215 x 203
MD 230 / 7	7,5 Aeff	15 Aeff	2,3 kVA	67 x 215 x 203
MD 400 / 1	1,25 Aeff	2,5 Aeff	0,7 kVA	67 x 215 x 203
MD 400 / 2	2,5 Aeff	5 Aeff	1,4 kVA	67 x 215 x 203
MD 400 / 5	5 Aeff	10 Aeff	3 kVA	67 x 215 x 203

Schéma synoptique :



1-3-1- General :

The DPL software, with its graphical user interface, allows the user to easily configure the drive from a PC.

Operating within a Windows environment, the user-friendly software provides for multiple windows and full help facilities.

The auto tuning, trajectory generator and oscilloscope functions ensure speedy and optimum system set-up and rapid commissioning.

1-3-2- Technical data :

- Configuration of all parameters, grouped by function : motor, regulation, encoder, analogue I/O, digital I/O, communication, supervision
- Downloading of set-up and parameters : speeds, currents, torques, positions
- Saving of all parameters on a PC
- Automatic resolver offset adjustment
- Trajectory generator : position, acceleration, deceleration, speed
- Digital multi-channel oscilloscope
- Set-up screen : axis, inputs, outputs
- Automatic recognition of connected drive
- Ability to work and edit parameters when not connected to a drive
- On-line help for each window

1-3-3- DPL programming language:

The MD series drives incorporate a real-time, multi-tasking kernel and have more than 1000 user variables.

The pseudo-basic language, DPL, allows users to develop, test and save their own application programs.

These applications can use any combination of operating modes e.g. torque, speed and position. All of the I/O can be controlled from within the program as well as parameters and variables.

2- Installation

2-1- General

 **It is very important to adhere to the following:**

↳A bad shield connection can damage drive electronic composants

↳ The drive must be installed vertically in free air to ensure cooling by natural convection.

↳ It must be protected from excess humidity, liquids, and dirt.

↳ The motor, resolver and encoder cables must be screened, the screen being earthed at both ends of the cable.

↳ The analogue I/O must use screened cable, the screen being earthed at one end only.

↳ The cable for the RS 232 serial link between the drive and the PC must be screened, the screen being earthed at both ends of the cable. It should be disconnected from the drive when no longer in use. All of these cables, as well as the I/O cables, should be run separately from the power cables.

↳ Diodes must be fitted across the loads on all static digital outputs (Q2 to Q10). These diodes must be positioned as close to the load as possible. The supply and signal cables must be free from over-voltage transients.

↳ Safety standards specify a manual reset after a stop caused either by a supply interruption, or by an emergency stop or by a drive fault.

↳ For all serious faults, it is obligatory to remove the high voltage supply to the drive.

↳ The Drive Ready output should be connected in series in the emergency stop loop.

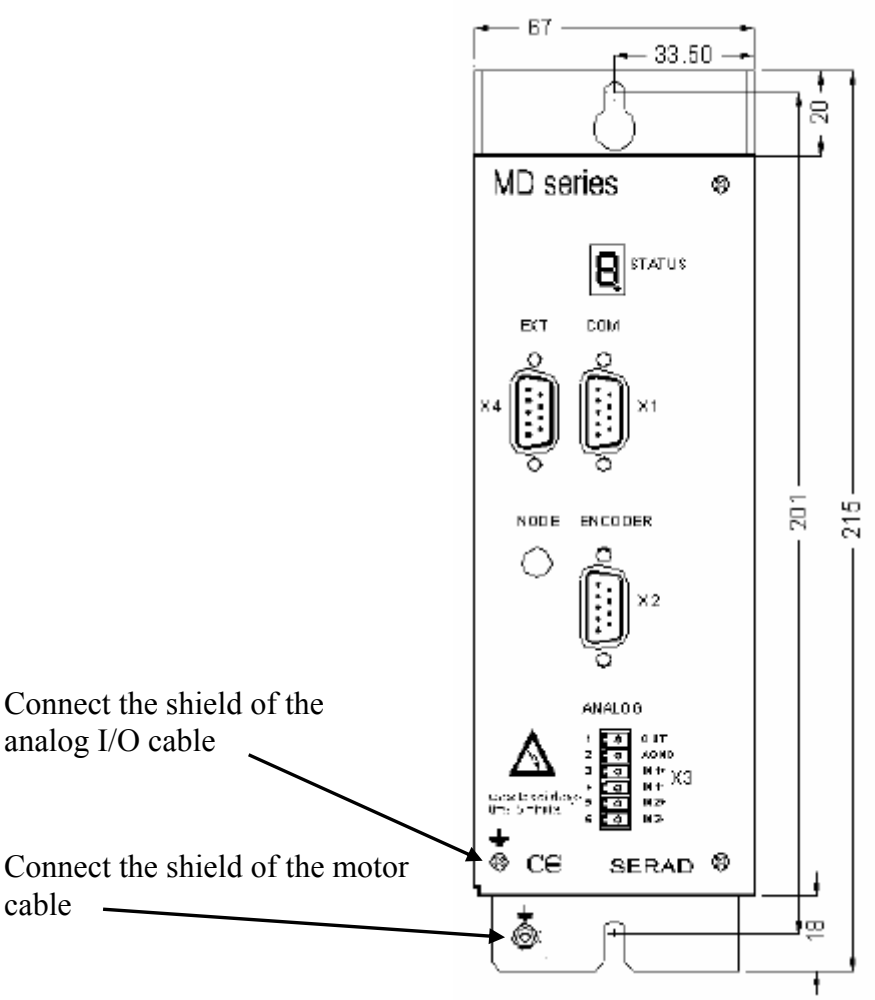
↳ In the case of axis over-travel, the over-travel limit switches must be connected to the limit inputs or in series with the emergency stop loop. It is also recommended to use the software limits.

↳ If the drive is configured in speed loop, the drive enable input should be controlled by the supervisory controller (CNC, PLC etc).

↳ If the drive is configured in position loop, the parameter "Maximum following error" should be set appropriately.

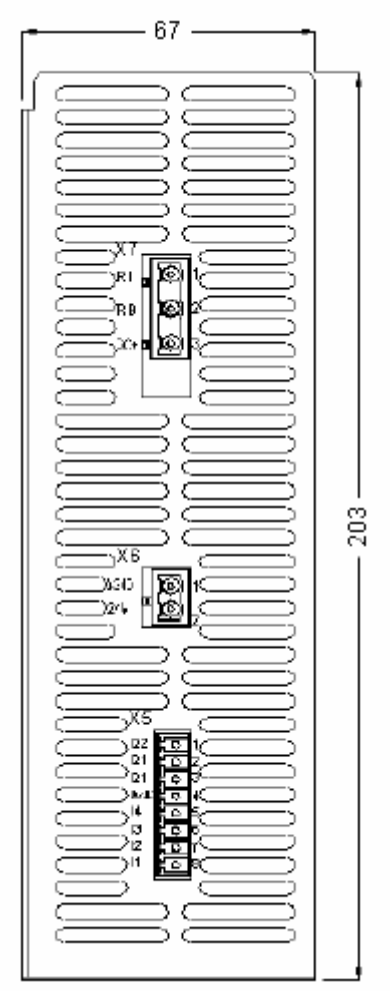
↳ If the drive contains an application program developed using DPL, connect a signal ‘Cabinet supplies OK’ to one of the digital inputs and monitor it in a non-blocking safety task. On detection of an excess following error the drive will be put in open loop mode and the drive ready relay will be opened. If another action is required you should use the SECURITY instruction.

2-2- Front view



	STATUS	7-segment diagnostic display
X1	COM	RS-232 serial port for communication with a PC
X2	ENCODER	Master encoder input / Simulated encoder output
X3	ANALOG	Analogue I/O
X4	EXT	Extension: Optional communications ports

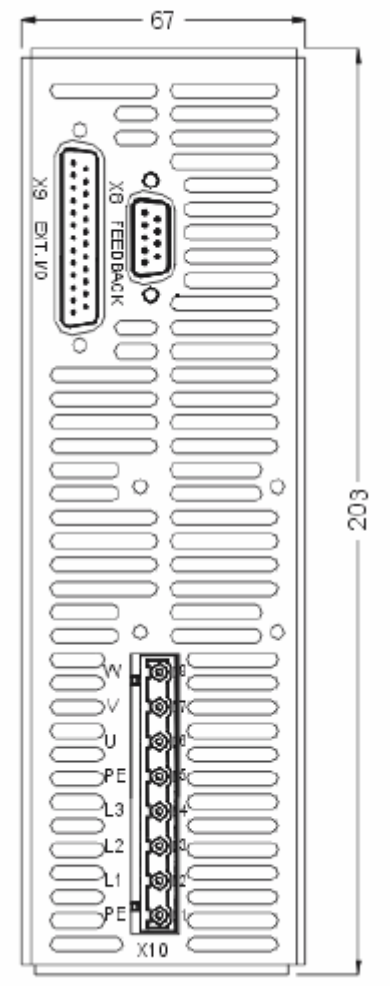
2-3- Top view



X5	I/O	Digital I/O
X6	24Vdc	Auxiliary 24V DC supply
X7	RB	External braking resistor

The voltage on connector X7 can reach 400V for an MD 230 and 800V for an MD 400!

2-4- Bottom view

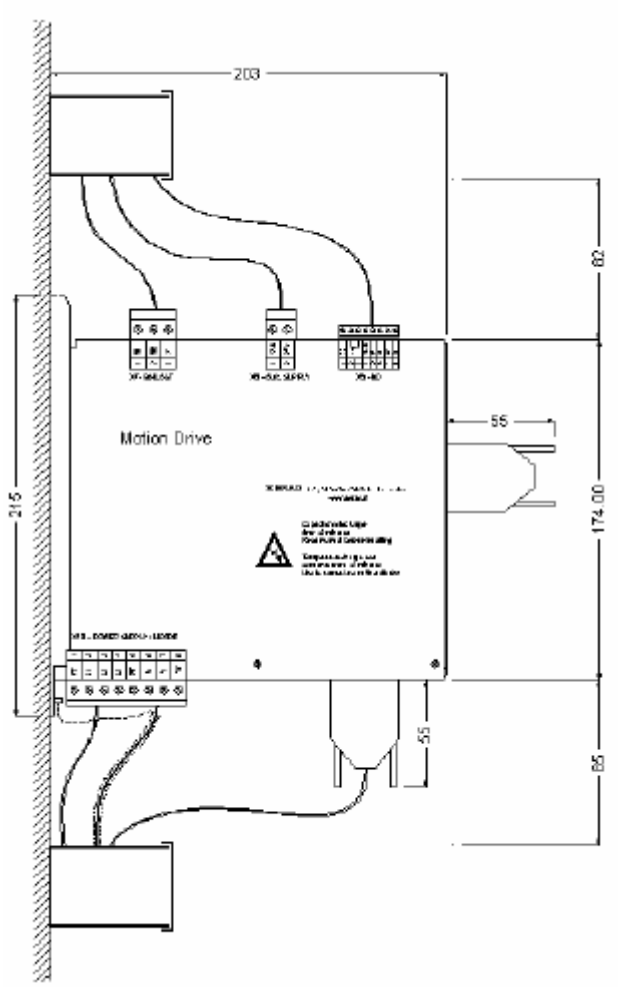


X8	FEEDBACK	Motor position feedback (resolver / encoder)
X9	EXT I/O	Option : I/O expansion board
X10	POWER	Single / Three-phase supply Motor armatures

Attention. Care must be taken when making connection to connector X10. An incorrect connection can seriously damage the drive. Dangerous voltages are present on X10.

2-5- Mounting

Several drives can be mounted side-by-side provided that enough space (at least 20 mm) is left to ensure good natural convection and also to allow for the various connectors and cables to be fitted

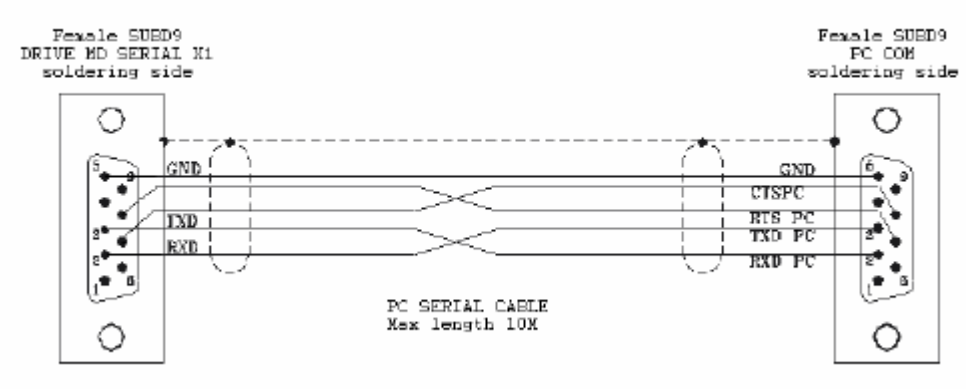


2-6- Connector pin assignments

X1: RS 232 serial port for downloading programs and parameters.

Connector: SUBD 9 way male

No.	Name	Type	Description
1			
2	RXD	In	Receive data
3	TXD	Out	Transmit data
4			
5	GND		0V
6			
7			
8	CTS	In	Clear to send
9			
	SHIELD		Connect the shield to the shell of the connector



X2: Master encoder input / simulated encoder output

Connector: SUBD 9 way female

No.	Name	Type	Description
1	A	I/O	Channel A
2	/A	I/O	Channel A inverted
3	B	I/O	Channel B
4	/B	I/O	Channel B inverted
5	Z	I/O	Zero marker
6	/Z	I/O	Zero marker inverted
7	+5Vdc	Out	Supply for external encoder, 100 mA max.
8	GND		0V
9			
	**SHIELD**		**Connect the shield to the shell of the connector**

X3: Analogue I/O

Connector : removable 6 way, 3.81mm pitch

No.	Name	Type	Description
1	OUT	Out	Analogue output (function monitor)
2	AGND		0V analogue
3	IN1+	In	Analogue input 1 : assigned to speed or torque command, according to mode
4	IN1-	In	Analogue input 1
5	IN2+	In	Analogue input 2 : assigned to torque limit
6	IN2-	In	Analogue input 2
	**SHIELD**		**Connect the shield to the screw of the drive case**

X4: Extension: Optional communications port

No.	RS 232 SUBD 9 way male	RS 422 SUBD 9 way female	RS 485 SUBD 9 way female	CANopen SUBD 9 way male
1				
2	RXD			
3	TXD	RX-		
4		RX+		
5	GND	GND	GND	CAN_GND
6				
7		TX-	TRX-	CAN_L
8		TX+	TRX+	CAN_H
9				
	**SHIELD - Connect the shield to the shell of the connector**			

- Node Address : For RS422, RS485 and CANopen, the NodeID corresponds to the rotary switch position + 1

  e.g. : rotary switch in position 3 ⇒ NodeID 4

- Extended Node Address : For RS422, RS485 and CANopen, link 1 to pin 6. The NodeID then corresponds to the rotary switch position + 17

  e.g. : rotary switch in position 3 ⇒ NodeID 20

- Check bus termination resistance (120Ω) :

  For RS422, link pin 2 to pin 3, and pin 8 to pin 9.

  For RS485 and CANopen, link pin 8 to pin 9.

X5: Digital I/O

Connector : removable 8 way, 3.81mm pitch

No.	Name	Type	Description
1	Q2	Out	Output 2, programmable : type NPN, 24 Vdc, 100mA
2	Q1	Out	Output 1, programmable : standard function DRIVE READY
3	Q1		Relay contact, N/O between terminals 2 and 3
4	DGND		0V digital I/O
5	I4	In	Input 4, programmable
6	I3	In	Input 3, programmable
7	I2	In	Input 2, programmable
8	I1	In	Input 1, programmable: standard function ENABLE

 The output Q2 is NPN open collector: the load must be connected between Q2 and +24V DC.

X6: 24V dc supply

Connector : removable 2 way, 5.08mm pitch

No.	Name	Type	Description
1	XGND		0V
2	24V dc	In	Control card supply, backup motor position

X7: External braking resistance

Connector : removable 3 way, 7.62mm pitch

No.	Name	Type	Description
1	RI		Internal braking resistor *
2	RB		Braking resistor *
3	DC Bus +	Out	DC bus (310 V for MD 230, 560 V for MD 400)

*Selection of the braking resistor :

- Internal resistor : Fit a link between terminals 1 and 2
- External resistor : Remove the link between terminals 1 and 2

Connect the external resistor between terminals 2 and 3

 The voltage on connector X7 can reach 400V for an MD 230 and 800V for an MD 400!

X8: Motor position feedback (resolver)

Connector : SUBD 9 way female

No.	Name	Type	Description
1	S2	In	Sine Hi
2	S1	In	Cosine Hi
3	AGND		0V analogue
4	R1	Out	Reference Hi
5	°CM+	In	Motor temperature sensor Hi
6	S4	In	Sine Lo
7	S3	In	Cosine Lo
8	°CM-	In	Motor temperature sensor Lo
9	R2	Out	Reference Lo
	**SHIELD**		**Connect the shield to the shell of the connector**

X9: Option : Expansion module, 12 inputs / 8 outputs

Connector : SUBD 25 way female

No.	Name	Type	Description
1	I5	In	Input 5, programmable
2	I6	In	Input 6, programmable
3	I7	In	Input 7, programmable
4	I8	In	Input 8, programmable
5	I9	In	Input 9, programmable
6	I10	In	Input 10, programmable
7	IOGND*		0V digital I/O
8	Q3	Out	Output 3, programmable
9	Q4	Out	Output 4, programmable
10	Q5	Out	Output 5, programmable
11	Q6	Out	Output 6, programmable
12	IO 24V dc**	In	External supply, 24 V dc
13	IO 24V dc**	In	External supply, 24 V dc
14	I11	In	Input 11, programmable
15	I12	In	Input 12, programmable
16	I13	In	Input 13, programmable
17	I14	In	Input 14, programmable
18	I15	In	Input 15, programmable
19	I16	In	Input 16, programmable
20	Q7	Out	Output 7, programmable
21	Q8	Out	Output 8, programmable
22	Q9	Out	Output 9, programmable
23	Q10	Out	Output 10, programmable
24	IOGND*		0V digital I/O
25	IOGND*		0V digital I/O
	**SHIELD**		**Connect the shield to the shell of the connector**

*Pins 7, 24, 25 : internal connection **Pins 12, 13 : internal connection

X10: High voltage supply, motor armature

Connector : removable 8 way, 7.62mm pitch

No.	Name	Type	Description
1	PE		Supply earth
2	L1*	In	Supply L1 (230V for MD 230, 400V for MD 400)
3	L2*	In	Supply L2 (230V for MD 230, 400V for MD 400)
4	L3	In	Supply L3 (230V for MD 230, 400V for MD 400)
5	PE		Motor earth
6	U	Out	Motor phase U
7	V	Out	Motor phase V
9	W	Out	Motor phase W

For a 230V ac single-phase supply, connect Live to L1 and Neutral to L2


Attention. Care must be taken when making connection to connector X10. An incorrect connection can seriously damage the drive. Dangerous voltages are present on X10.

The armoured motor cable must arrive directly on the terminals of the drive.

Connect the shield (on drive side) to the srew provided (see 2-2 Front view).

2-7- Cables

- **RS 232 serial communication cable, X1 :**

Screened cable, 4 core

Connect the shield on each extremity, to the shell of the connector.

- **Encoder cable, X2 :**

Screened cable with 4 twisted pairs, 0.25 mm²

Connect the shield on each extremity, to the shell of the connector.

- **Analogue cable, X3 :**

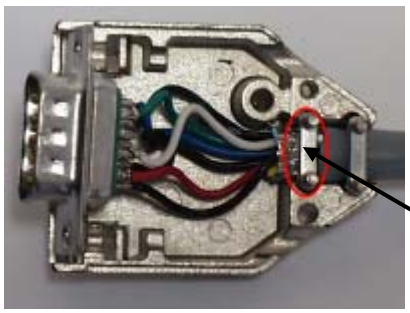
Screened cable, 2 core, 0.25 mm² per analogue input.

Connect the shield: on drive side to the srew provided (see 2-2 Front view) and on the other side to the sheld equipement (ex. Motion controller …)

- **Motor feedback cable (resolver), X8 :**

Screened cable with 4 twisted pairs, 0.25 mm²

Ground the shield of the feedback SUBD as shown below:



- **Motor power cable, X10 :**

Cable with general shielding, 4 wires (more two if brake).

Section 1,5 mm² for variator until 8 A. Beyond that, envisage of the 2,5 mm².

Connect the shield (on drive side) to the srew provided (see 2-2 Front view).

2-8- Connection diagrams / Protections

The cables must be tested before being connected as any wiring fault can give rise to serious problems

Remove all voltages before inserting the connectors.

Ensure that the earth connection to the drive is correctly made (pin 1 of the connector X10).

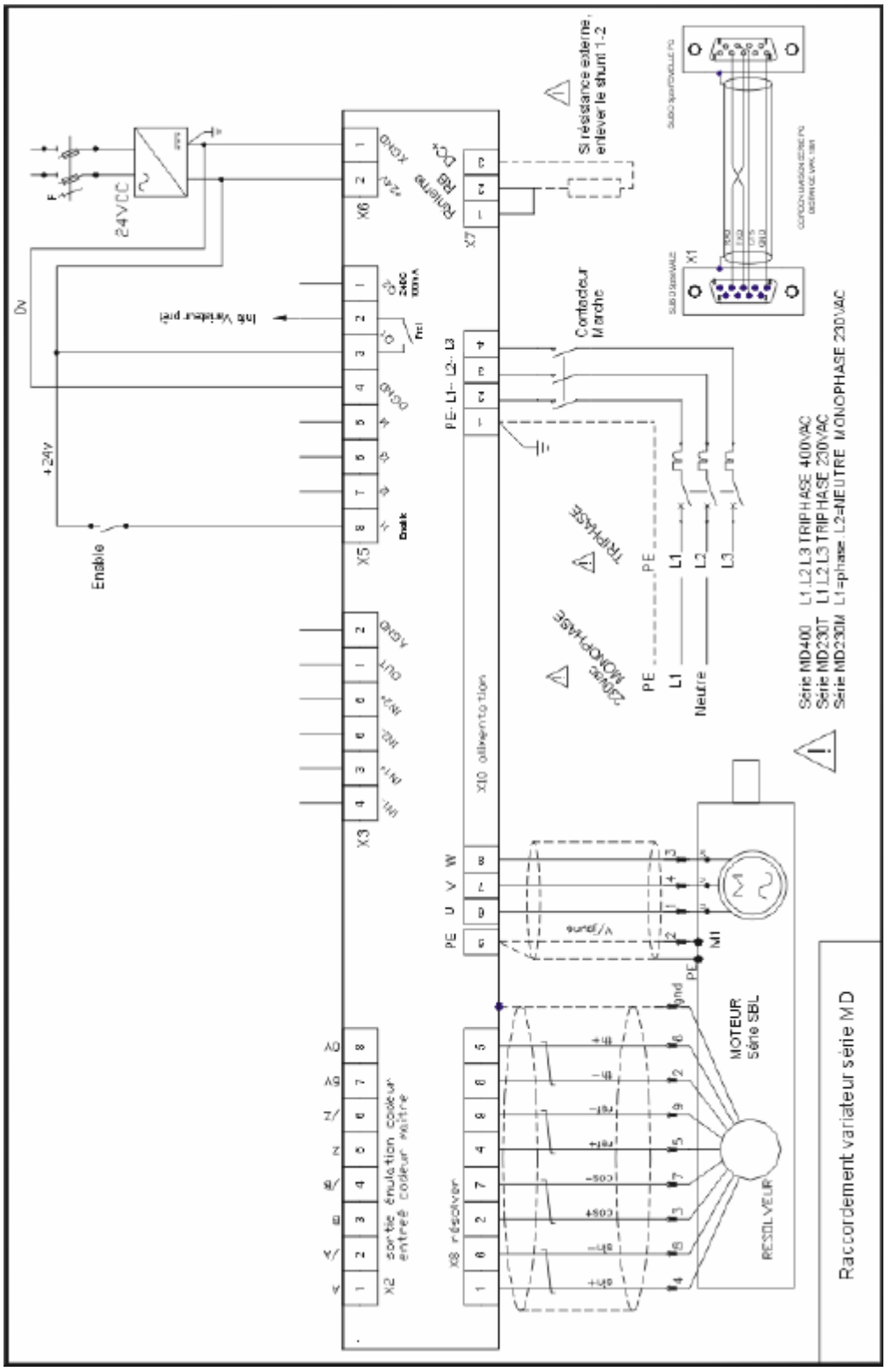
Connect the motor earth to the drive (pin 1 of the connector X10) before applying any voltages.

For the shielded cables, to connect the braid to the frame at each extremity via the caps of the connectors (for the SUBD) or the screws provided for this purpose (X3 connectors, X10) in order to ensure an optimal equipotentiality.

Drive	Input voltage	Maximal input current	Safety device : cutout C curve	Wire
MD230/1	230V single phase	3,5A	10A maxi	1,5²
MD230/2	230V single phase	7A	10A maxi	1,5²
MD230/5	230V single phase	14A	10A maxi	1,5²
MD230/7	230V single phase	21A	16A maxi	2,5²
MD400/1	400V three phase	2,2A	10A maxi	1,5²
MD400/2	400V three phase	4,2A	10A maxi	1,5²
MD400/5	400V three phase	8,2A	10A maxi	1,5²

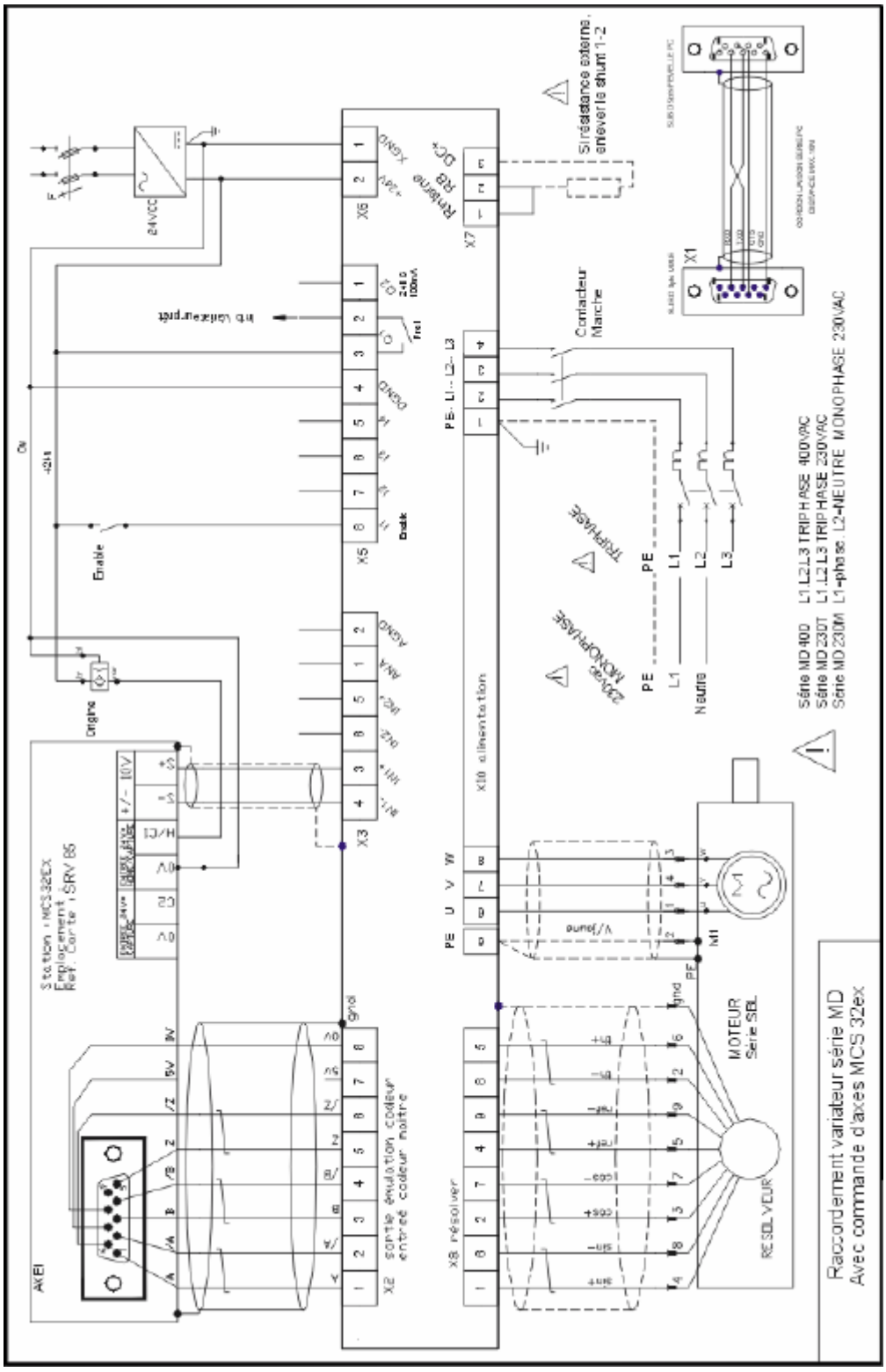
Caution: the ringing current can reach 25A.

2-9- Stand-alone drive



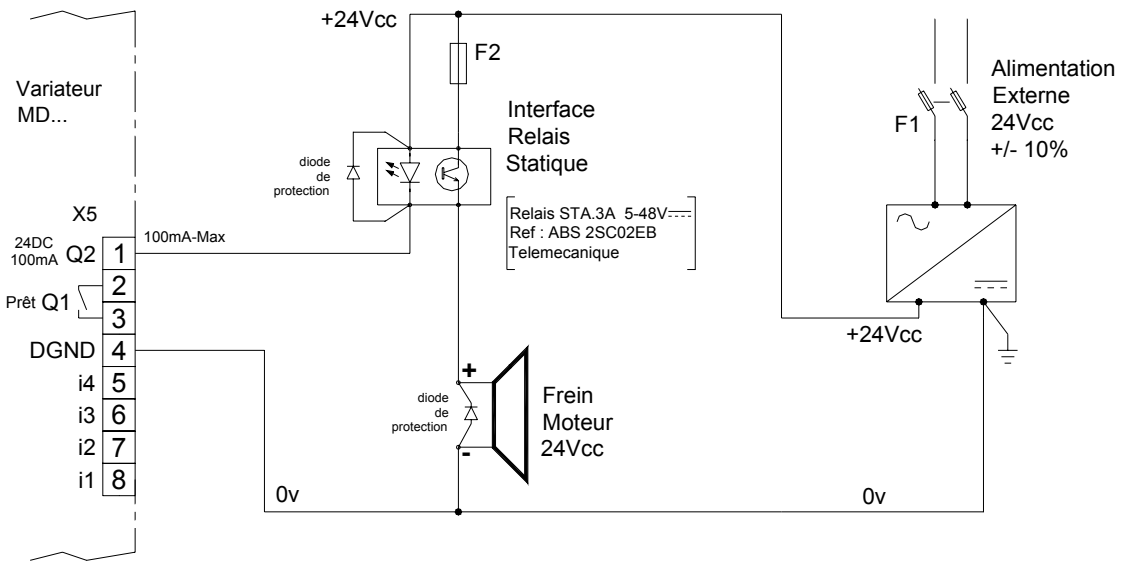
The output Q2 is NPN open collector, 100mA max. The load must be connected between Q2 and +24Vdc.

2-10- Drive controlled by a motion controller



The output Q2 is NPN open collector, 100mA max. The load must be connected between Q2 and +24Vdc.

2-11- Connecting a motor brake



The output Q2 is NPN open collector, 100mA max. The load must be connected between Q2 and +24Vdc.

Using the DPL parameter set-up window, select the function Brake for output 2.

2-12- System checks before starting

- With the Enable input off, switch on the auxiliary 24V dc supply.
- Ensure that the STATUS display is lit.
- Apply power.
- If the Status display shows an error message check the list of error codes.

3- DPL software

3-1- DPL software installation

3-1-1- System configuration

- Minimum configuration :

⇒ Pentium PC
⇒ 32M Byte RAM
⇒ Hard disk (35 M Bytes free)
⇒ Microsoft® Windows™ 95, 98 , NT, 2000 and XP
⇒ CD-ROM (2X)
⇒ SVGA monitor
⇒ Mouse or other pointing device

- Recommended configuration :

⇒ Pentium® II PC
⇒ 64M Byte RAM
⇒ Hard disk (35 M Bytes free)
⇒ Microsoft® Windows™ 2000 or XP
⇒ CD-ROM (4X)
⇒ SVGA monitor
⇒ Mouse or other pointing device

This software can also function under Microsoft® Windows NT™. It does not function with Unix, Mac, MS-DOS and Microsoft® Windows 3.11.

3-1-2- DPL installation procedure

The software package “Drive Programming Language” is supplied on a CD-ROM. It should be installed as follows:

- Check that the system has the required configuration.
- Insert the CD-ROM in the appropriate drive.
- Follow the on-screen instructions

  The installation program runs..

- During the installation the user is asked for :

  1. destination directory
  2. type of installation (typical, compact, custom)
  3. program folder

Caution : only one level of program folder can be created.

The installation of the files begins and progress is indicated with a bar graph.

The installation ends with the addition of the DPL application icon in the programs folder.

3-2- DPL software structure

3-2-1- Directories

The default installation folder for the software is :
C:\Program Files\SERAD\Dpl\
It contains 4 sub-directories :

- Data: containing the sources of the software and the table of words for addressing by MODBUS.
- Help : containing the help files
- Lib : containing the various parameter files for the drive.
- Os : containing the drive operating system.

3-2-2- Project contents

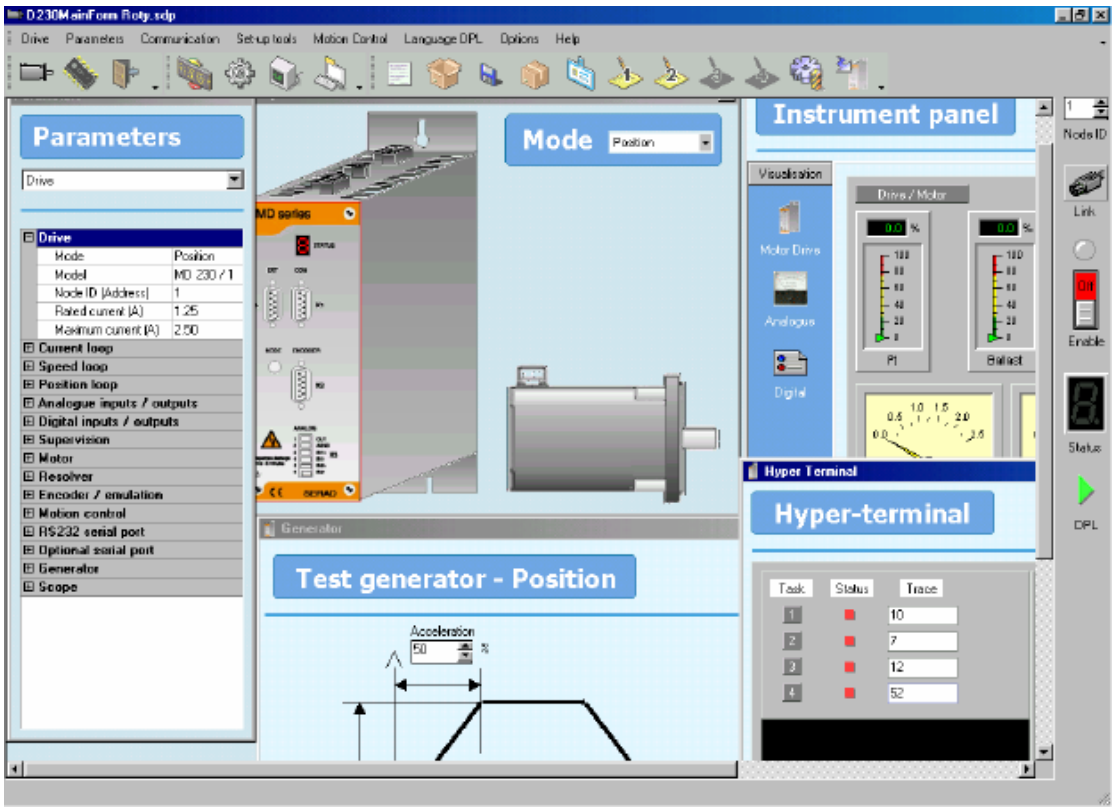
A project comprises a file .sdp and a folder having the same name.
The folder contains :

- Files (.dpl) containing the various tasks in text format.
- A file (.dpv) containing a list of variables and their values.
- A file (.dpi) containing information relating to the project.
- A folder (bin) containing the compiler output files and parameter files required by the drive.

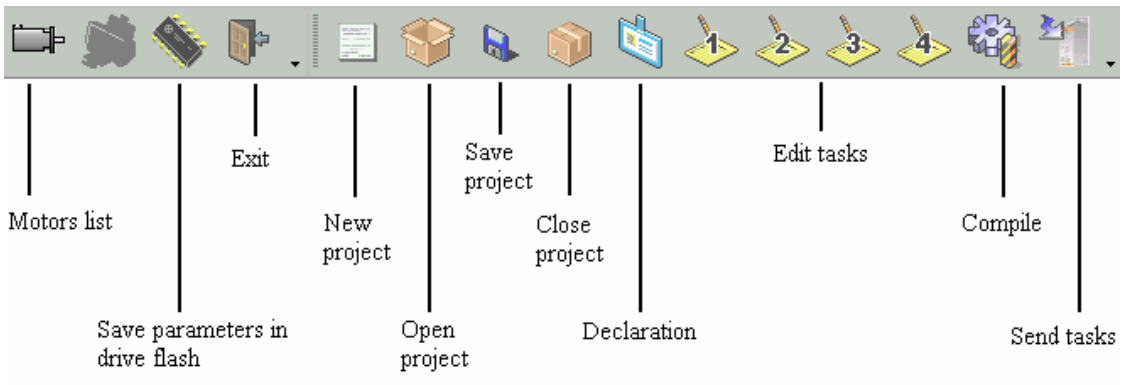
3-3- Presentation

3-3-1- Initial screen

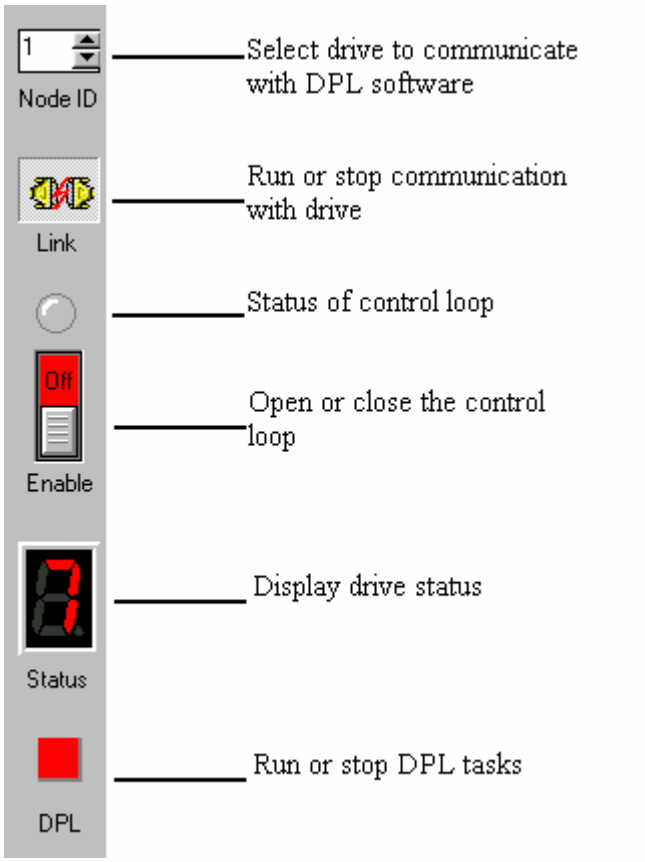
The DPL software is characterized by a main window that contains a menu bar, icon bar and a number of selectable windows. The ability to have multiple windows allows the user to simultaneously view several aspects of the drive.



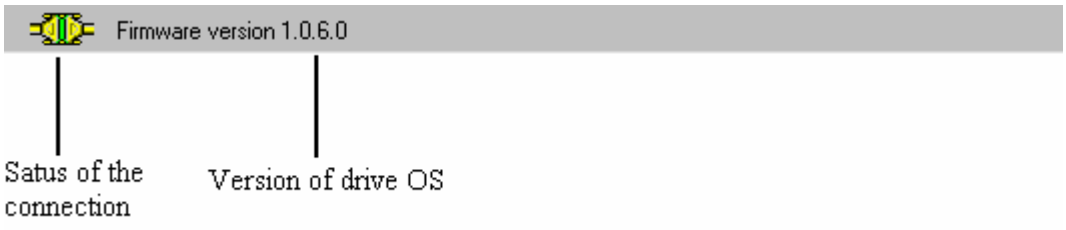
- Tools bar:



- Commands bar :



- State bar :



3-4- Menus and icons

3-4-1- Drive

- **Open motor file :**

Icon : 

Action : Load parameters from a file in the motor library.

- **Preferences :**

Icon : 

Action : Set up / alter the printing options.

- **Print :**

Icon : 

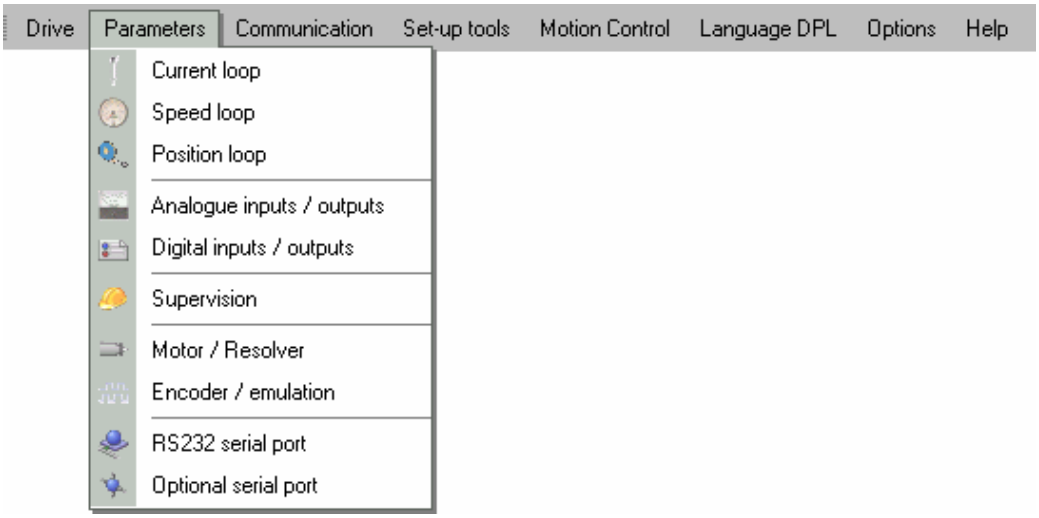
Action : Print the entire contents or selected items of a project.

- **Exit :**

Icon : 

Action : Exit the program.

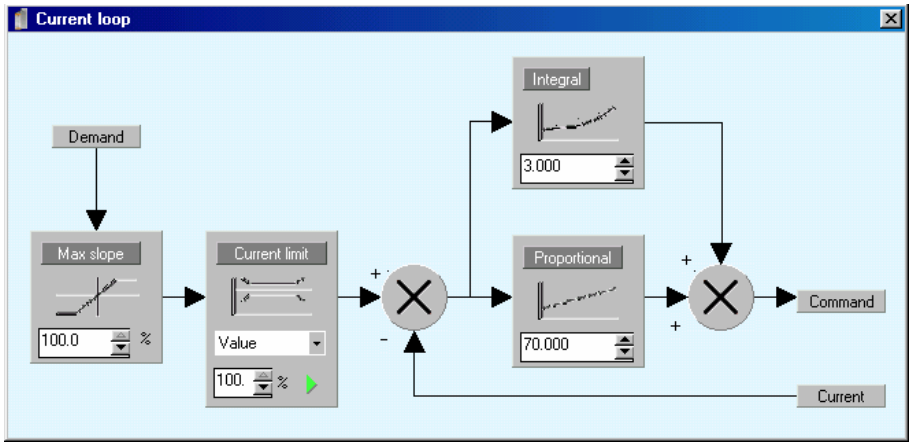
3-4-2- Parameters



• Current loop :

Icon : 

Action : Configure the drive's current loop parameters.



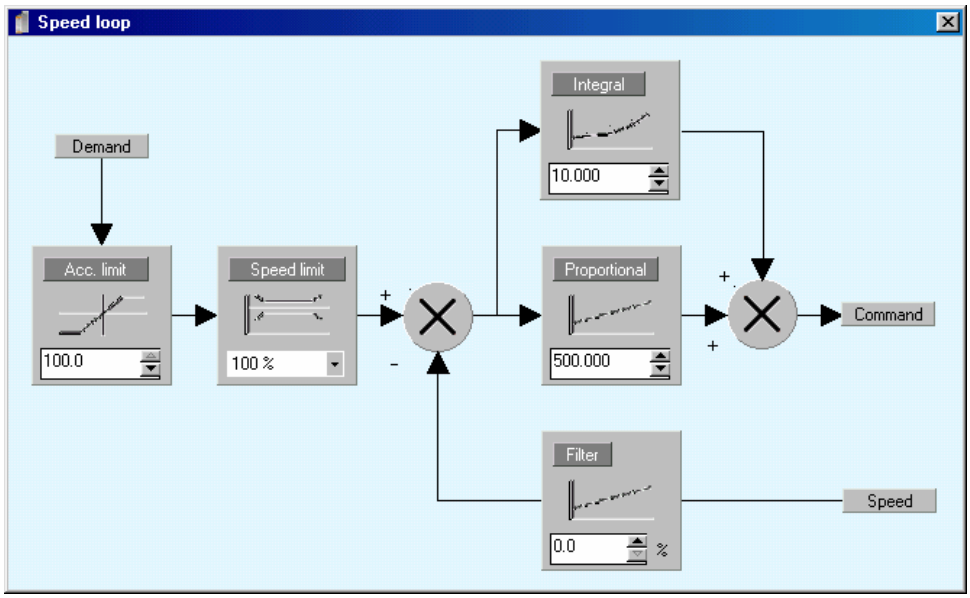
- Command : Select the command source :value, analogue input, speed loop.
- Acceleration limit : Limit the rate of change of current.
- Current limit : Limit the current as a percentage of the nominal value.
- Integral gain : Set the integral coefficient of the control loop.
- Proportional gain : Set the proportional coefficient of the control loop.

The acceleration limit and current limit are accessible only when the advanced parameter option has been selected (see Menu / Options/ Accessibility).

• Speed loop :

Icon : 

Action : Configure the drive's speed loop parameters.



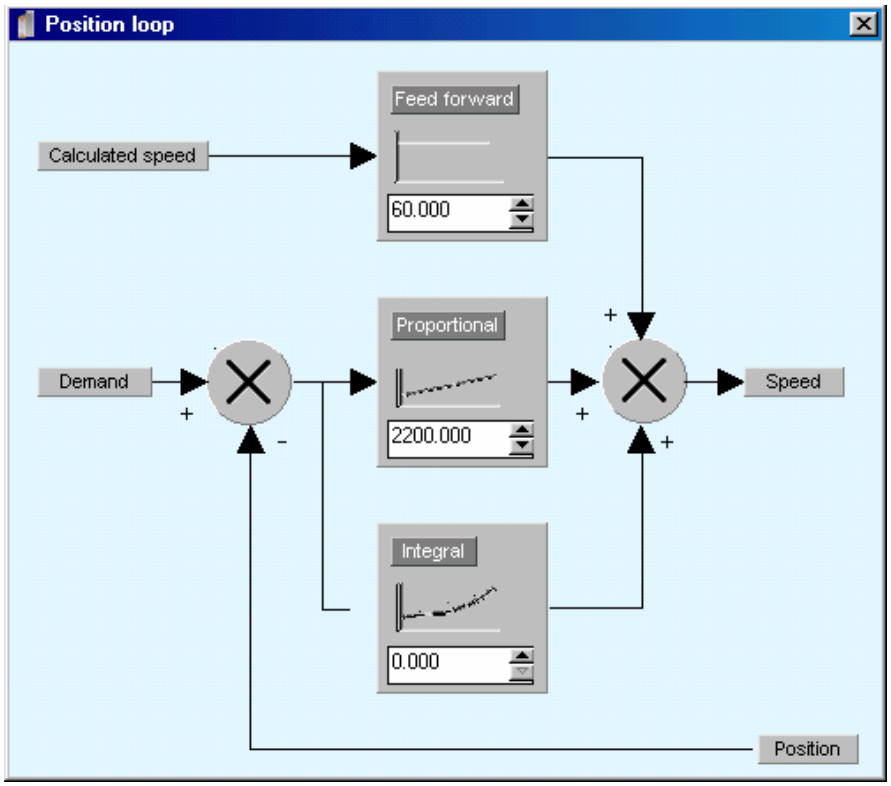
- Command : Select the command source : value, analogue input, position loop.
- Acceleration limit : Limit the rate of change of speed.
- Speed limit : Limit the speed as a percentage of the nominal value.
- Integral gain : Set the integral coefficient of the control loop.
- Proportional gain : Set the proportional coefficient of the control loop.
- Filter : Sets the filter time constant for the speed feedback.

The acceleration limit, speed limit and filter value are accessible only when the advanced parameter option has been selected (see Menu / Options/ Accessibility).

• Position loop :

Icon : 

Action : Configure the drive's position loop.



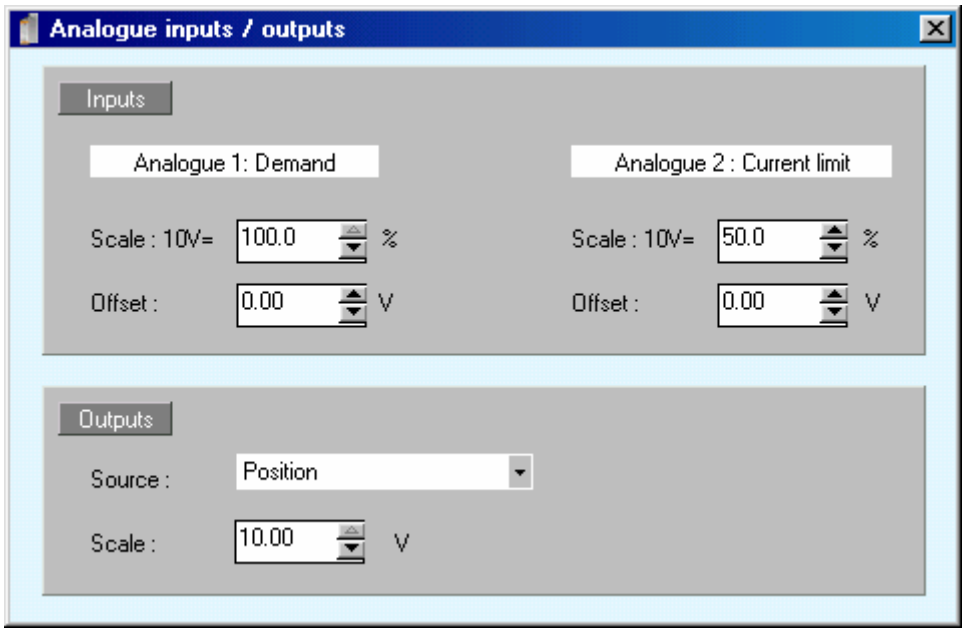
- Feed forward : The feed forward gain can be used to give a following error close to zero.
- Proportional gain : Set the proportional coefficient of the control loop.

The integral gain is accessible only when the advanced parameter option has been selected (see Menu / Options/ Accessibility).

• Analogue inputs / output :

Icon : 

Action : Configure the analogue I/O.



- Scaling : The percentage of the nominal value used for the speed or current command (depending on the mode) for a 10V signal on the analogue input.

  Example : Nominal speed = 3000 rpm

  Scaling = 50%

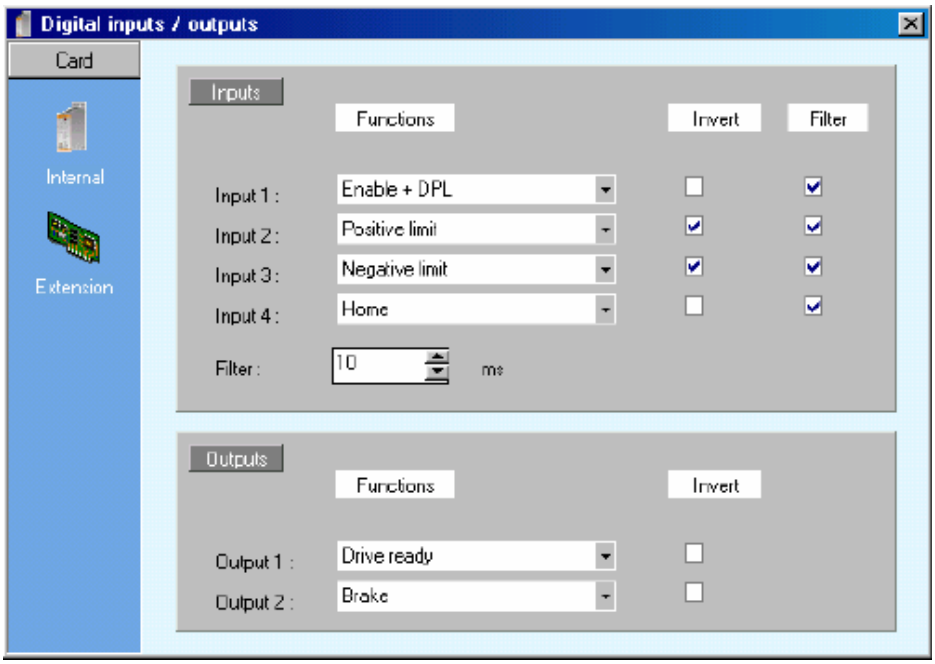
  Speed mode selected

  A voltage of 10V on the analogue input will give a speed of 1500 rpm
- Offset : Applies an offset voltage to the input signal.

• Digital inputs / outputs :

Icon : 

Action : Configure the digital I/O.



- Input 1 : Selection : Drive **Enable** or none.
  1. If **None**, the power stage of the drive is activated by the Enable button in the main DPL window or by an Axis On / Axis Off instruction in a DPL task.
  2. If **Enable**, control is done on rising edge of the logical input E1.
  3. If **Enable + DPL**, control is done on rising edge of the logicial input E1 and by the Axis Off instruction followed by Axis On of language DPL.

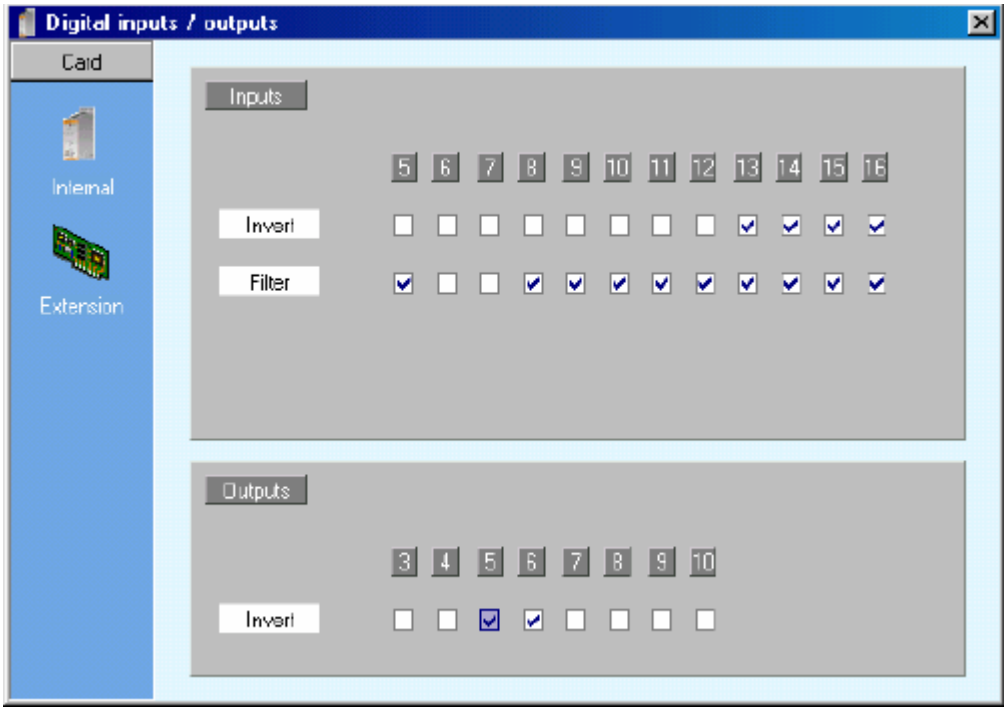
     Caution: So then the variator passes in open loop (that is to say by E1=0, Axis Off or a default), a new rising edge is needed on E1 and the Axis Off instruction followed by Axis On of language DPL for controlling the axis again.
- Input 2 : Selection : **Over-travel +** or none.
- Input 3 : Selection : **Over-travel –** or none.
- Input 4 : Selection : **Home limit**, **Fault reset** on the failing edge, or none.
- Filter delay : Value of the input filter delay in ms.
- Inversion : If inversion is not selected the input is activated with positive logic. If inversion is selected, the input is activated with negative logic.
- Filter : Activate filtering of the selected input.
- Output 1 : **Drive ready** or none.
- Output 2 : **Motor brake** or none

The output Drive Ready can be connected in series with the emergency stop control loop.

If the brake option is selected for output 2, it is necessary to add an external relay to control the brake as the output current from the drive is limited to 100mA.

The logic state of the brake output corresponds to the internal enable state of the drive.

In position mode, the urgent deceleration (Motion control \ Speed profile) is used to stop axis when limit censors are actives.



With an extension card, you can have :

- 12 additional inputs.
- 8 additional outputs.

•Supervision :

Icon : 

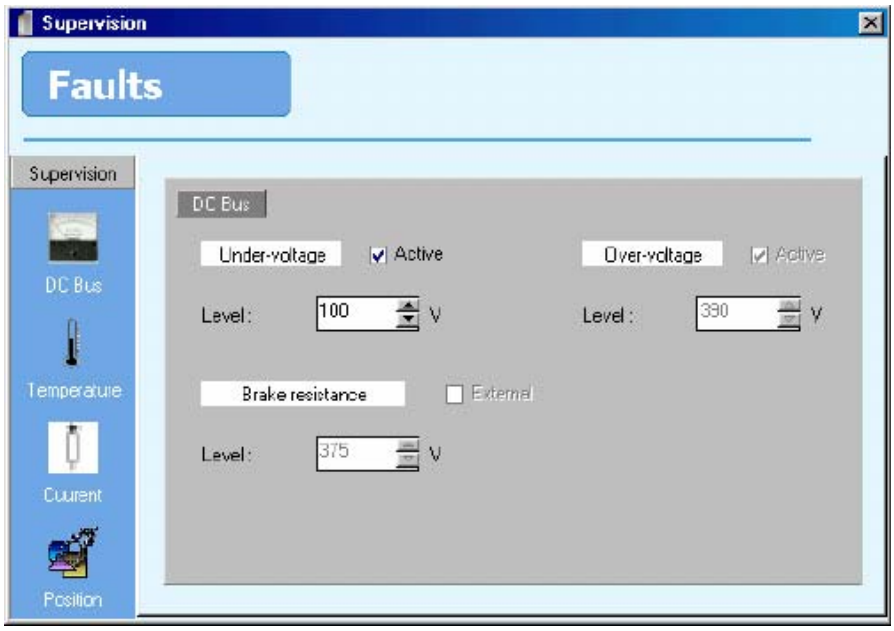
Action : Configure the security parameters.

1. DC Bus monitor :

 **Factory settings, do not modify.**

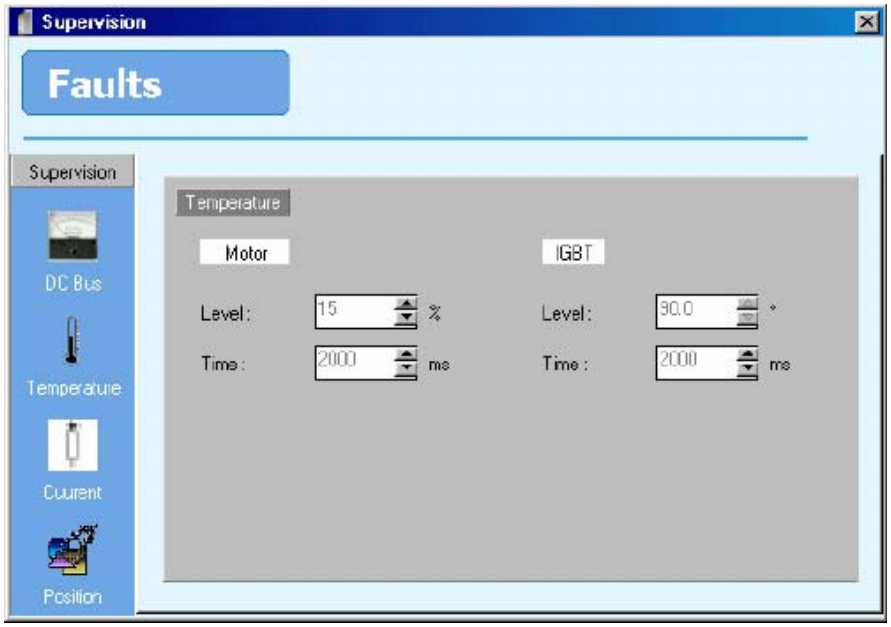
When an external brake resistor has been used select the tick-box External.

 **This resistance must be carefully chosen. The adjustments are only accessible when advanced parameters are selected.**



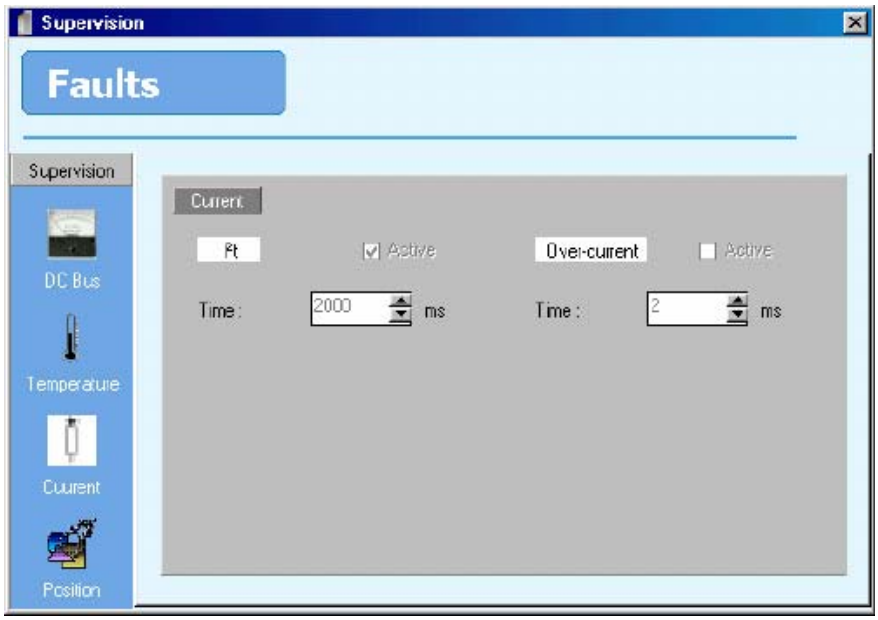
2. Temperature monitor :

Factory settings, do not modify.



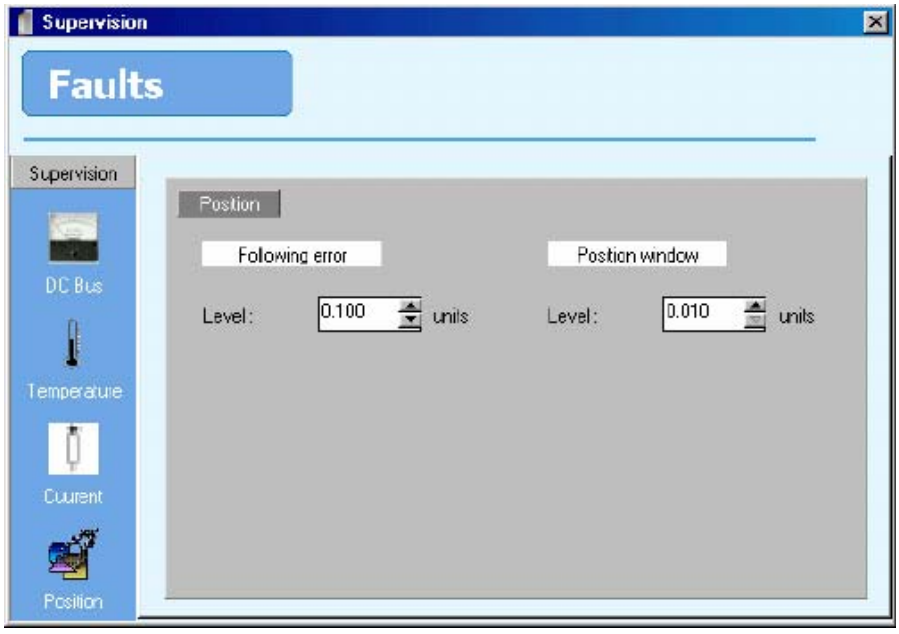
3. Current monitor :

Factory settings, do not modify.



4. Position monitor :

When the drive is used in position mode, control the following error to be as small as possible. The maximum permissible following error is 8 motor revs. The value of the following error limit should be as small as possible, for example 0.2 motor revs.



- Following error : The following error is monitored whenever the drive is enabled, either stopped or moving. If the difference between the calculated position and the actual position exceeds the following error limit the power

stage of the drive is disabled and an error code appears on the status display.

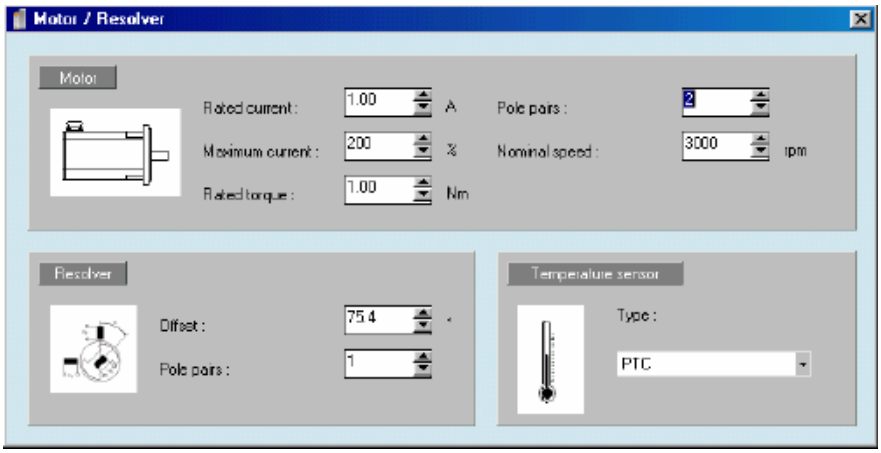
The control of this value is very important: a value too small can lead to spurious errors, a value too large can reduce the overall safety margins of the machine.

- Position window : A the end of a movement, the movement is considered to be completed only when the difference between the actual position and the theoretical position is less than the position window value.

• Motor / Resolver :

Icon : 

Action : Configure the motor and resolver.



1. Motor :

Rated current : The rated current of the motor in amps.

Maximum current : A percentage of the rated current. Default value 200%. This information is not used and is there for information only.

Rated torque : Rated motor torque in Nm.

Pole pairs : Must correspond to the motor being used.

2. Resolver :

Offset : Resolver offset.

Pole pairs : Set for 1 pair of poles

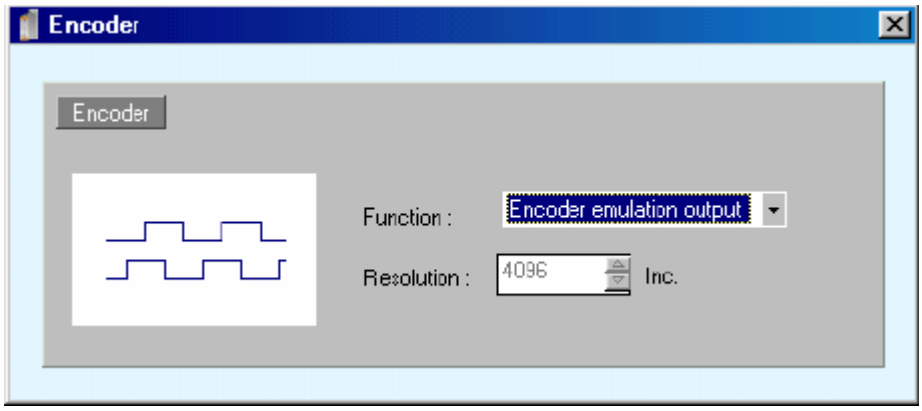
3. Temperature sensor :

Type : PTC or NTC

• Encoder :

Icon : 

Action : Set up for either encoder input or simulated encoder output.



Function : Selection : master encoder input or simulated encoder output.

Resolution : Master encoder : input the resolution in increments (4 increments per line). For example, for an encoder with 500 pulses per rev enter 2000 increments.

Simulated encoder output : Fixed at 1024 ppr (4096 edges)

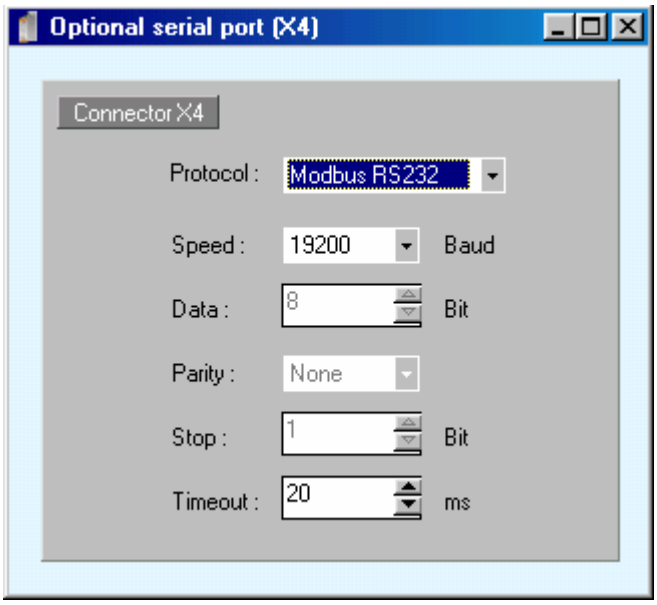
• RS232 serial port (fitted as standard) :

Icon : 

Action : Configure the port for Modbus.

The drive uses this connection in Modbus RTU slave mode.

The data format is fixed as 8 bits, 1 stop bit, no parity.



This window is used to set the transmission speed and the timeout in cases where the port is not using the system communication. When the port is using the system communication (set as the default in the menu Options / ComPC), the speed is fixed at 57600 bauds.

 With the system communication, the signal RTS from the PC is used and is forced to a logic 1.

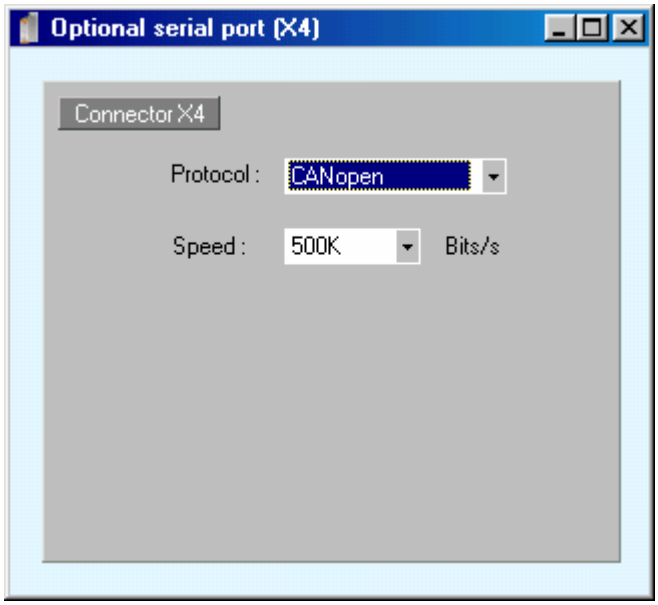
 The messages exchanged on this connection are always addressed to the slave number 1 (Modbus slave number = 1).

• Optional serial link :

Icon : 

Action : Configure the optional serial port for CANopen, RS232, RS422 or RS485.

- CANopen :



Speed : Define the communication speed used by the CANopen bus.

For more information, see the appendix relating to CANopen.

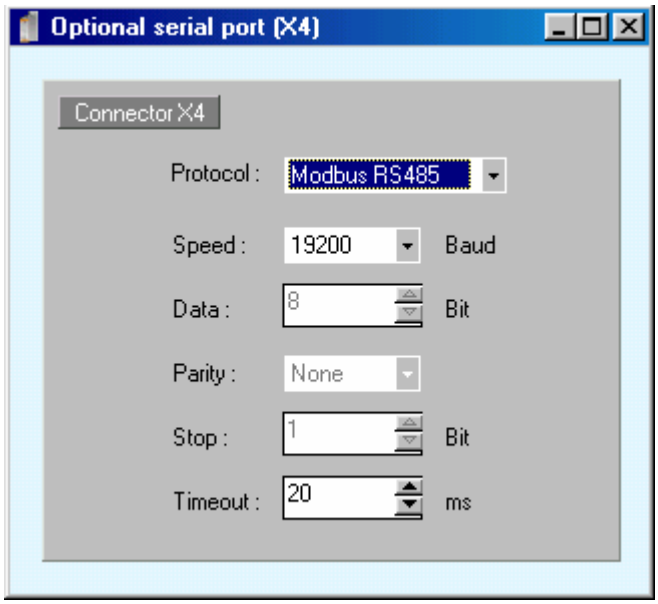
The Node ID of the drive is set on the rotary coded switch on the front face of the drive.

 Node ID = Switch position + 1. Example : Switch position = 5 gives Node ID = 6.

- Port RS232, RS422 or RS485 :

The drive uses this connection in Modbus RTU slave mode.

The data format is fixed as 8 data bits, 1 stop bit and no parity.



Settings :

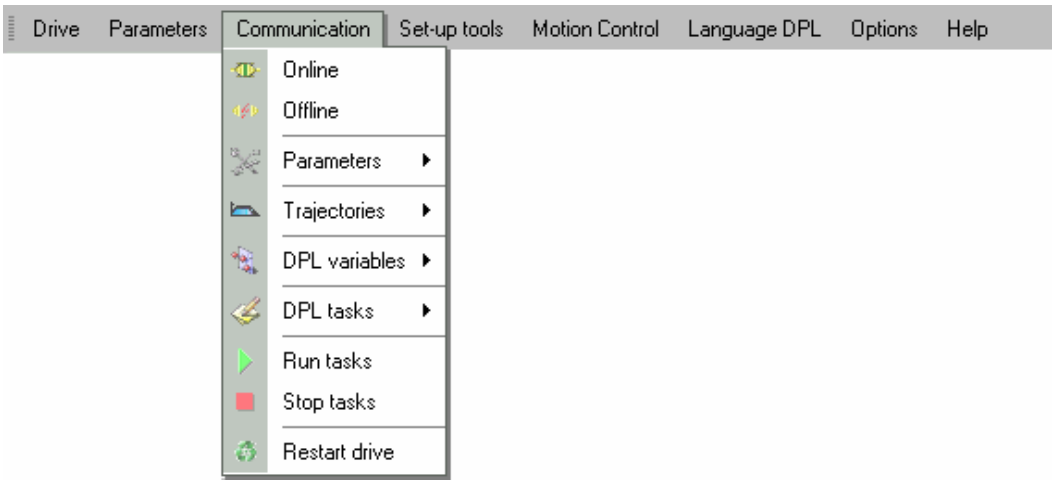
The Node ID of the drive is set on the rotary coded switch on the front face of the drive.

Node ID = Switch position + 1. Example : Switch position = 5 gives Node ID = 6.

Speed : Set the communication speed of the port.

Timeout: Maximum time without a response.

3-4-3- Communication



- **Online :**

Icon : 

Action : Establish communication with the drive. All parameters shown on the screen correspond with the values stored in the drive.

- **Offline :**

Icon : 

Action : Continue to work without being connected to a drive.

- **Parameters :**

Icon : 

Action : When working online you can :

- ***Send parameters PC -> Drive*** : send a parameter file from the PC to the drive. These parameters are automatically saved in the drive.
- ***Receive parameters PC <- Drive*** : save the drive's stored parameters in a file on the PC.
- ***Save drive parameters*** : transfer the current drive parameters to Flash memory. This allows them to be restored automatically after a supply interruption.

When working offline you can :

- ***Open a parameter file*** : allows the user to open and edit a parameter file.
- ***Save parameters in a file*** : allows the user to save a set of parameters to a file.

• Trajectories :

Icon : 

Action : Send or receive the 64 pre-programmed movements.

• DPL variables :

Icon : 

Action : Send or receive the initial values of the variables to or from the drive.

 Only variables VR0 toVR63 and VL0 to VL63 are applicable. At each power-on of the drive these 128 variables are loaded with these initial values.

• DPL tasks :

Icon : 

Action : Allows the user to send tasks to the drive or clear the tasks in the drive.

• Run DPL :

Icon : 

Action : Run all of the active tasks that are designated as automatic.

• Stop DPL :

Icon : 

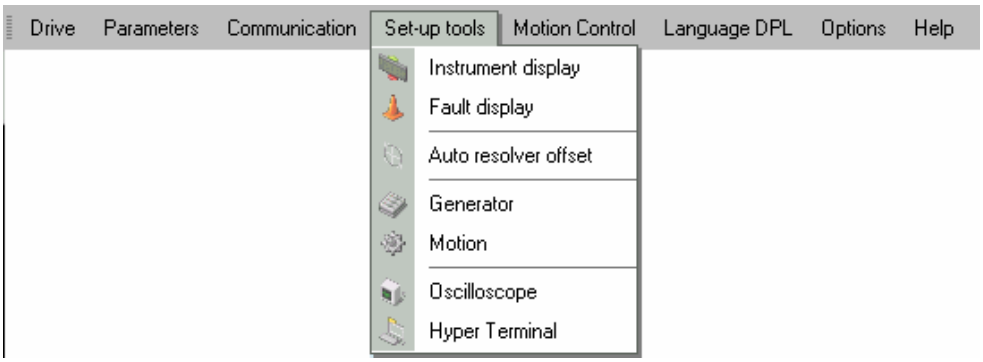
Action : Stops the execution of all of the tasks.

• Restart :

Icon : 

Action : Restart the drive.

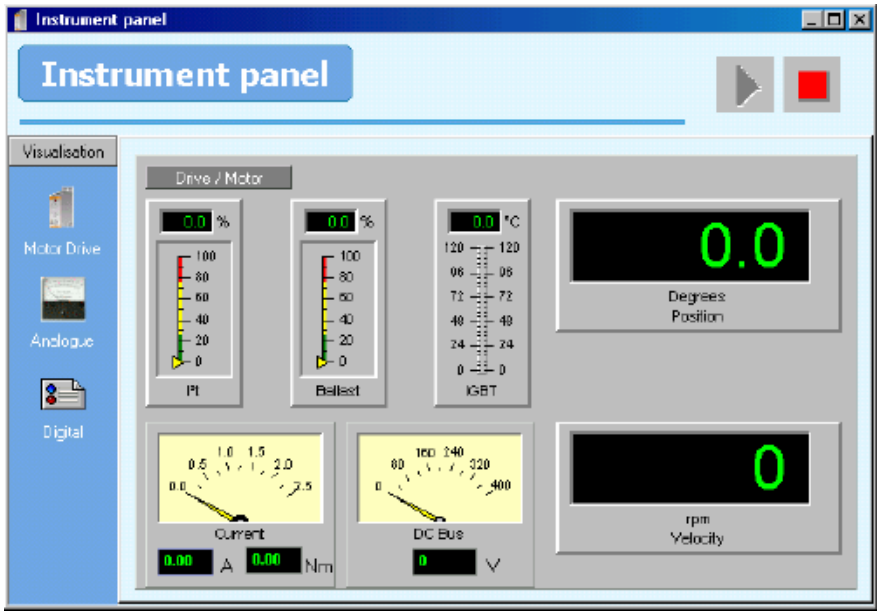
3-4-4- Diagnostics



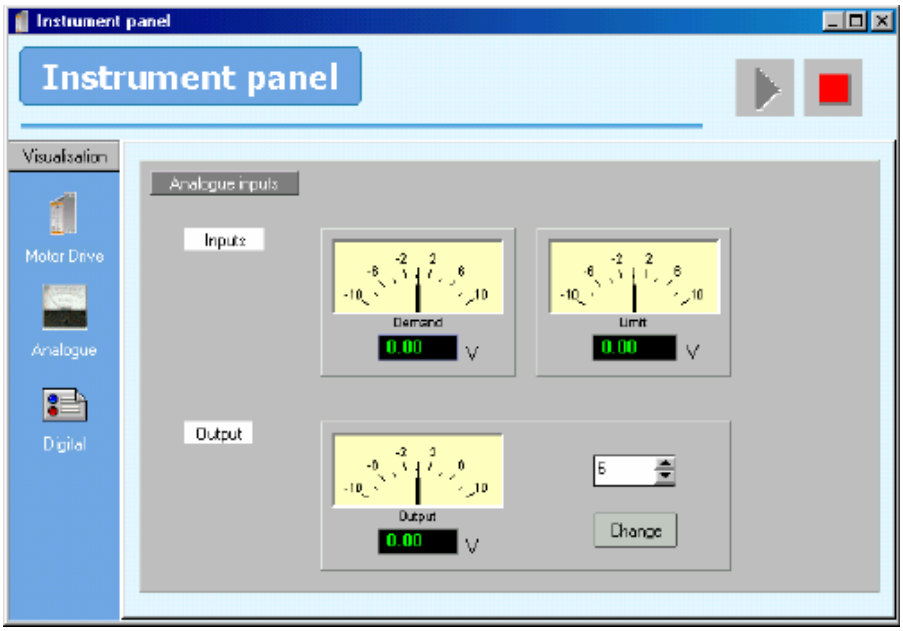
- **Instrument panel :**

Icon : 

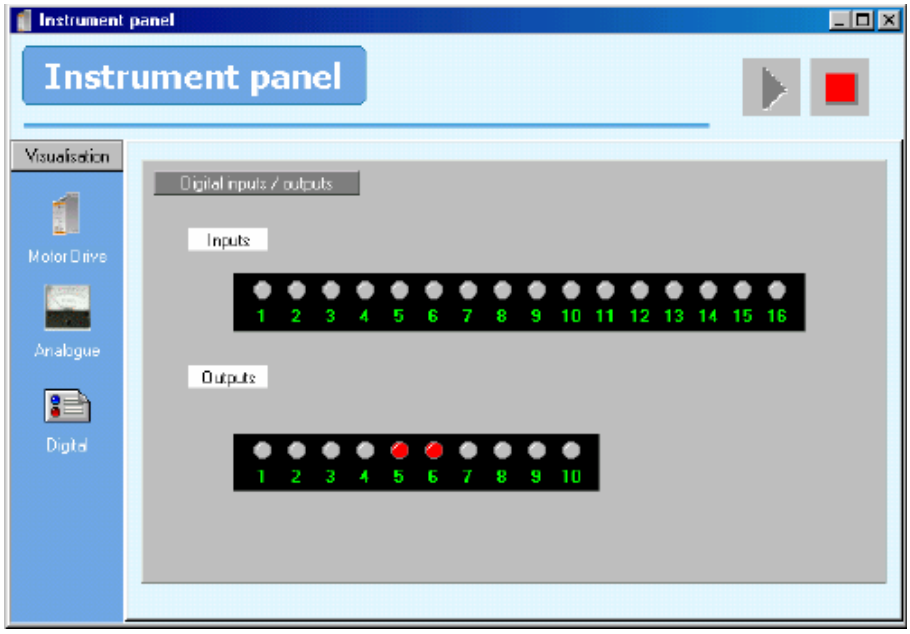
Action : Allows the user to see the internal state of the drive and motor.



Allows the user to see the digital I/O states and to change the outputs.



Allows the user to see the analogues I/O states and to change the output.



• Fault display :

Icon : 

Action : Displays the drive faults.

When a fault has occurred the fault can be reset by disabling and re-enabling the drive.

• Resolver auto-offset:

Icon :

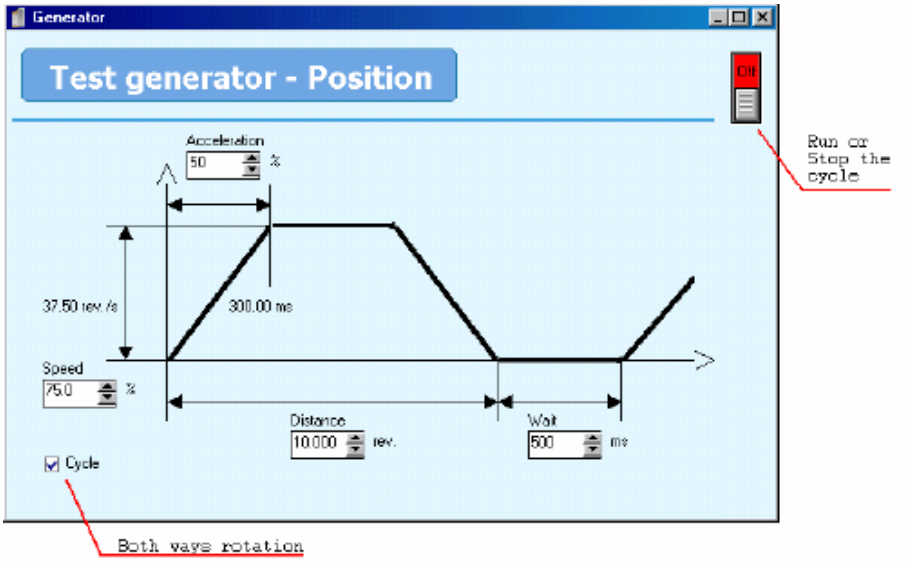
Action : Performs an automatic evaluation of the resolver offset.

Option only available with advanced parameters selected.

• Generator :

Icon :

Action : Generates a range of movements allowing the user to optimise the various control loops in the drive.

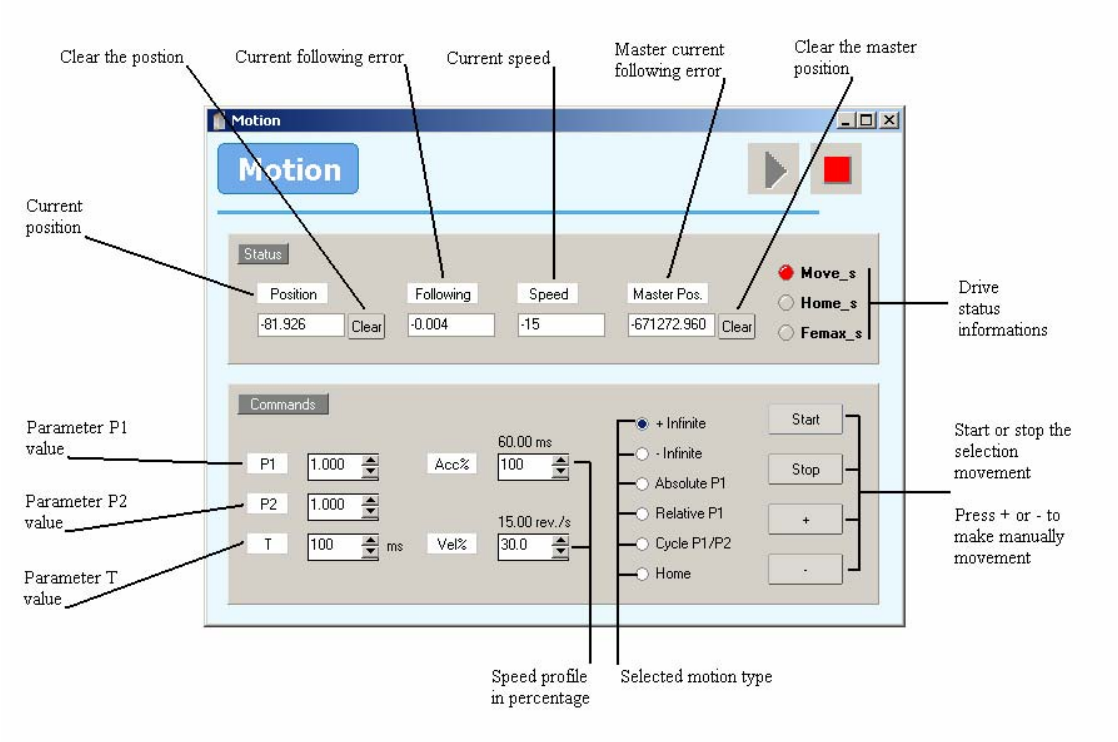


- Set up the generator to carry out the desired movement.
- Activate the drive with the ENABLE button (and / or Input 1).
- Start the movement with the ON/OFF button on the generator.

• Motion :

Icon:

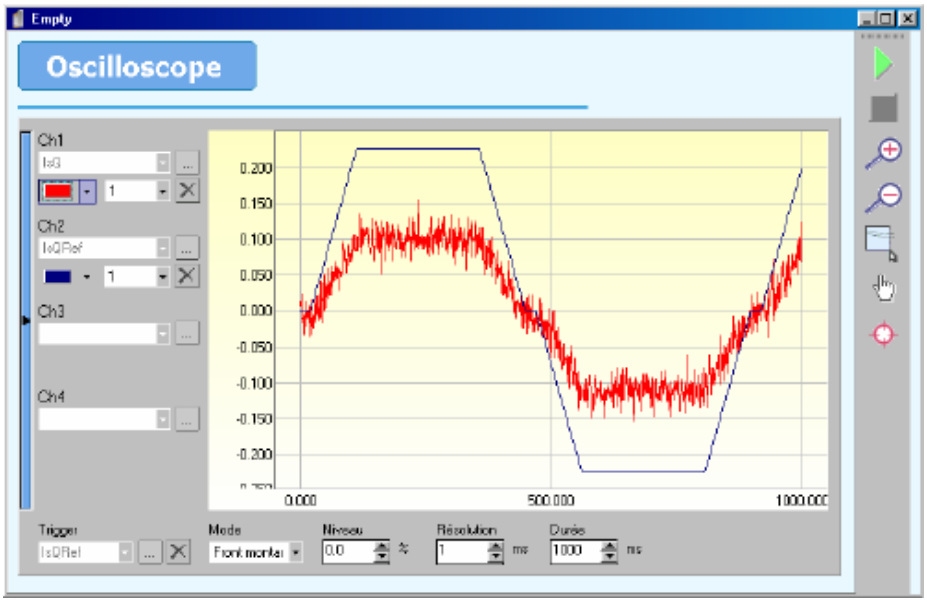
Action: Allows testing the loop of positioning of the axis. It is preferable to start by checking the behavior of the motor/drive by forcing the source with a value ranging between +10V and –10V (the axis must be in open loop). One can then passes in controlled mode and regulate the parameters of control. If one wishes to safeguard these modifications, it is necessary to make a safeguard of the parameters in the variator.



• Oscilloscope :

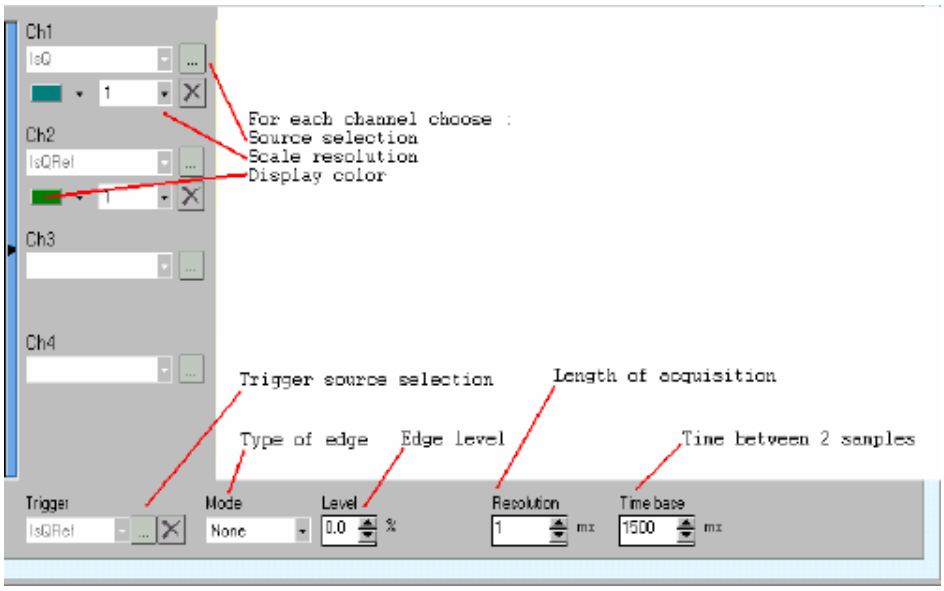
Icon :

Action : Opens the oscilloscope window. This tool aids commissioning by allowing all of the drive's parameters and states to be observed. Up to 4 channels can be observed simultaneously.
The oscilloscope is divided into three areas :
The display screen
The configuration control area
The display control area



↳ The display screen is the central part of the oscilloscope where the data are plotted.

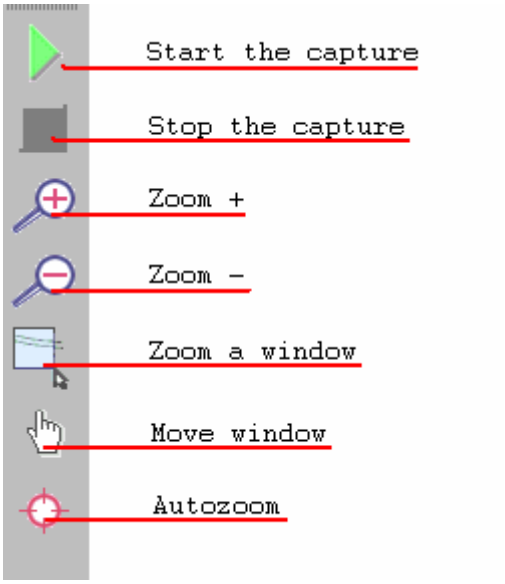
↳ The configuration controls make it possible to choose the signals to be displayed and to set up the mode of acquisition, the number of samples, duration etc.



Each signal is plotted in its own units, e.g. current in amps, speed in revs/min.

Each channel has a scaling factor allowing the amplitude of the signal to be amplified or attenuated.

➧ The display control area allows acquisition to be started and stopped, and also to modify the plotting on the display screen.

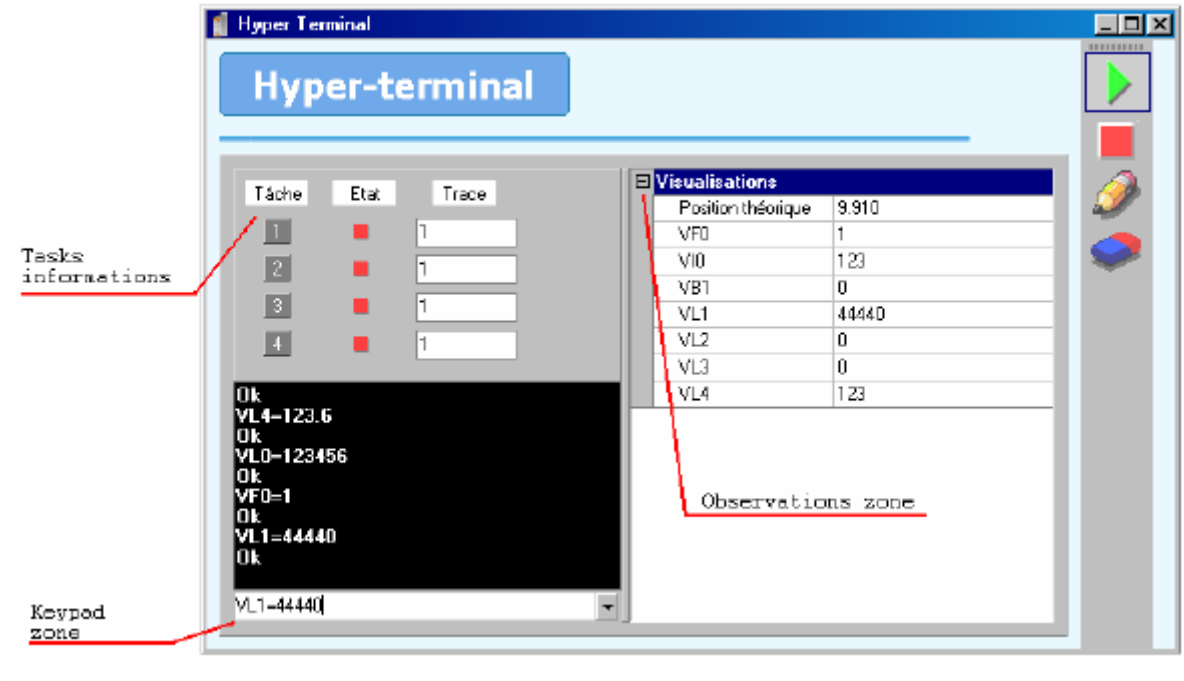


- Zoom window : Click on the button zoom window. With the button active, trace out a rectangle on the display screen by keeping the left button of the mouse pressed. Releasing the button completes the zoom

• Hyper terminal :

Icon : 

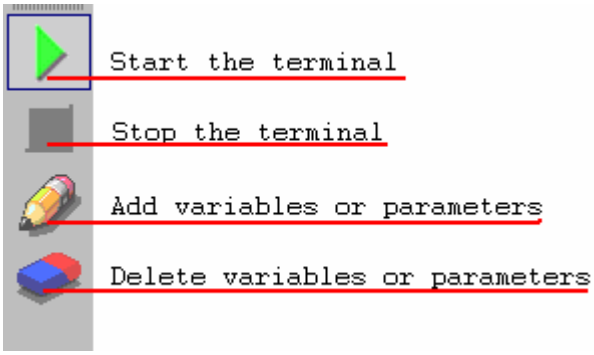
Action : Opens the hyper terminal. This tool aids commissioning by allowing the user to display variables, inputs, outputs and parameters relating to the internal state of the drive. It is also possible to directly modify variables.



➧ The main section of the window allows all variables and parameters to be read and written to in real time.

➧ <Variable or parameter name>=<Value> : assignment of a value to a variable or a parameter.

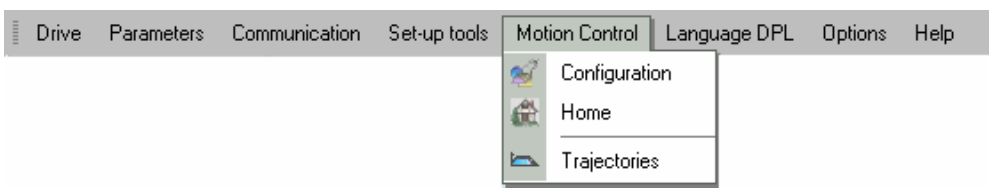
To facilitate the editing of variables or parameters, a configuration editor is available. This window regroups the various parameters and variables. By double clicking on the variable or parameter in this window, its name appears in the terminal screen.



➧The " observations " window allows continuous display of parameters and variables. The number of items displayed is limited to 16. Two command are used to add or delete an item from this display.

3-4-5- Motion control

Menu only available in position mode

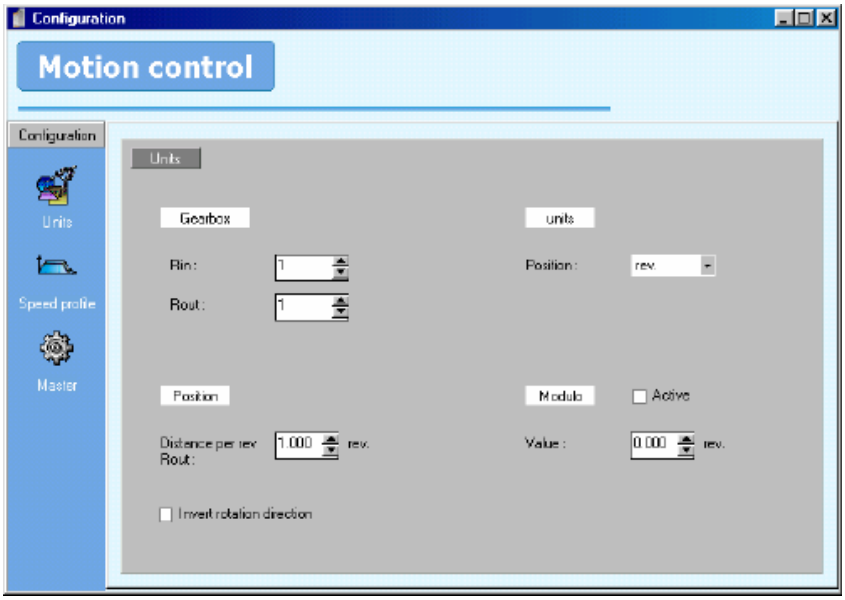


• Configuration :

Icon :

Action : Set the working units (mm, degrees …) as well as the default speed, acceleration and deceleration.

- Units :



Example 1 : Linear axis

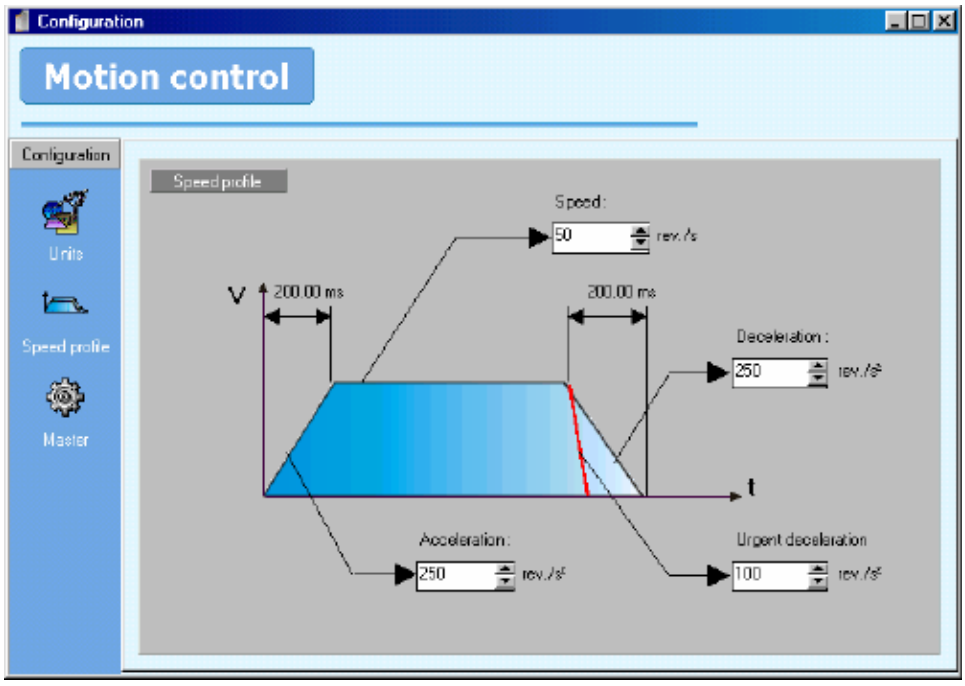
Motor connected to leadscrew with 5mm pitch. Units = mm, Rin = 1, Rout = 1, Distance par tour = 5.000, Modulo not active.

Example 2 : Rotary axis

Motor with 10:1 reduction gearbox. 360° rotary table on output of gearbox. Units = degrees, Rin = 10, Rout = 1, Distance per rev = 360.000, modulo active with a value of 360.000

Note : the number of decimal places is a parameter in menu *Options / Language DPL*

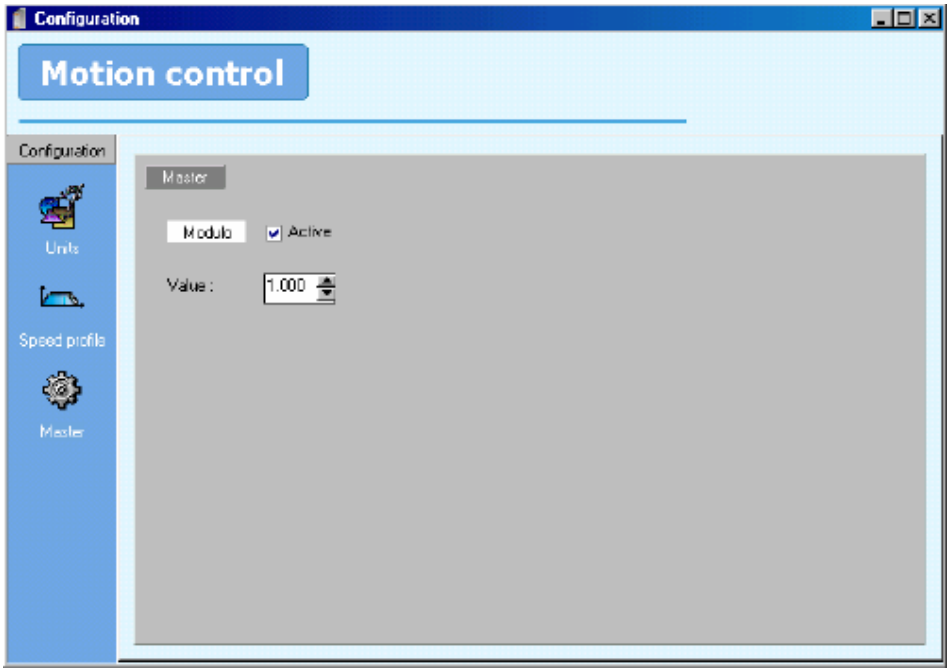
- Speed profile :



Speeds, accelerations and decelerations, expressed as percentages, are referred to these values.

The urgent deceleration is used to stop axis when limit censors are actives.

- Master encoder :

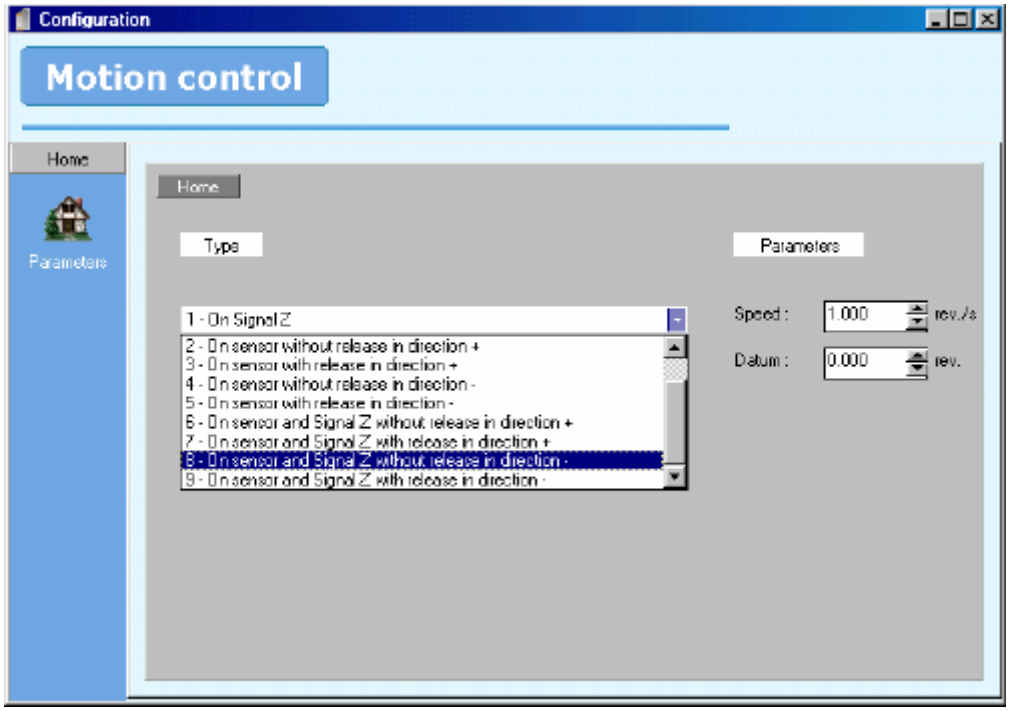


The master encoder uses the same units as the motor axis. Only in modulo mode can they be different.

• Home :

Icon : 

Action : Configure the homing mode.



- Homing method.
- Homing speed.

- Home position (0 by default)

• Trajectories :

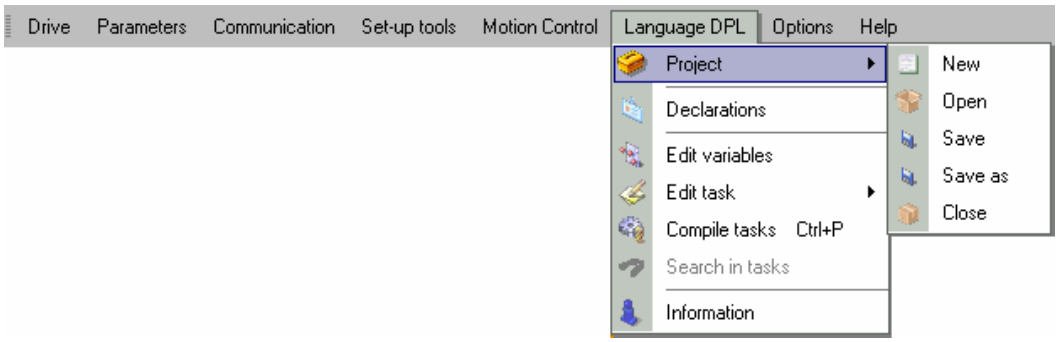
Action : Launches trajectories selected by the digital inputs.
See section on trajectory definition.

3-4-6- DPL language

• Project :

Icon : 

Action : Access the project menu.



1. New :

Icon : 

Action : Define a new project.

2. Open :

Icon : 

Action : Open an existing project.

3. Save :

Icon : 

Action : Save the entire contents of the project.

4. Save as :

Icon : 

Action : Save the project under a different name. This command creates a file and a directory having the same name but with extensions .sdp for the file and .data for the directory.

5. Close :

Icon : 

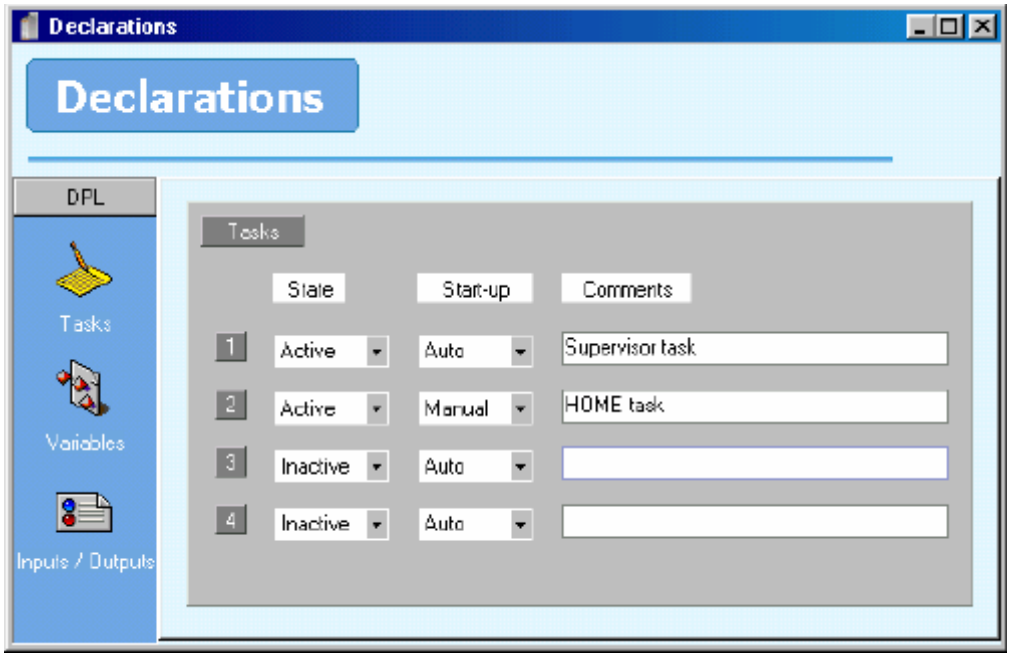
Action : Close the current project.

• Declarations :

Icon : 

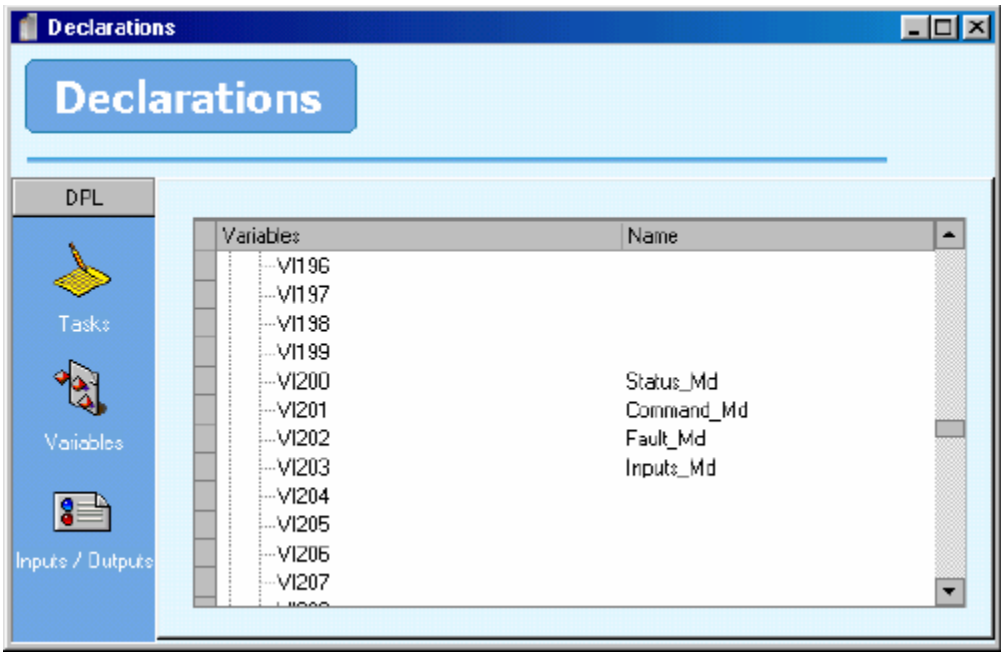
Action : Declares tasks, variable names and I/O names.

- Tasks:



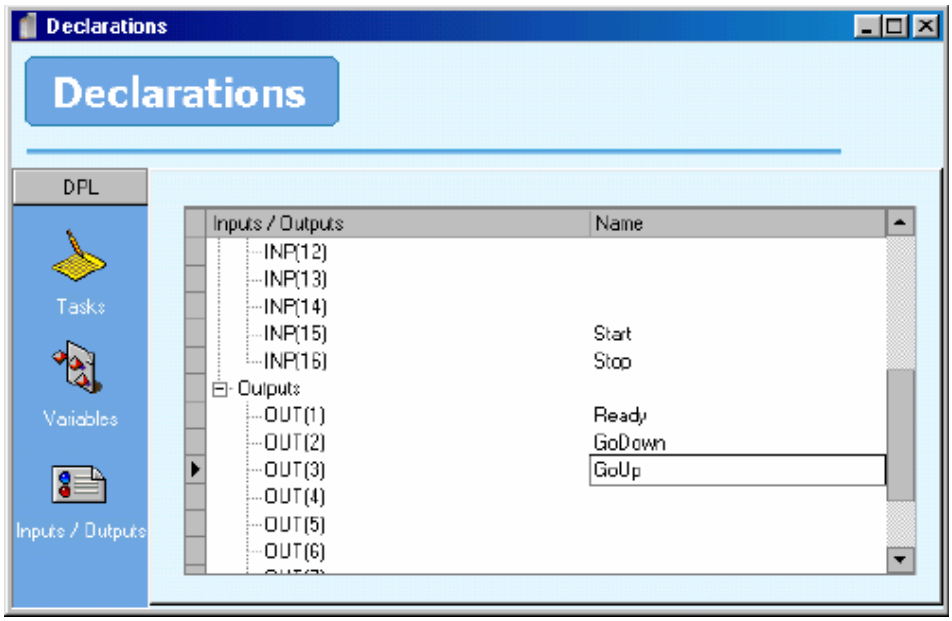
In this example the project contains 3 tasks. At power-on, Task 1 runs automatically.

- Variables :



Allows variables to be assigned names that can be used by the DPL tasks.

- Digital I/O :

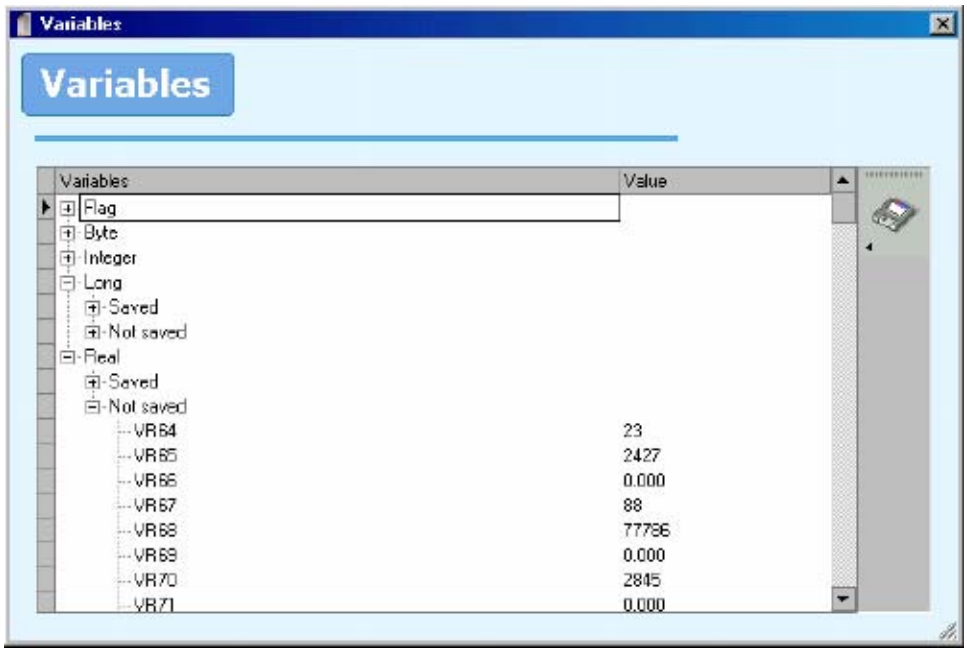


Allows I/O to be assigned names that can be used by the DPL tasks.

• Edit variables :

Icon : 

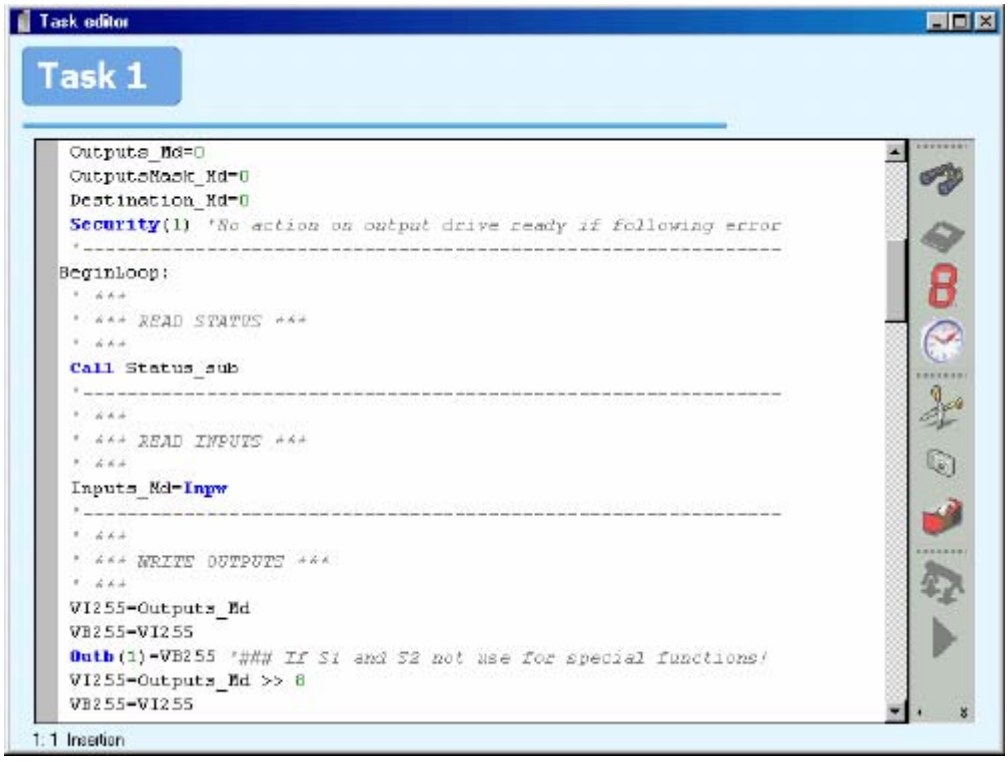
Action : Examine and modify variables (contained in the project file dpv) and send these to the drive using the command *Communication / Variables DPL / Send variables.*



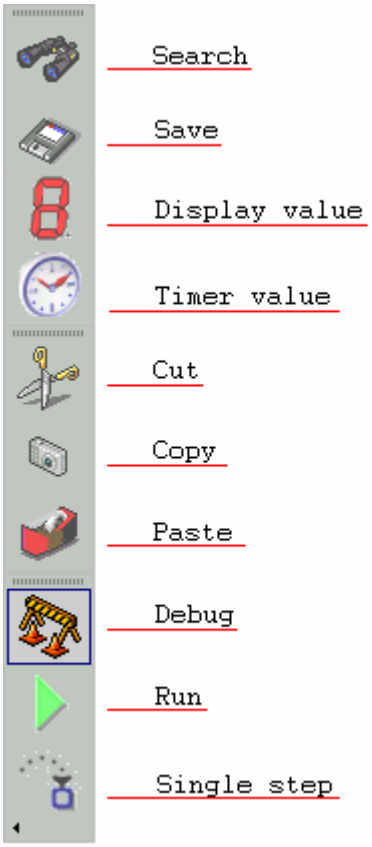
• Edit a task :

Icon : 

Action : The task editor allows the user to enter and modify the Basic code used by the program.



The tools used to simplify the editing process are :



- **Compile tasks :**

Icon : 

Action : Compile the tasks

- **Search tasks :**

Icon : 

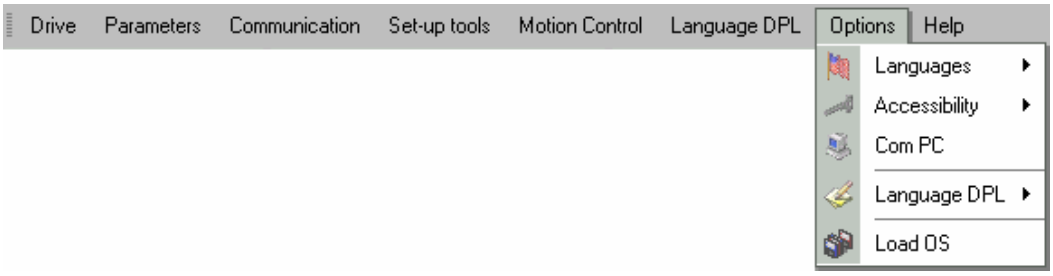
Action : Allows the user to search for a string of characters in the tasks.

- **Information :**

Icon : 

Action : Provides information on the program memory usage and other information associated with the project..

3-4-7- Options



• Languages :

Icon : 

Action : Select the language to be used by the software.

• Accessibility :

Icon : 

Action : Select the level of access to the various parameters. :

- Standard parameters
- Advanced parameters
- Restricted parameters

Select or de-select the DPL menu.

 **The modification of advanced and restricted parameters can have an adverse effect on the performance of the drive. This must only be carried out by suitably qualified personnel.**

• Com PC :

Icon : 

Action : Select the PC communication port : COM1, COM2, COM3 or COM4.

The option *System Communication* forces the PC and the drive to use a fixed format of : 57600 baud, 8 data bits , 1 stop bit, no parity, slave address = 1

In System Communication mode the RS232 parameters are not used.

 On activating *System Communication,* the PC forces RTS to a logic 1. When the drive sees a 1 on its CTS input the link is established.

• DPL language :

Icon : 

Action : Access to the DPL programming options.

- Precision : defines the number of decimal points used for real numbers. Variables (VR0 to VR63), position (POS_S in DPL) etc.
- Task ageing time : defines the maximum time spent in a task before switching to the next task. It is necessary to re-compile the tasks after a modification.

• Operating system :

Icon : 

Action : Download a new version of the operating system (firmware).

 **This should only be done by qualified personnel. The downloading affects the drive parameters. It is therefore necessary to re-load the parameters from a file.**

3-4-8- Help

• Help :

Icon : 

Action : Access the help files.

• Index :

Action : Search by group or keyword.

• About :

Icon : 

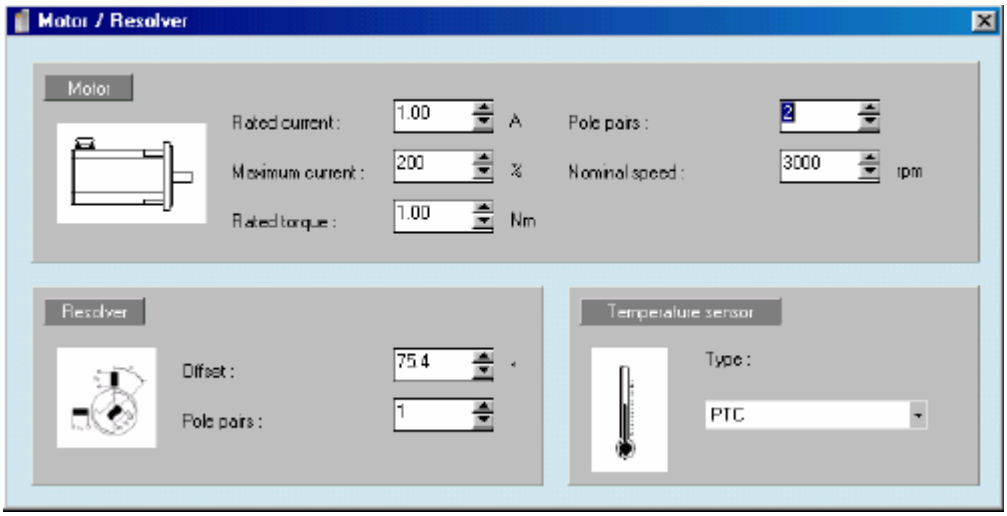
Action : Displays the current version of the software and drive firmware.

4- Drive adjustements

4-1- Motor and resolver parameter adjustments

If you have transferred a parameter file for the motor and drive combination in use then it will not be necessary to adjust the control loop parameters or the resolver offset.

- If not, the parameters can be adjusted by selecting the menu **Parameters/motor resolver**. The following menu is displayed :



4-2- Motor adjustments :

Refer to the motor manufacturer's data or the motor nameplate.

- Enter the motor parameters (rated current, maximum speed etc).

In normal situations, enter a maximum current of 200% of the rated current.

4-3- Resolver adjustments :

The resolver must be a TAMAGAWA TS2620N21E11 or equivalent. For other resolver types, verify suitability before use.

- Verify that the SINE and COSINE signal of the resolver vary between +0.9 and –0.9. This should be done using the software oscilloscope function as follows:

1. Supply the drive with 24V DC only (connector X6); the resolver and the RS232 serial link already being connected.
2. Open the **oscilloscope** in the **diagnostic tools** menu.
3. Select the signals SINE and COSINE in RESOLVER then start the data acquisition.
4. Turn the motor by hand and observe the signal traces. If the highest and lowest points of signals exceed +0.9 or –0.9, go to the list of resolver parameters (accessible with the advanced parameters option) and reduce the value of *Gain excitation*. If the signals are too weak (between +0.5 et –0.5), contact our technical department.

- Resolver offset adjustment :

  1. Provide the drive with its main AC supply.
  2. Enter **options** then **accessibility** and select **advanced parameters**.
  3. Enter **diagnostic tools** and select **auto resolver offset.**

  The drive will energise the motor windings and automatically measure the resolver offset. This step lasts only a few seconds.

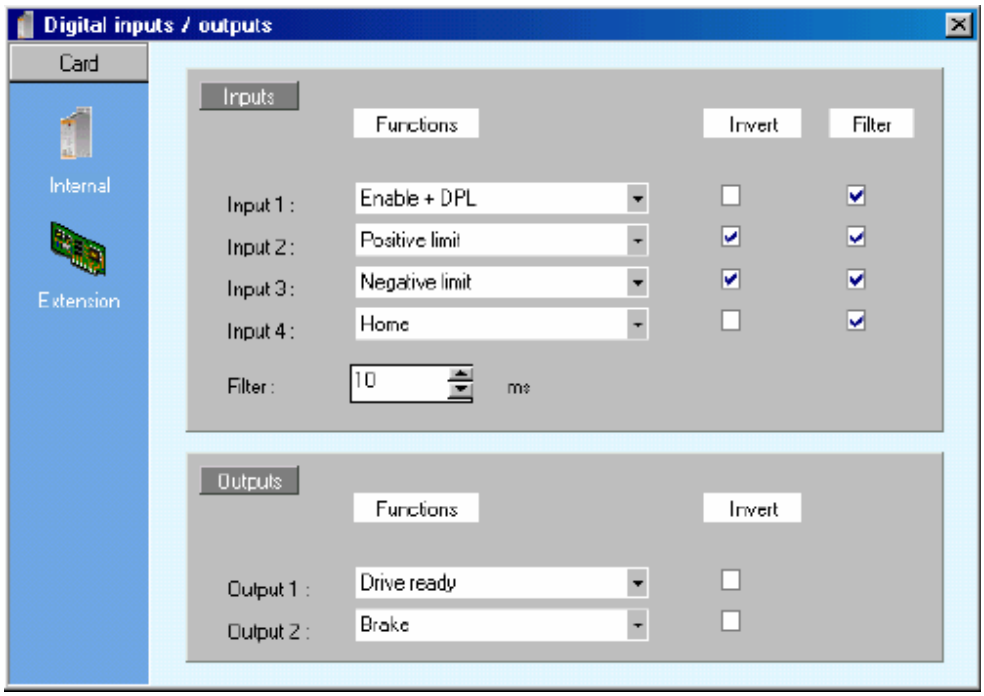
- Close the parameter window.
- Save the parameters.

Note : The number of resolver pole is fixed as 1 pair.

4-4- Adjustment of drive enable mode

To facilitate adjustment of the various control loops the drive enable mode should initially be set as follows :

- Select the menu **Parameters/Digital inputs outputs.**



- Select **None** in the filed **Input 1**. (At the end of the control loop adjustments this should be reset according to the requirement s of the system).

  The Enable button in the main window Enable can now be used to enable and disable the drive.

- Save the parameters

4-5- Operating modes adjustements

4-5-1- Operating modes

The MD series drives have 3 operating modes requiring various internal control loops.

- **TORQUE MODE** Current loop.

In torque mode, the motor maintains the specified torque. The speed depends on the applied load.

- **SPEED MODE** Current loop.

  Speed loop.

In speed mode, the motor maintains the specified speed irrespective of the load.

- **POSITION MODE** Current loop.

  Speed loop.

Position loop.

In position mode, the motor follows the demanded trajectory.

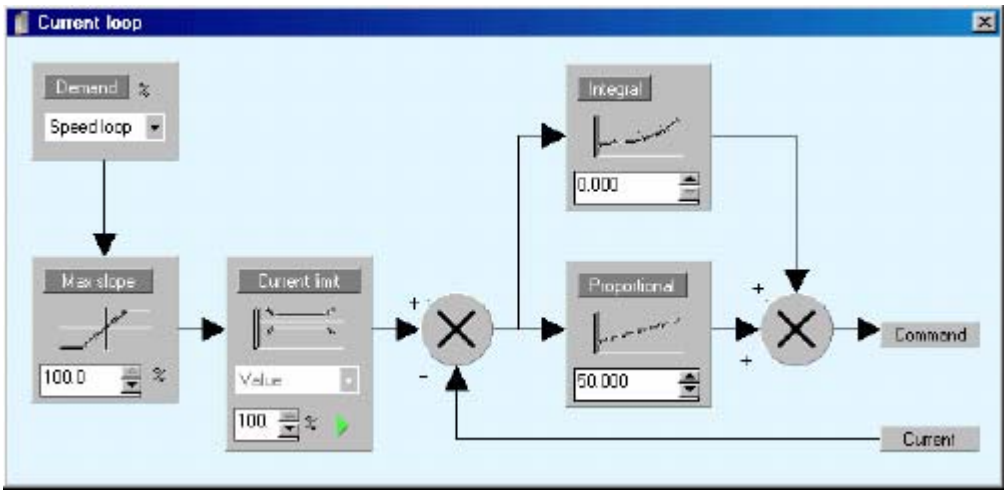
The choice of operating mode is made in the PARAMETERS window on the line Drive. Select one of the three modes (TORQUE, SPEED, POSITION)

The drive must be disabled before changing the mode.

4-5-2- Current loop adjustment

Good control of the current loop is required before it is attempted to optimise the speed loop and subsequent stages. The parameters are **integral gain** and **proportional gain**. This adjustment is directly linked to the characteristics of the motor and does not depend on the load.

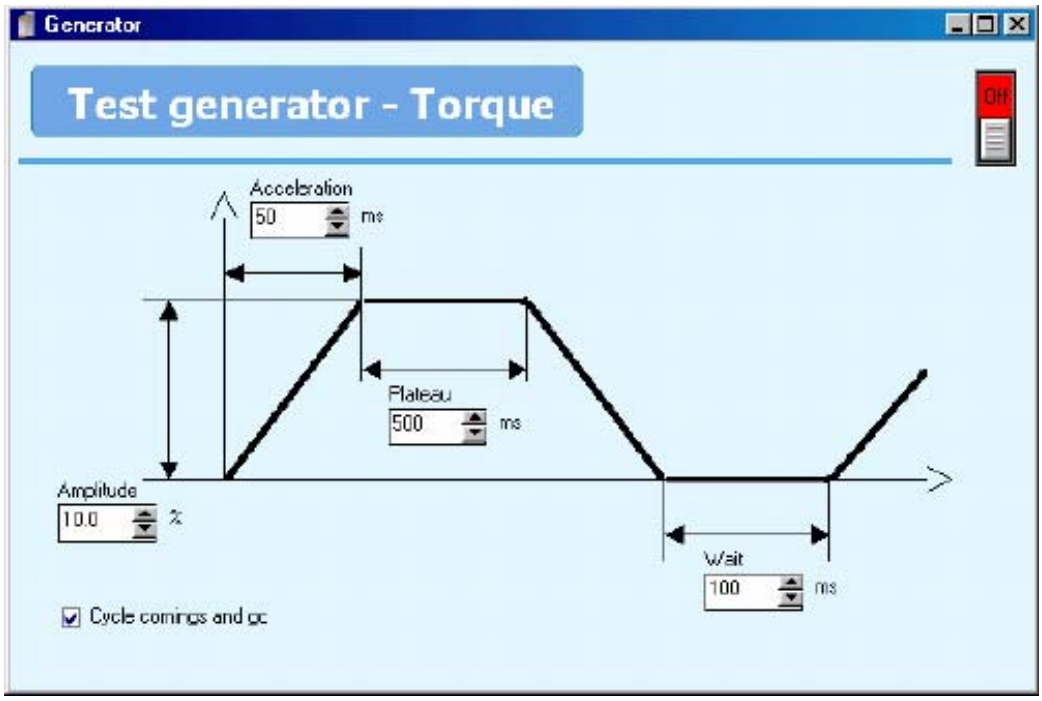
- Disable the drive (*Enable* button OFF in the main window).
- Select torque mode in the main window.
- Select the menu **Parameters / Current loop**. The following menu appears:



To start the current loop adjustments use the values shown above.

The command source must be of type : **value**.

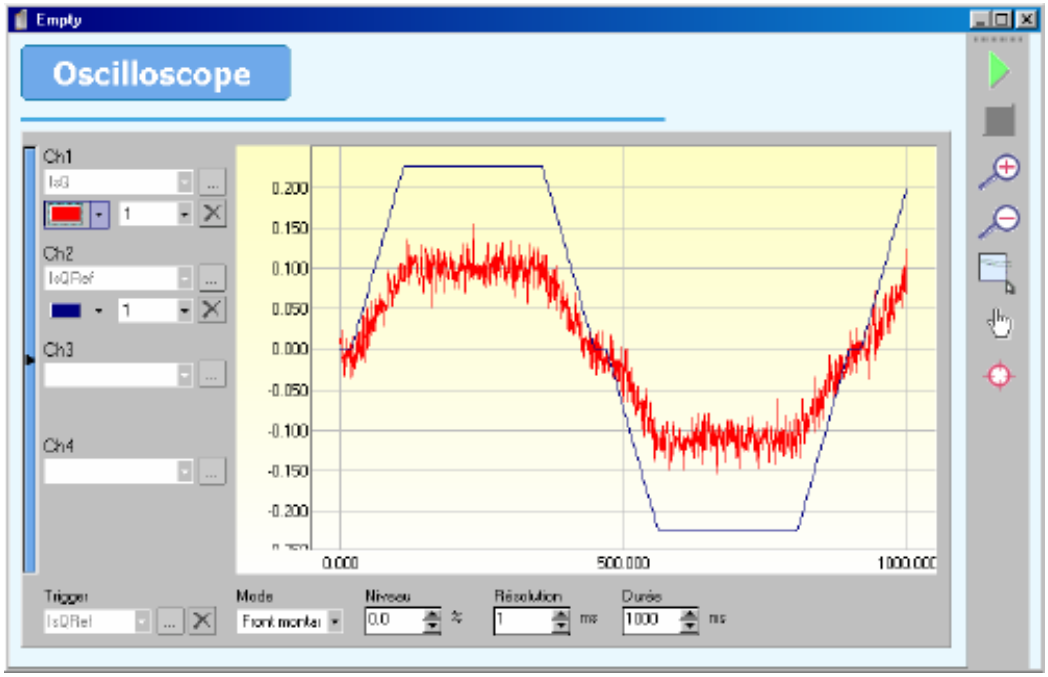
- In **Diagnostic tools / Generator**, start a movement as shown below :



You can adjust the amplitude between 5 and 15 % and the acceleration between 50 and 100%, according to the type of motor. The amplitude is expressed as a percentage of the maximum motor current.

To start the movement you must enable the drive by putting the *Enable* button to the ON position in the main screen.

- Use **Diagnostic tools / Oscilloscope** to observe the form of the current during the movement :



1. Select **IsQ** in **Current loop** for channel 1.
2. Select **IsQREF** in **Current loop** for channel 2.
3. Select **IsQREF** as the trigger and choose rising edge.

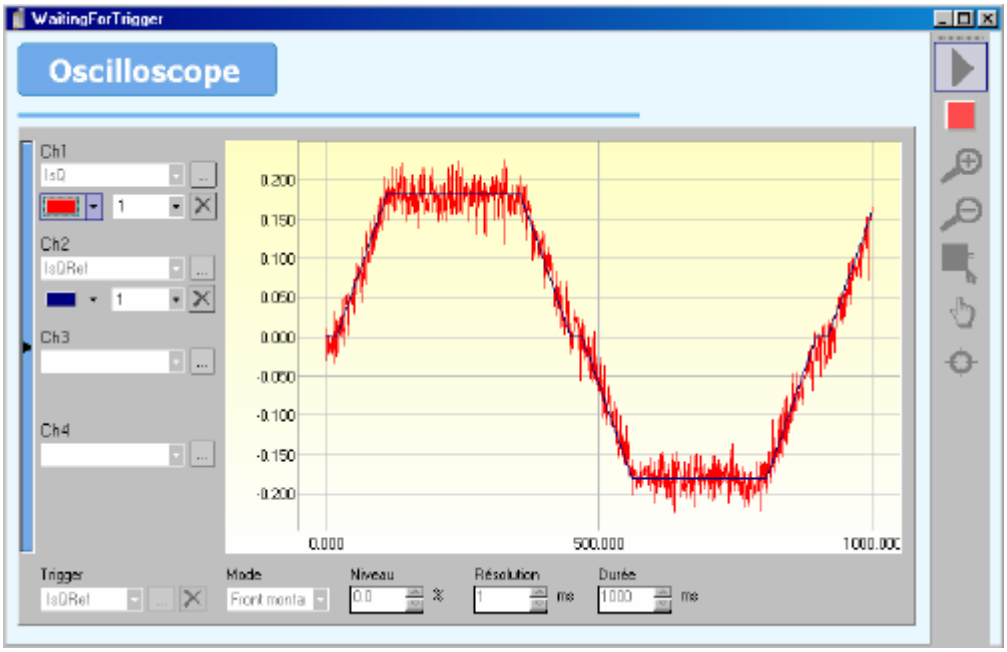
If the signal IsQREF is not trapezoidal, adjust the generator parameters.

- Before starting it is preferable to lock the motor shaft.

1. Increase the proportional gain until the actual current (IsQ) is as close as possible to the command (IsQREF).
2. If the motor vibrates, reduce the gain by 20%.
3. Increase the integral gain until the actual current follows the command exactly.

Typical values : proportional gain from 30 to 500, integral gain from 1 to 10.

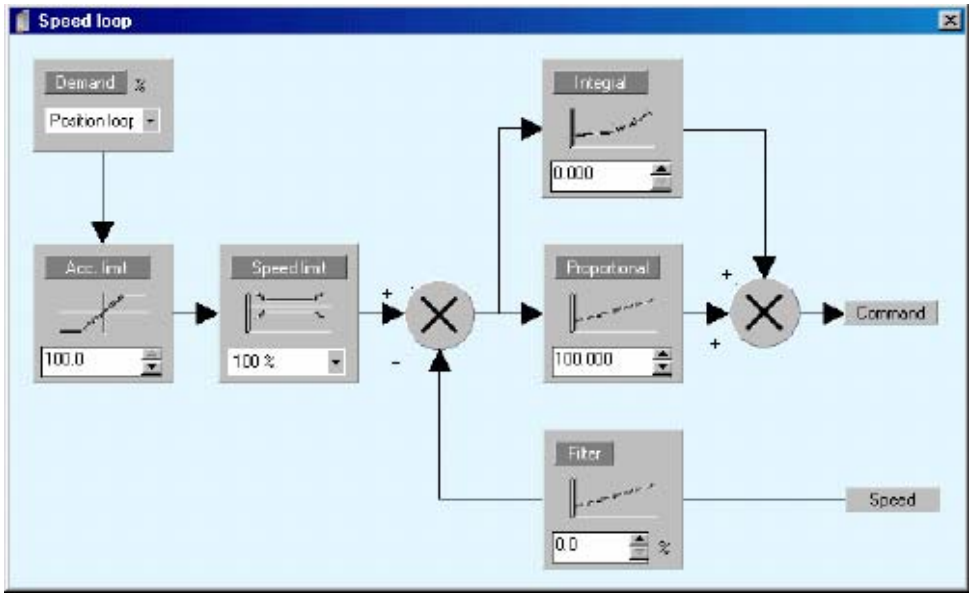
Typical curves for optimised gains.



- Save the adjustments using **Parameters/Save parameters**.

4-5-3- Speed loop adjustment

- Disable the drive (*Enable* button OFF in the main window).
- Select **speed** mode in the main window.
- Select the menu **Parameters / Speed loop**

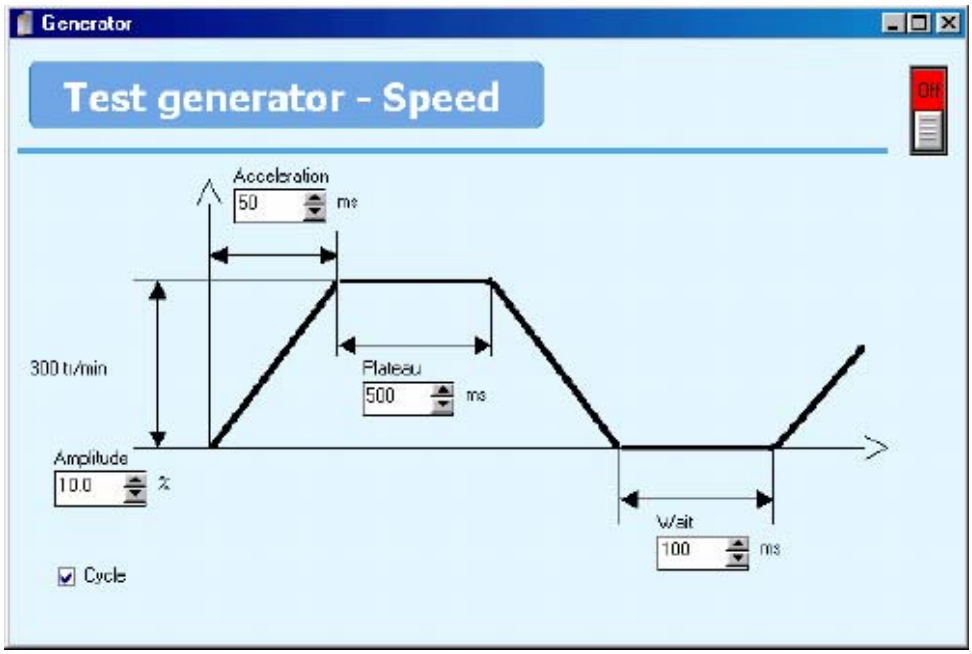


To start the speed loop adjustments use the values shown above.

The command source must be of type : **value**

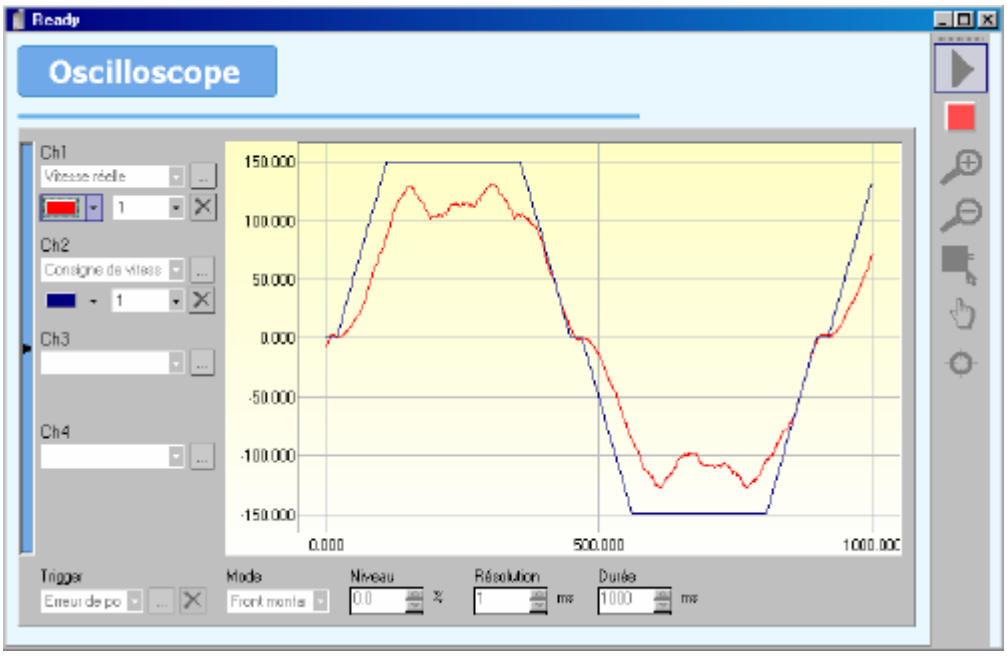
- Enable the drive (*Enable* button ON in the main window).
- In **Diagnostic tools / Generator**, start a movement as shown below :

The motor shaft must be free to rotate. Optimum adjustment of the speed loop is done using a loaded motor.



1.

- Use **Diagnostic tools / Oscilloscope** to observe the form of the speed during the movement :



1. Select **Actual speed** in **Speed loop** for channel 1.
2. Select **Speed command** in **Speed loop** for channel 2.
3. Select **Speed command** as the trigger and choose rising edge.

If the signal speed command signal is not trapezoidal, adjust the generator parameters.

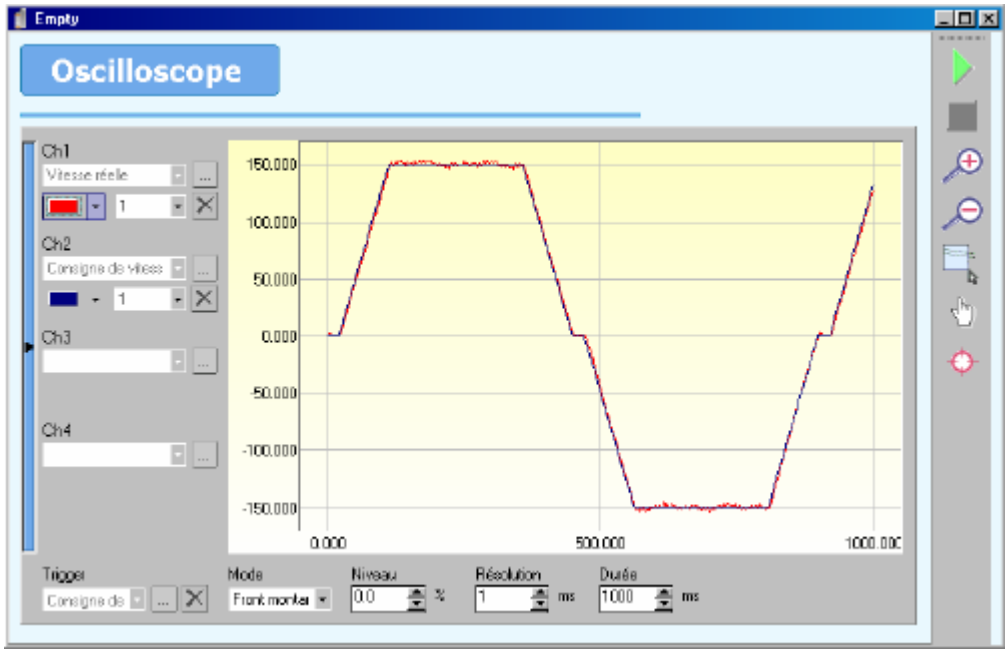
- Increase the **proportional gain** until the actual speed is as close as possible to the command.

If the motor vibrates, reduce the **proportional gain** by 20%.

Increase the **integral gain** until the actual speed follows the command exactly.

Typical values : proportional gain 200 to 1000, integral gain 1 to 20.

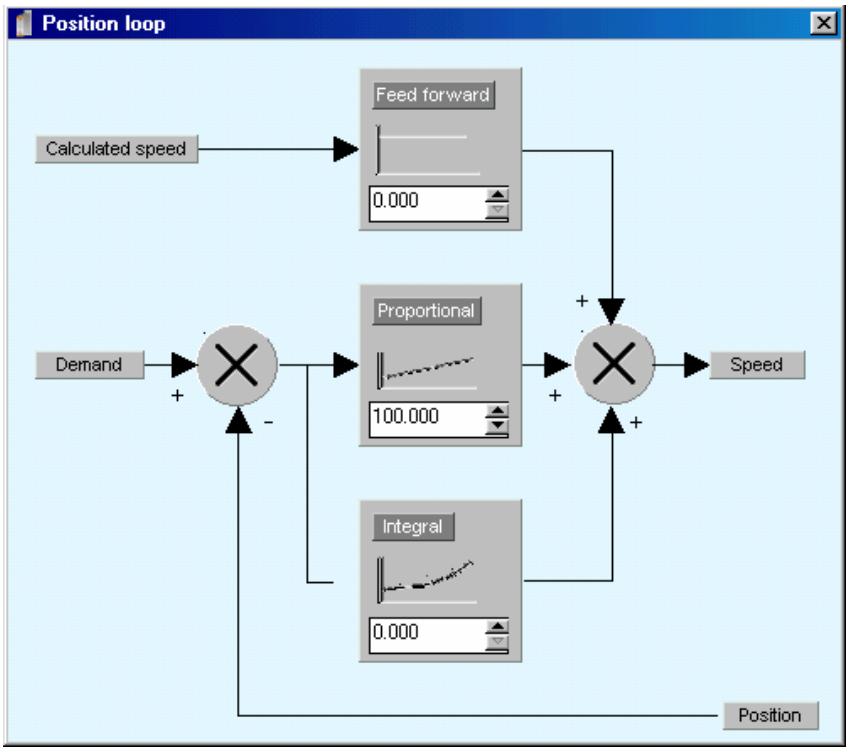
Typical curves for optimised gains.



- Save the adjustments using **Parameters/Save parameters**.

4-5-4- Position loop adjustment

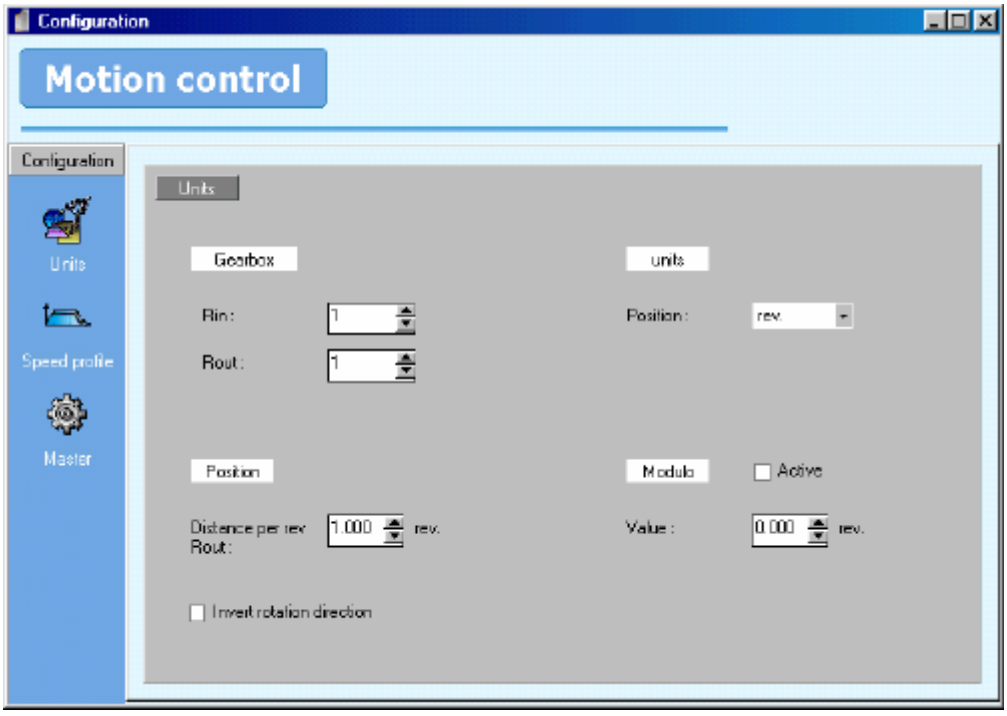
- Disable the drive (*Enable* button OFF in the main window).
- Select **position** mode in the main window.
- Select the menu **Parameters / Position loop**

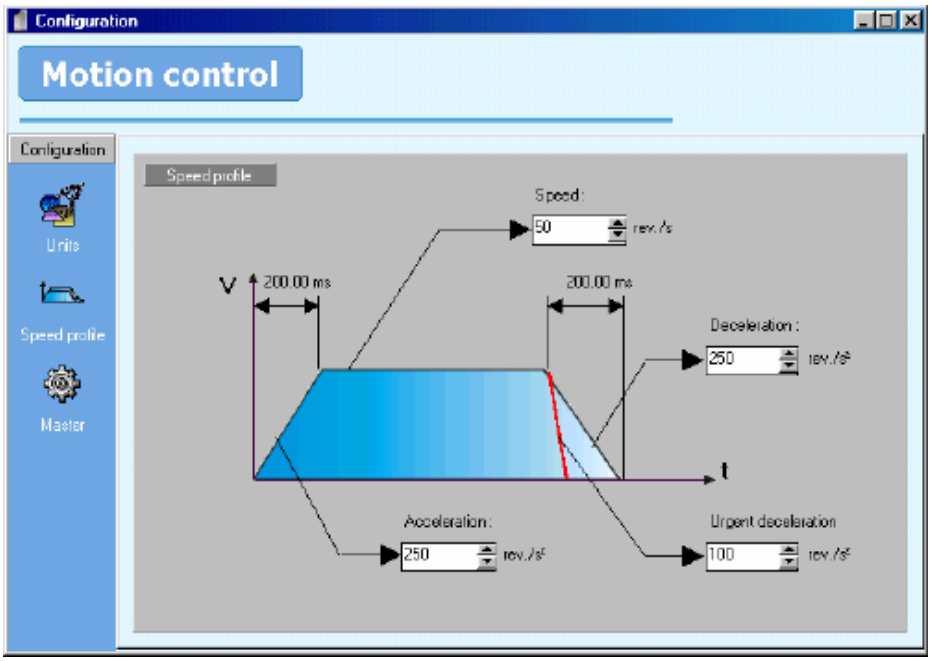


To start the position loop adjustments use the values shown above.

- In **Motion control** / **Configuration**, modify the units and the speed profile as required.

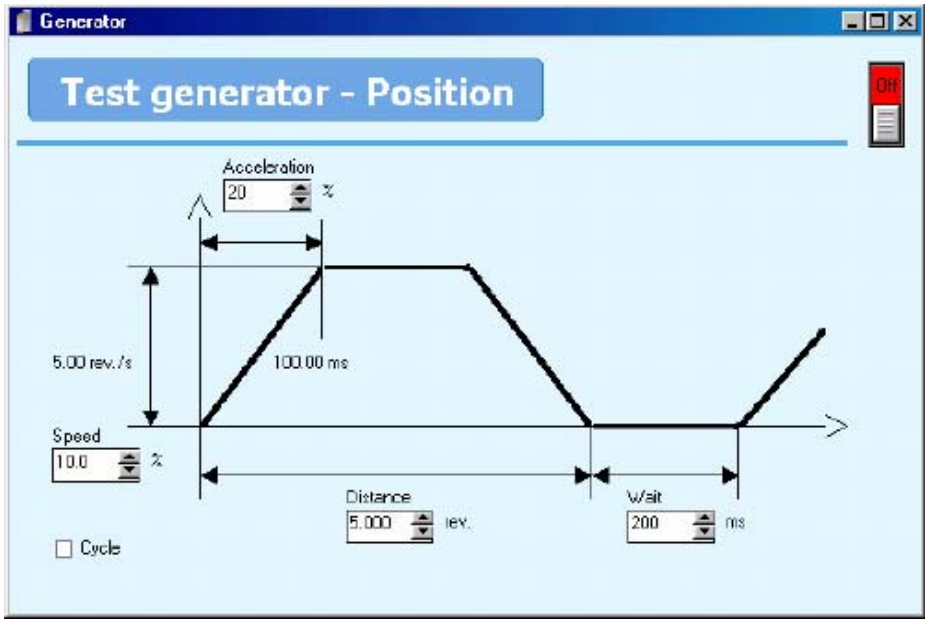
  The percentage speed and acceleration used in the generator window are referenced to the values in the menu **Motion control** / **Configuration** / **Speed profile.**



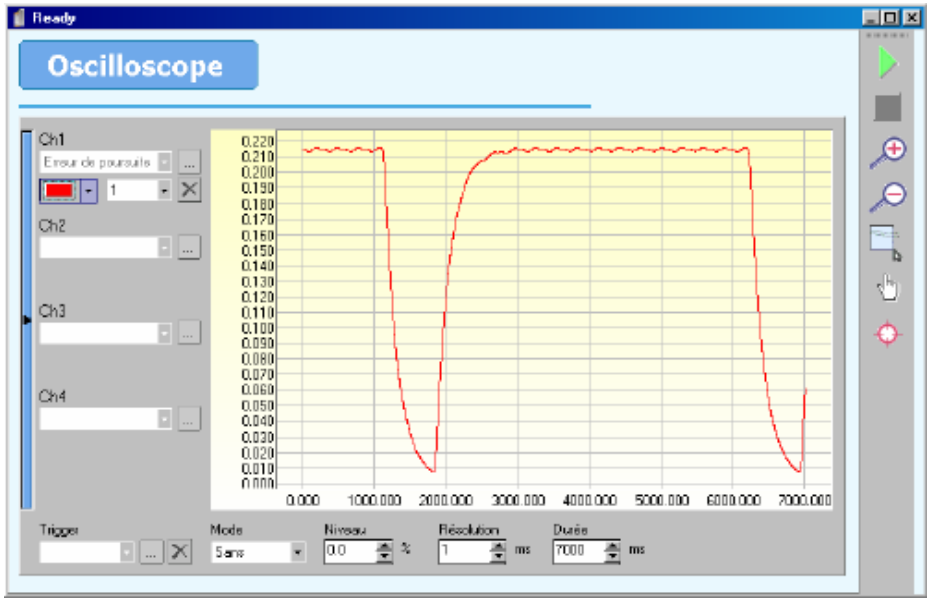


According to the characteristics of the motor, set the following error in **Parameters** / **Supervision** / **Position** / **Following error**

- In **Diagnostic tools / Generator**, start a movement as shown below :



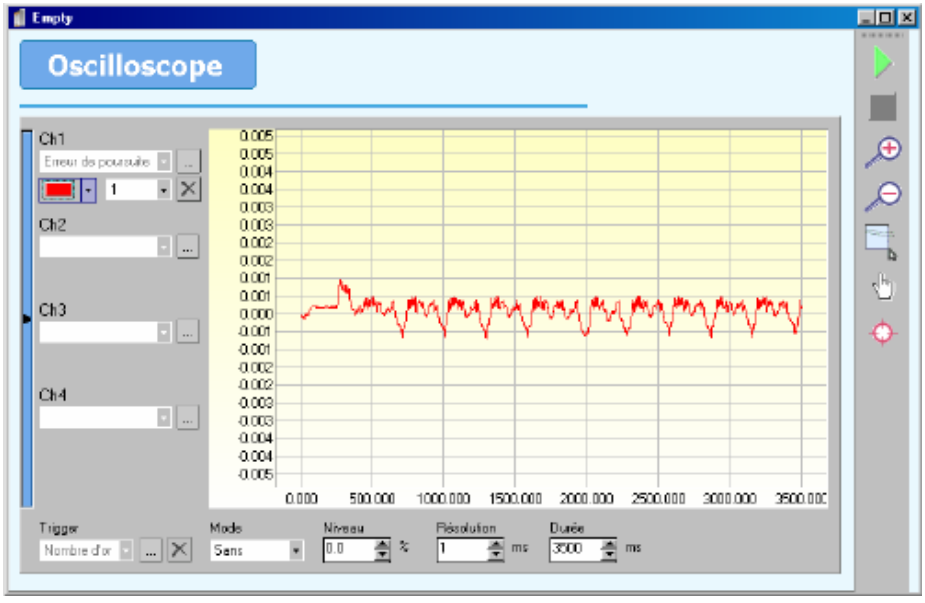
- Use **Diagnostic tools / Oscilloscope** to observe the following error during the movement :



- Select **Following error** in **Position loop** for channel 1.
- Do not select a trigger function.
- Increase the **proportional gain** until the system becomes unstable then reduce the gain by 20%.
- Increase the **feed forward** to reduce the following error to zero.

Typical values : proportional gain 1000 to 3000, feed forward 60 to 65.

Typical curves for optimised gains.



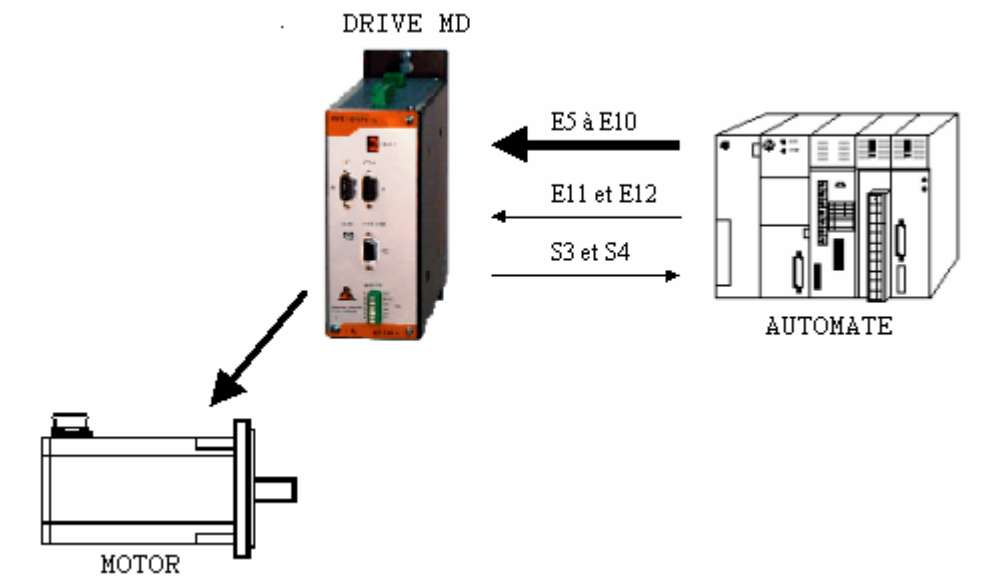
Note : It is useful to observe the theoretical speed on channel 2 in order to know the following error during the acceleration and deceleration phases. In this case adjust channel 1 by a factor of 1000 and channel 2 by a factor of 0.001

- Save the adjustments using **Parameters/Save parameters**.

5- Trajectories

5-1- Introduction :

The trajectory mode allows a PLC or an external controller to start one of up to 64 pre-stored movements using the digital inputs to select a particular one.

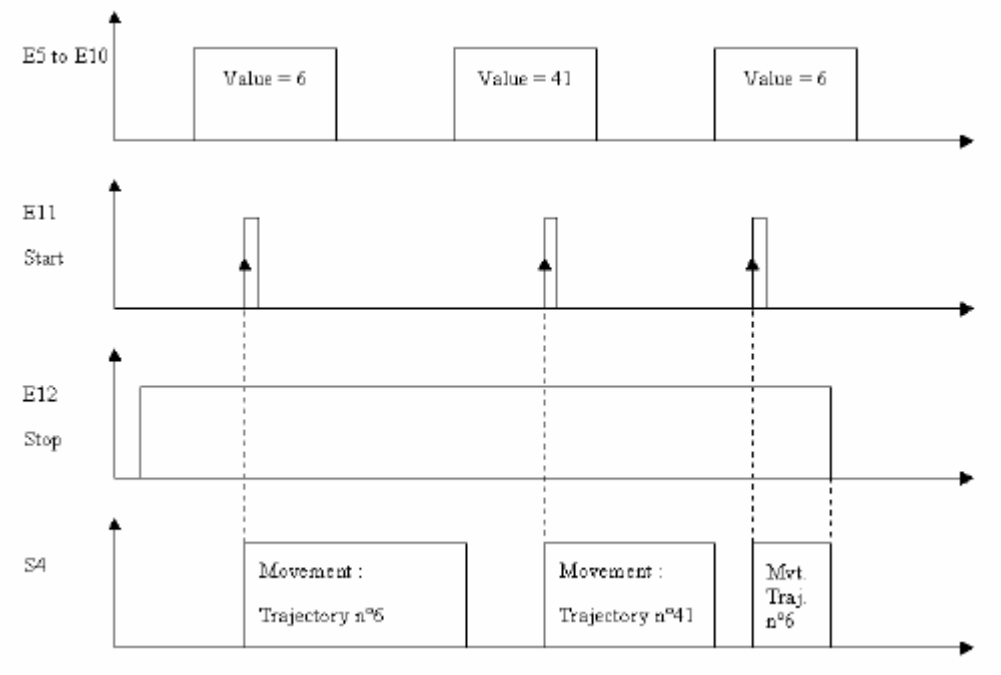


Each trajectory profile is defined by a speed, acceleration and deceleration. All of these parameters are stored in the first 64 real and long-integer variables.

If DPL is used at the same time as the trajectories any modification of VR0 to VR63 or VL0 to VL63 by the tasks will also modify the corresponding trajectory.

5-2- Operation :

5-2-1- Timing:



5-2-2- I/O expansion card :

- Inputs 5 to 10 : used to code the trajectory number. Input 5 is the LSB.
- Input 11 : START the trajectory on the rising edge of this input.
- Input 12 : STOP. A logic 1 allows operation. A logic 0 stops the movement.
- Output 3 : Homing state. 0 if homing not done, 1 if homing completed.
- Output 4 : Movement status (MOVE_S) : 0 if axis stopped, 1 if axis moving.

Note : Input 5 corresponds to the first input on the I/O expansion module.

5-2-3- Composition of a trajectory :

Each trajectory is coded using a real number and a long-integer.

e.g. : The trajectory TRJ0 is coded using VR0 et VL0

The trajectory TRJ19 is coded using VR19 et VL19

- The real variable contains the position.
- The long integer is divided into 4 bytes :

  1st byte : Mode (MS byte)

  2nd byte : Speed

  3rd byte : Acceleration

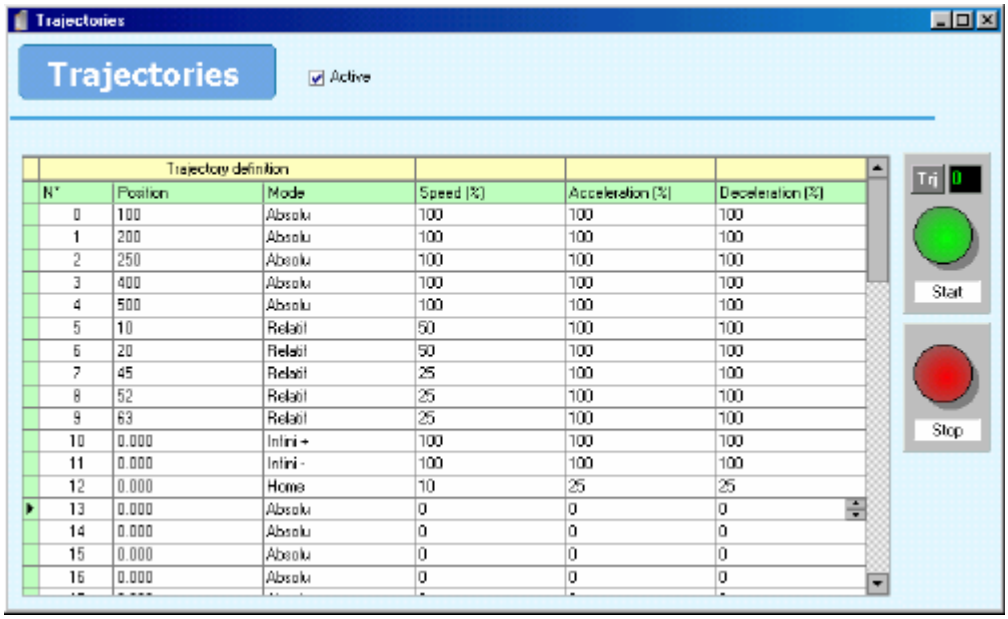
  4th byte : Deceleration (LS byte)

5-3- Implementation:

• Define trajectories :

To use the trajectories the drive must be in position mode.

- Select **Trajectories** in the menu **Motion Control** .
- If the drive is connected to a PC, the PC will search for any trajectories contained in the drive and display them. Otherwise the user will be asked to open a trajectory file or create a new one.



- Tick Active to activate the trajectories.
- For each trajectory you must enter :

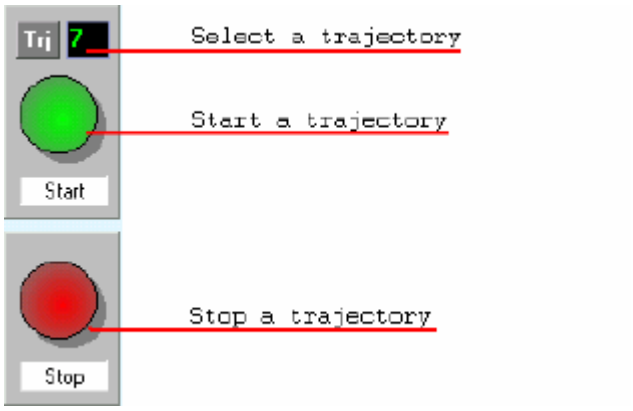
1. A position
2. A mode : absolute, relative, infinite +, infinite – , or home
3. A speed in %
4. An acceleration in %
5. A deceleration in %

All of the values entered relate to the **units** and **speed profile** entered in **Motion Control** / **Configuration.**

- Save the trajectories with **Communication** / **Trajectories** / **Save trajectories**.

• Simulate trajectories :

In the screen **Define trajectories,** you can simulate the trajectories entered :



1. Verify that the drive is enabled and that the 'Active' box is selected.
2. Select the number of the trajectory to execute.
3. Press START to launch the trajectory.
4. Press STOP to stop the movement before the end.

The input STOP (Input 12) must be at a logic 1 to perform a trajectory.

- **TRJ files :**
  - It is possible to save the trajectories in a file .trj with **Communication** / **Trajectories** / **Receive trajectories.**
  - In the same way, it is possible to transfer the contents of a .trj file to the drive using **Communication** / **Trajectories** / **Send trajectories.**

6- Programming language

6-1- Introduction

6-1-1- Introduction

- The language DPL (Drive Programming Language) is a programming tool that is both powerful and simple to use. It provides a structured architecture found in other high level languages. DPL comprises a real-time, multi-tasking kernel using pseudo-basic instructions supplemented by specific instruction for automation and motion control.

- DPL supports various data variable formats.

- A project developed using DPL can contain up to 4 tasks running in parallel, each task being assigned its own priority level.

6-1-2- Memory map

FLASH memory

Reserved area to system : ⇨ Boot ⇨ Operating system (Firmware)
256 system parameters : ⇨ 512 bytes
Initialised variable values : => 512 bytes ⇨ VR0 to VR63 ⇨ VL0 to VL63
BASIC programming : ⇨ 7 Kbytes 4 motion-basic tasks

RAM memory

Reserved area to system : ⇨ Boot ⇨ Operating system
256 real variables : ⇨ 1Kbytes ⇨ VR0 to VR255
256 signed dword variables : ⇨ 1 Kbytes ⇨ VL0 to VL255
256 word variables : ⇨ 512 bytes ⇨ VI0 to VI255
256 byte variables : ⇨ 256 bytes ⇨ VB0 to VB255
256 bit variables : ⇨ 32 bytes ⇨ VF0 to VF255

6-2- Variables

6-2-1- Variables

All variables are global and can be used by several tasks.

Variables can also be handled as arrays (using index notion).

You can allot a name to a variable in order to facilitate the reading of your program while passing by **Project / Language DPL / Declaration**.

Ex: Position = POS_S

Variables are numbered from 0 to 255.

Summary of the different variable types:

Type	Value	Memory occupation	Exemple
Bit	1/0, On/Off ou True/False	1 bit of a word	VF0 to VF255
Byte	0 to 255	1 byte	VB0 to VB255
Integer	0 to 65535	2 bytes	VI0 to VI255
Long	+/- 2 147 483 647	4 signed bytes	VL0 to VL255
Real	+/- 2 147 483 647	4 signed bytes	VR0 to VR255

It is possible to use indexed variables in the form of a table.

```
VL22 = VL0[7]      'is equivalent to VL22 = VL7
VL23 = VL2[9]      'is equivalent to VL23 = VL11
VB3 = 9
VL24 = VL5[VB3]  'is equivalent to VL24 = VL14
```

Real variables are signed long-integers multiplied by a coefficient type 1, 0.1, 0.01 … (fixed point)

To change the coefficient enter menu **Option** -> **Language DPL** -> **Compiler** or **Motion control -> Configuration -> Units ->Precision**

6-2-2- Conversion between data types

To convert one data type to another, simply make an assignment :

- Flag :

```
VB1 = VF0
VI1 = VF0
VL1 = VF0
VR1 = VF0
```

- Byte

```
VF2 = VB0          ' VF2 is equal to the LSB of VB0
VI2 = VB0
VL2 = VB0
VR2 = Vb0
```

- Integer

```
VF3 = VI0          ' VF3 is equal to the LSB of VI0
VB3 = VI0          ' VB3 is equal to the LS Byte of VI0
VL3 = VI0
VR3 = VI0
```

- Long-integer

```
VF4 = VL0       ' VF4 is equal to the LSB of VL0
VB4 = VL0       ' VB4 is equal to the LS Byte of VI0
VI4 = VL0       ' VI4 is equal to the 16 LSBs of VL0
VR4 = VL0
```

- Real

```
VF5 = VR0       ' VF5 is equal to the LSB of the integer part of VR0
VB5 = VR0       ' VB5 is equal to the LS Byte of the integer part of VR0
VI5 = VR0       ' VI5 is equal to the 16 LSBs  of the integer part of VR0
VL5 = VR0       ' VL5 is equal to the integer part of VR0
```

6-2-3- Numerical notation

Values can be given in decimal, hexadecimal and binaries.

```
E.g. :  VB0=254                 ' decimal notation
        VB1=0FEh                        ' hexadecimal notation
        VB2=11111110b           ' binaries notation
```

6-3- Tasks

6-3-1- Multi-tasking principles

The real-time, multi-tasking kernel can manage up to 4 tasks in parallel :

The multi-task passes from the current task to the next task if :

- The time spent in the task exceeds the *ageing time*. This time is a parameter set in menu Options / Language DPL / Compiler. It is necessary to recompile the tasks after a modification.
- A blocking instruction is encountered :
  - ⇨ Wait, Delay
  - ⇨ Mova, Movr, Stop, Home
- The instruction NEXTTASK is executed.

As a general rule, a short task allows events to be treated more rapidly than a long task.

6-3-2- Task management

Each task has a starting mode defined when it is created :

- Automatic : the task is launched automatically at power-on of the drive.
- Manual : the task must be launched manually from within a program.

A project must contain at least one automatic task. It is recommended that there is a single task with all of the initialisation routines after which the other tasks can be launched.

There are 5 instructions to manage the tasks :

- Run : Launch a task that is stopped..
- Suspend : Suspend (pause) the execution of a task.
- Continue : Continue the execution of a suspended task.
- Halt : Stop the execution of a task.
- Status : Indicate the state of a task.

Example :

Task 1	Task 2
Prog	Prog

```
.....                          .....
Run 2                          If VR1 = 0  Halt 2
Wait Status(2)=0               .....
....                           End Prog
End Prog
```

Caution : The stopping or suspension of a task does not affect any movements initiated by that task.

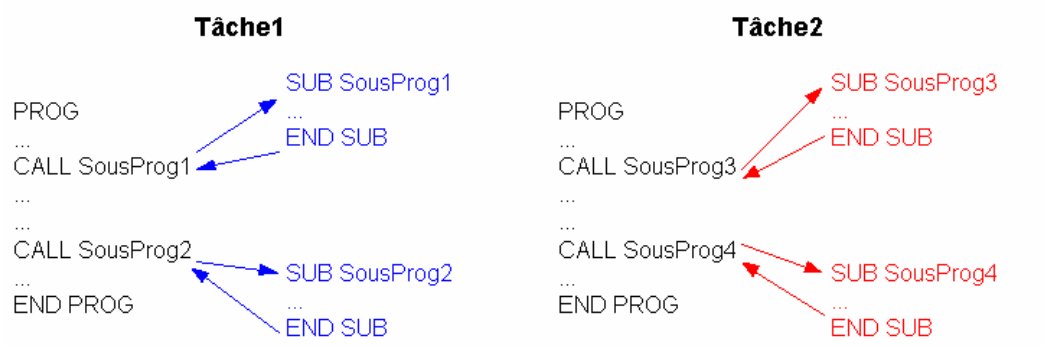
Example :

```
Task 1                         Task 2
Prog                           Prog
.....                          .....
If VF=0 Goto CYCLE_PROD        Mova(1000)
        Halt 2                 Out(6)=1
        Stop                   Mova(2000)
CYCLE_PROD                     .....
....                           End Prog
End Prog
```

6-3-3- Basic task structure

Each task is composed of a main program defined by the keywords PROG and END PROG and by subroutines defined by the keywords SUB .. END SUB.

For example :



• Main program

The main program of a task can call all of its subroutines but it cannot call the subroutines of other tasks. A task corresponds to a file. In the previous example, Task 1 can call Subroutines 1 and 2 but not subroutines 3 and 4. A subroutine can call another subroutine in the same task.

Only one PROG ... END PROG structure can be used in each task and this can be positioned anywhere within the program.

During the execution of a task, the execution of the instruction END PROG causes a branch to PROG.

• Subroutines

A subroutine must be declared using SUB...END SUB. It can be placed either before or after the main program.

To call a subroutine you must use the instruction CALL. The subroutine called must be in the same task.

After a subroutine call the execution continues automatically with the instruction following the CALL instruction. The system leaves the subroutine when it encounters the instruction END SUB or EXIT SUB. For example :

```
SUB Calculate
VR2=0
  IF VR1<>=0 GOTO DIV_OK        ' If VR1 is zero the division is impossible
  EXIT SUB
DIV_OK:
  VR2=VR10/VR1 ' Division
END SUB
```

A subroutine can be called from anywhere within the program but it cannot call itself. If data are used in both the program and subroutine it is recommended that the data be carefully specified. In fact, all variables can be modified by a subroutine. You could use specific variables for each subroutine, setting their values just before the call.

For example :

```
...
VR100=VR1
VR101=VR18
CALL Divide
IF VR102>10  Goto ...
...

SUB Divide
VR102=0
IF VR100=0 EXIT SUB
VR102=VR100/VR101
END SUB
```

• Branch to a label

The GOTO instruction causes a branch to a label. A label is composed of a name ending in ":". If the GOTO instruction is used within a subroutine, the label must be in the same subroutine SUB...END SUB structure.

A branch using the GOTO instruction can be directed either forwards or backwards in the program. For example :

```
GOTO Label1
...
Label1:
...
```

• Operators

Expressions are made up of operators and operands. In Basic, nearly all operators are binary, meaning that they use two operands. Operators using only one operand are called unary operators. Binary operators use common algebraic forms e.g. A + B. Unary operators are always placed before the operand e.g. NOT A. In complex expressions, priority rules govern operator order.

Operator	Priority	Type
NOT	First (High)	Unary
*, /, DIV, MOD, ,AND , <<, >>	Second	Multiplication
+, -, OR, XOR	Third	Addition
=, <>, <, >, <=, >=	Fourth (Low)	Comparison

 In one program line, a single operator can be treated

• Arithmetic operators

The operator 'NOT' is a unary operator. The operators + and – are used as both unary and binary operators; the remainder are only binary.

A unary operator has only one parameter.

For example : NOT <Expression>

A binary operator requires two parameters.

For example : <Expression1> * <Expression2>

• Binary operators :

Operator	Operation	Operand type	Type
+	Addition	Byte, Integer, Long integer or real	Operand type
-	Substraction	Byte, Integer, Long integer or real	Operand type
*	Multiplication	Byte, Integer, Long integer or real	Operand type
/	Division	Byte, Integer, Long integer or real	Real
DIV	Integer division	Byte, Integer, Long integer or real	Operand type
MOD	Modulus	Byte, Integer, Long integer or real	Operand type

• Unary operators :

Opérateur	Opération	Type de l'opérande	Type
+	Même signe	Octet, Entier, Entier long ou réel	Type de l'opérande
-	Inversion de signe	Octet, Entier, Entier long ou réel	Type de l'opérande

• Logic operators :

Opérateur	Opération	Type de l'opérande	Type
NOT	Négation binaire	Octet, Entier	Type de l'opérande
AND	ET logique	Octet, Entier	Type de l'opérande
OR	OU logique	Octet, Entier	Type de l'opérande
XOR	OU exclusif	Octet, Entier	Type de l'opérande
>>	Décalage à droite	Octet, Entier	Type de l'opérande
<<	Décalage à gauche	Octet, Entier	Type de l'opérande

• Bit operators :

Operator	Operation	Operand type	Result type
+	Same sign	Byte, Integer, Long integer or real	Operand type
-	Invert sign	Byte, Integer, Long integer or real	Operand type

- **Relationship operators :**

Opérateur	Opération	Type de l'opérande	Type
=	Egal	Octet, Entier, Entier long, Réel ou chaîne de caractères	Bit
<>	Différent de	Octet, Entier, Entier long, Réel ou chaîne de caractères	Bit
<	Inférieur à	Octet, Entier, Entier long, Réel ou chaîne de caractères	Bit
>	Supérieur à	Octet, Entier, Entier long, Réel ou chaîne de caractères	Bit
<=	Inférieur ou égal à	Octet, Entier, Entier long, Réel ou chaîne de caractères	Bit
>=	Supérieur ou égal à	Octet, Entier, Entier long, Réel ou chaîne de caractères	Bit

- **Tests**

Conditional instructions are a useful means of executing, or not, a group of instructions according to a condition being true or false :

```
IF <Expression> GOTO <Label>
…
Label:
…
```

<Expression> must have a bit type value. If <Expression> is true, the jump to <Label> is executed. If <Expression> is false, le program moves directly to the following line.

Example :

```
VEL%=100                          ' Rapid speed
STTA=2000                         ' Move to absolute position 2000
MOVE_ON:
IF POS_S <1000 GOTO NEXT_VEL      'If the position is greater or equal to 1000 then
VEL%=50%                          ' Speed is reduced to a half.
NEXT_VEL:
F POS_S<1500 GOTO NEXT_OUT        'If the position is greater or equal to 1500 then
OUT(9)=1                          'Set output 9.
NEXT_OUT:
IF MOVE_S<>1 GOTO MOVE_ON         'Loop until the movement is finished.
…
```

7- Motion control programming

7-1- Introduction

The drive can control a servo axis and a master encoder.

The DPL software contains numerous instructions associated with motion control :

positioning, electronic gearbox, superposition etc.

The position counter can count up to ± 2048 motor revs.

The sense of the position control loop can be inverted in the parameter list : Motion control / Invert motor sense (Caution, this does not reverse the rotor position shown on the instrument panel).

7-2- Open loop / Closed loop

7-2-1- Open loop operation

The axis switches out of the controlled mode (open loop) :

↳ Each time the drive is restarted.

↳ Each time the instruction AXIS OFF is executed in a task.

↳ On detecting a following error (unless the instruction SECURITY has been executed).

↳ By using the debug menu (*enable* button OFF), or the communication menu (stop tasks, send tasks, restart the drive).

The instruction AXIS_S allows the state of the axis to be read.

If a movement instruction is executed whilst in open loop, the instruction will appear to have been executed but no motion will take place.

For example :

Task Process

```
PROG
...
...             ' the drive has detected a following error
...             ' => the axis goes open loop
MOVA=1000       ' the instruction is consumed but not acted on
OUT(3)=1        ' Output 3 is activated
MOVA=2000       ' the instruction is consumed but not acted on
OUT(3)=0        ' Output 3 is deactivated
...             ' Output 3 would only be on transiently since
...             ' the instruction Mova(2000)took very little system time
END PROG
```

7-2-2- Closed loop operation

In order that the servo axis can control movements, it is necessary to switch to closed loop control.

The axis is in controlled mode (closed loop) :

↳ Each time the instruction AXIS ON is executed by a task.

↳ By using the debug menu (*enable* button ON).

The instruction AXIS_S allows the state of the axis to be read.

 The AXIS instruction takes approximately 3ms to become effective. To ensure that the axis is in closed loop mode use :

Axis On
Wait AXIS_S=On

7-3- Positioning

7-3-1- Absolute movements

- **Start a movement : STTA**

To initiate a movement towards an absolute position and not to wait for the movement to be completed before continuing with the task, we must use STTA. This instruction is very useful if the speed or the target position must be changed during the course of the movement. With this function the absolute error is minimal.

This instruction does not block the task (unless the movement buffer is full).

It uses the current values for acceleration, deceleration, and speed. The syntax is :

STTA=Position

For example :

```
VEL%=100
STTA=2000             ' Start moving towards absolute position 2000
WAIT POS_S >200       ' Wait for position 200
OUT (6)=1             ' Set an output
WAIT POS_S >700       ' Wait for position 700
OUT (6)=0             ' Clear an output
WAIT MOVE_S=0         ' Wait for the end of the movement
```

In this example, during the movement we can change the outputs since the task is not blocked.

If the instruction MERGE is active and several STTA instructions are loaded, the movements will be executed one after the other without passing through zero speed.

If the axis is declared as modulo, the motion towards a position will be in a positive sense if the demanded value is positive, and a negative sense if the demanded value is negative. For example :

Axis modulo 360°

Axis at an initial position of 90°

```
STTA=-10   'movement in a negative sense for a distance of 80°
WAIT MOVE_S=0
STTA=350   'movement in a positive sense for a distance of 340°
WAIT MOVE_S=0
STTA=30   'movement in a positive sense for a distance of 30°
WAIT MOVE_S=0
```

- **Movement : MOVA**

The instruction MOVA sends the axis to an absolute position. It uses the current values for acceleration, deceleration, and speed. The syntax is :

MOVA=Position

This instruction sends the axis to an absolute position having the value <Position>. The program waits for the end of the movement before continuing. The positioning error is minimal.

For example :

```
MOVA=100
CALL Punch
MOVA=0
```

The instruction MOVA blocks the task until the movement is finished (condition MOVE_S=0).

MOVA=100 is equivalent to STTA=100

WAIT MOVE_S=0

- **Trajectory : TRAJA**

The Trajectory function is designed to simplify the definition of complex movements.

It allows a movement to be launched towards an absolute position with a specific speed.

Syntax :

TRAJA (<Position>, <Speed>)

For example :

TRAJA (500,2000) is equivalent to :

```
VEL=500
STTA = 2000
```

If the MERGE instruction is active and several TRAJA or TRAJR instructions are loaded, the movements will be executed one after the other without passing through zero speed. For example :

MERGE On

TRAJA(500,2000)

TRAJA(1000,50) ‘change to low speed at position 500

7-3-2- Relative movements

- **Start a movement : STTR**

To initiate a movement towards a relative position and not to wait for the movement to be completed before continuing with the task, we must use STTR. This instruction is very useful if the speed or the target position must be changed during the course of the movement

This instruction does not block the task (unless the movement buffer is full).

It uses the current values for acceleration, deceleration, and speed. The syntax is :

STTR=Position

For example :

```
VEL%=100                        ' Rapid speed
VR1=POS_S
STTR=2000                       ' Start moving to a relative position 2000
LOOP  :
VR2 = POS_S
VR2 = VR2 - VR1
IF VR2 < 100 GOTO LOOP          ' Wait for position +100
VEL%=10                         ' Slow speed
WAIT MOVE_S=0                   ' Wait for the end of the movement
```

In this example, during the movement the speed can be modified since the instruction does not block the task.

If the MERGE instruction is active and several STTR instructions are loaded, the movements will be executed one after the other without passing through zero speed.

• Movement : MOVR

The instruction MOVA sends the axis to an relative position. It uses the current values for acceleration, deceleration, and speed. The syntax is :

MOVR=Distance

This instruction sends the axis to an relative position having the value <Position>. The program waits for the end of the movement before continuing

For example :

```
VB1=0
LOOP:
  MOVR=100
  CALL PUNCH
  VB1=VB1+1
  IF VB1<10 Goto LOOP
```

The instruction MOVA blocks the task until the movement is finished (condition MOVE_S=0).

MOVR=100 is equivalent to

```
STTR=100
WAIT MOVE_S=0
```

• Trajectory : TRAJR

The Trajectory function is designed to simplify the definition of complex movements.

It allows a movement to be launched towards an relative position with a specific speed.

Syntax :

TRAJR (<Position>, <Speed>)

For example :

TRAJR (500,2000) is equivalent to :

```
VEL=2000
STTR=500
```

If the MERGE instruction is active and several TRAJA or TRAJR instructions are loaded, the movements will be executed one after the other without passing through zero speed. For example :

MERGE On

TRAJR(500,2000)

TRAJR(1000,50) ‘change to low speed at position500

7-3-3- Infinite movements

To start a continuous movement you must use the instruction STTI.. The axis moves at the current speed.

This instruction does not block the task (unless the movement buffer is full).

The instruction STOP or SSTOP is required to stop a continuous movement. The direction of the movement is defined by "+" or "-"

Syntax :

STTI (Sign)

Example :

```
WAIT  INP(4)=On
STTI(+)
WAIT  INP(4)=Off
STOP
```

7-3-4- Stopping a movement

To stop a movement you must use either STOP or SSTOP. The axis is stopped using the programmed deceleration and the movement buffer is cleared.

The instruction STOP blocks the task until the movement is finished (condition MOVE_S=0) whereas SSTOP is non-blocking.

Syntax : STOP

Example : move until a sensor is activated.

```
STTI(+)
WAIT  INP(4)=On
STOP
```

The instruction AXIS OFF also stops the movement but without any control as the drive is inhibited.

7-4- Synchronization

7-4-1- Electronic gearbox :

- **GEARBOX :**

This instruction implements an electronic gearbox between a master encoder and the motor (slave axis).

Syntax :

GEARBOX(<Numerator>, <Denominator>)

<Numerator> / < Denominator > define the ration between one rev of the slave and one motor rev of the encoder. In fact, for [<Master coder resolution*<Denominator >] increments, the motor will move of [4096*<Numerator >] increments, knowing that we have 4096 increments per rev.

This instruction does not block the task (unless the movement buffer is full). As long as the link between the master and the slave is not broken the instruction MOVE_S(slave) will return a value of 1 (even if the slave axis is stopped).

 The instruction GEARBOX internally sets the value of GEARBOXRATIO to 4096.

Example: If Numerator = 1 and Denominator = 2, for 1 rev of the master encoder the slave motor moves by 0.5 revs.

The Numerator must be lower or equal to 8 and integer type.

The Denominator must be an integer and (Denominateur *Master coder resolution) < = 32768 must be true.

Example: Master coder 4000 increments - > Denominator must be lower than 8.

Gearbox with valous < Numerator > or < Denominator > different from 1, affects the scale of the position of the main coder (if you use master postion or Cambox).

- **STARTGEARBOX :**

This instruction initiates an electronic gearbox using an acceleration and a ratio previously defined by GEARBOX. The ratio between master and slave is : (ratio × <Numerator>) / (<Denominator> × 4096), with <Numerator> and <Denominator> defined in the instruction GEARBOX.

Syntax : STARTGEARBOX(<Acceleration>)

< Acceleration > 0 to 65535

The acceleration phase is : (Ratio × 640µs) / Acceleration

With Ratio corresponding to the value of GEARBOXRATIO.

- **GEARBOXRATIO :**

This instruction modifies the reduction ration of an electronic gearbox (the instruction STARTGEARBOX having already been executed).

Syntax : GEARBOXRATIO(<Ratio>)

<Ratio> 0 to 32767 : the ratio of the gearbox is defined by (<Ratio> × <Numerator>) / (<Denominator> × 4096). <Numerator> and <Denominator> are parameters of the instruction GEARBOX.. In fact, for [<Master coder resolution*<Denominator >] increments, the motor will move of [Ratio*<Numerator >] increments, knowing that we have 4096 increments per rev.

The instruction is non-blocking and allows the ratio to be changed without stopping the gearbox.

GEARBOXRATION don't affect the position scale of the master coder.

 The instruction GEARBOX internally sets the value of GEARBOXRATIO to 4096.

- **STOPGEARBOX :**

This instruction stops an electronic gearbox using the deceleration defined in the instruction.

Syntax : STOPGEARBOX(<Deceleration>)

<Deceleration> 0 to 65535

The deceleration phase is : (Ratio × 640µs) / Deceleration

With Ratio corresponding to the value of GEARBOXRATIO.

 **For instruction STOPGEARBOX, it is necessary to recopy the real position in the theoretical position because the latter did not evolve/move any more because of the electricgearbox.**

- **Example :**

```
GEARBOX (1, 2)              'The motor turns twice as fast as the master encoder
GEARBOXRATIO (4096)
…
STARTGEARBOX(4)             'Initiate a gearbox with an acceleration phase
…                           'of  (4096 × 640 / 4) = 655360µs  i.e. 0.65s
GEARBOXRATIO(3687)          'Ratio  : (3687 × 1) / (2 × 4096) = 0.45
…
STOPGEARBOX(2)              'Stop the gearbox with a deceleration phase
                            'of (3687 × 640 / 2)=1180000µs  i.e. 1.18s
WAIT MOVE_S=0
VR0=POS_S
HOME (0,VR0)                'update theoretical position
```

7-5- Capture

7-5-1- Capture :

Capture allows for the registration of the current axis position on the rising edge of an input signal to the drive. The capture is done in less than 640 µs.

- **CAPTURE1 or CAPTURE2 :**

The instructions CAPTURE1 and CAPTURE 2 are used to record the current position of the axis.

Syntax : CAPTURE1 (<Source>, <InputNo>, < Window >, <Min>, <Max>, <Interior>)

With this instruction the drive waits for the rising edge of a capture input signal. When the edge is detected, the position is stored in variable REGPOS1_S. The flag REG1_S is set as true.

<Source> 0 for motor position, 1 for master encoder.

< InputNo > the input no of the capture signal (1 to 16).

< Window > if true then the input is only tested when the axis is between the positions <Min> and <Max>.

<Interior> defines whether the test is performed inside or outside the limits <Min> and <Max>

<Min> must always be less than <Max>.

- **REG1_S or REG2_S :**

Syntax : <VFx>=REG1_S

Description : This function indicates if a position capture has been carried out.

Remarks : The returned value is only true once per capture. REG1_S is automatically reset to zero by a read operation. On starting a new capture operation, if REG1_S is currently 1 it is set to 0.

- **REGPOS1_S or REGPOS2_S :**

Syntax : <Variable>=REGPOS1_S

Data types : Variable : real

Description : This function returns the last captured position of the axis obtained using the instruction CAPTURE1.

- **Example :**

```
CAPTURE1(0,4,On,10,20,On)   'Capture position on rising edge of input 4,
```

```
                                        ' when the motor axis is between 10 and 20
WAIT REG1_S = ON            'Wait for a capture
VR1 = REGPOS1_S             'VR1 = value of the captured position
```

8- PLC programming

8-1- Digital I/O

8-1-1- Read inputs

The function INP is used to read 1 bit, INPB a block of 8 bits and INPW a block of 16 bits.

The syntaxes are : INP(<InputNumber>), INPB(<BlockNumber>), INPW

<InputNumber> must represent the number of an input <BlockNumber> the number of a block of 8 inputs. This number corresponds to the number in the configuration module. The types of data returned are:

- Bit for an input
- Byte for a block of 8 inputs
- Integer for a block of 16 inputs

For example:

```
VF1= INP(3)          'read input number 3
VB2 = INPB(1)        'read the first block of 8 inputs
VB4 = INPB(2)        'read the second block of 8 inputs
VI3= INPW            'read 16 inputs
```

8-1-2- Write outputs

The function OUT is used to write 1 bit, OUTB a block of 8 bits.

The syntaxes are : OUT(<OutputNumber>), OUTB(<BlockNumber>).

< OutputNumber >must represent the number of an output, < BlockNumber > the number of a block of 8 outputs. This number corresponds to the number in the configuration module. The types of data used are :

- Bit for an output
- Byte for a block of 8 outputs

For example :

```
OUT(5) = 1           'set output 5 high
OUTB(1) = 48         'write to a block of 8 outputs
```

8-1-3- Read the outputs

All outputs can be read as well as written to. The value read is the last value written. This property is very useful when more than one program uses the same block of outputs. It is possible to write only to the required outputs in one operation without changing the others.

For example :

To set bit 4 in a block of 8 bits :

```
OUTB(2)= 16          'set bit 4 to 1

VB0 = OUTB(2)        'read a block of 8 outputs
```

8-1-4- Wait state

It is possible to wait a change of state on an input using the instruction WAIT.

The syntax is: WAIT <Condition>

The function WAIT is used to wait for a changing state during normal execution. The execution of the task is stopped for as a long as the condition is false. When the condition becomes true, execution continues. This function is very useful to wait for the end of a movement etc.

Example :

```
WAIT INP(2) = ON              'Wait until input 2 is 1

STOP                          'Stop the axis

WAIT INP(5) = ON              'Wait until input 5 is 1
```

8-1-5- Test state

It is possible to test the state of an input using the instruction IF…

The syntax is : IF (<Condition>) GOTO <Label>

The structure IF… is used to test a condition at a given instant. If the <Condition> is true the program execution branches to the label.

Example :

```
IF INP(5) = ON GOTO Label_1        'Test the state of input 5,

                                   ‘If the input is a 1 jump to Label_1
```

8-2- Analogue I/O

8-2-1- Read an input

The functions ADC(1) et ADC(2) are used to read the 2 analogue inputs. The data returned by this instruction are always real and in the range -10 to +10.

For example:

```
VR1 = ADC(1)                    'Read analogue input 1
VR5 = ADC(2)                    'Read analogue input 2
```

8-2-2- Write an output

The function DAC is used to write to the analogue output.

The syntax is : DAC=<Real_expression>

The data used by this instruction are always and in the range -10 to +10.

For example:

```
DAC=5.0                 'Set the output with a value of 5 V
```

8-3- Timers

8-3-1- Passive wait

The function DELAY is used to give a passive wait.

The syntax is : DELAY <Duration>

<Duration> is an integer expressed in milliseconds. This instruction is recommended for long passive waits since during the wait the program does not use any processor time.

With this function the program waits for the duration indicated.

For example:

```
Start:
        WAIT INP(5) = 1
        ...
        DELAY 5000              ' Wait for 5 seconds
        ...
GOTO Start
```

8-3-2- Active wait

- **TIME :**

The internal global variable TIME can be used to give an active wait. TIME is a long-integer that represents the number of 0.640 thousandths of second elapsed since the last power-on. This variable can, therefore, be used as a time base. It is particularly suitable for machines that are powered-up for less than 16 days at a time. This is because at power-on TIME is initialised to 0. After 16days the variable reaches its maximum value of 2^31 and then goes to 2^-31. This transition can, in certain cases, give timing errors. To avoid this problem it is preferable to use the instruction LOADTIMER.

For example :

```
VL2=TIME

VL2=VL2 + 7812

Loop  :

 VL3 = TIME

  IF VL3<VL2 GOTO Loop            ‘5 second delay
```

Note : TIME is a long-integer

Warning : TIME don’t work in a test.

- **LOADTIMER and TIMER :**

The instruction LOADTIMER can be used to give an active wait. This is a real variable that represents the number of 0.640 thousandths of second elapsed since the last power-on. This variable can, therefore, be used as a time base. It is particularly suitable for machines that are permanently powered-up.

It also allows the loading of a value into a timer which decrements automatically down to 0. We can tell if the timer has timed-out using the instruction TIMER(VLXX), with XX between 0 and 255.

If TIMER(VLXX) = 1 the time has not elapsed.

If TIMER(VLXX) = 1 the timer has timed-out.

It is possible to use 256 timers simultaneously.

For example :

```
LOADTIMER(VL129)=4688                    ‘Load a delay of 3s

Loop:

        IF TIMER(VL129)<>0 GOTO Loop     ‘Wait for the end of the delay
```

Note : During the execution of these lines the long-integer variable VL129 is used by the system.

8-4- Counters

Caution :

- The same input cannot be used both as a counter and for position capture.

- When the counter reaches its maximum value it goes to 0 on the next edge (maximum value : 65535).

8-4-1- Configuration :

The instruction SETUPCOUNTER is used to configure the counter.

Syntax : SETUPCOUNTER(<CounterNo>,<Input>,<Filter>)

< CounterNo > : 0 or 1

<Input> : Input number (1 to 16)

<Filter> : Activation of filter : 0 for no filter, 1 for filter.

If the filter is not activated the maximum frequency is 781 Hz otherwise it depends on the filter parameter in **Parameters** / **Digital Inputs Outputs** .

8-4-2- Writing :

The instruction COUNTER(1 or 2) is used to initialise the counter with a value.

Syntax : COUNTER(<CounterNo >) = <Value>

< CounterNo > : Counter number (1 or 2)

<Value> : Value between 0 and 65535

8-4-3- Reading :

The instruction COUNTER_S is used to read the counter.

Syntax : <Variable>=COUNTER_S(<CounterNo >)

<Variable> : Integer between 0 and 65535

< CounterNo >: Counter number (1 or 2)

8-5- Cam boxes

8-5-1- Cam box

Cam boxes allow digital outputs to be controlled according to angular or linear positions.

DPL can have 2 cam boxes with up to 4 segments per box. For example, outputs 3, 4 and 12 can be controlled by a cam box and the other can be used elsewhere.

The outputs of a cam box are updated every 1.3ms.

The functions available are :

CAMBOX, CAMBOXSEG, STARTCAMBOX and STOPCAMBOX

When a segment is declared, the starting value can be greater than the end value. The program zero is taken into account with each definition of segment.

Before declaring a cam boxes you must pass the axis as a slave mode with GEARBOX (1, 1)

8-5-2- Cam boxes

The drive handles up to two cam boxes, each having four segments.

The source can be either the motor position or the position of the master encoder (connector X2).

When the source is the motor position, the values for the start and the end of the segment are directly tied to scaling and units in the screen **Motion control** / **Configuration** / **Units**.

When the source is the master encoder:

- Verify that the increments per rev of the encoder have been correctly entered in the window **Parameters** / **Encoder**: function = encoder input and resolution = 4000 for an encoder with 1000 lines for example.
- Verify that in the case of an infinite axis the menu **Motion control** / **Configuration** / **Master**: modulo = active and value = the scaled units for the slave axis.

Before declaring a cam boxes you must pass the axis as a slave mode with GEARBOX (1, 1)

Gearbox with valous < Numerator > or < Denominator > different from 1, affects the scale of the position of the main coder (if you use master postion or Cambox).

Example : We require a master module equal to 15 revs of the encoder.

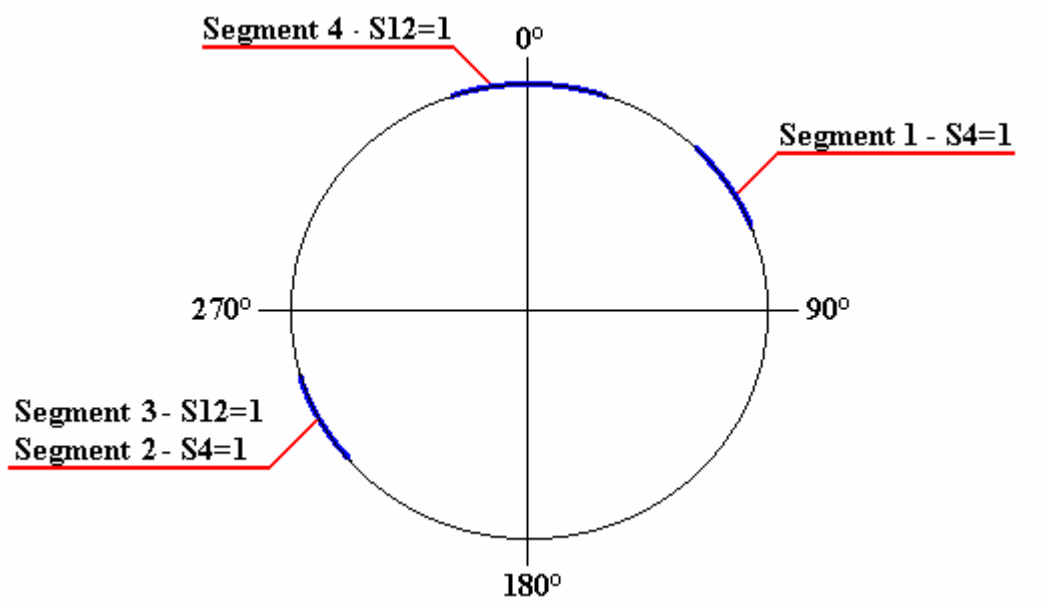
Master encoder : 4000 increments per rev.

Slave motor : Rin = 10, Rout = 1, Distance = 360° (see screen **Motion control** / **Configuration** / **Units**).

Therefore one rev of the slave motor represents 36°.

Internally there is a direct correspondence between 1 rev of the master encoder and 1 rev of the motor, 1 rev of the master encoder = 36°.

In the instruction CAMBOXSEG, the start and end of the segments must be between 0° and 539.9°.



In this example, the master encoder is modulo 360. The cam boxes are written in the following way :

```
GEARBOX (1,1)

GEARBOXRATIO(4096)

CAMBOX (1,1,4)                  'Cam box 1, master encoder, 4 segments

CAMBOXSEG(1,1,4,40,60)          'Cam box 1, segment 1, output 4, between
40° and 60°

CAMBOXSEG(1,2,4,230,250)        'Cam box 1, segment 2, output 4, between
230° and 250°

CAMBOXSEG(1,3,12,230,250)       'Cam box 1, segment 3, output 12
between 200° and 400°

CAMBOXSEG(1,4,12,350,10)        'Cam box 1, segment 4, output 12
between 350° and 10°

STATCAMBOX(1)                   'Start cam box 1

…

STOPCAMBOX (1)                  ' Stop cam box 1
```

9- Alphabetical list

To know the time execution of each instruction, read the DPL TIME INSTRUCTION.XLS file in DATA directory.

9-1- Program

CALL	Call a subroutine
NEXTTASK	Move immediately to the following task
GOTO	Jump to a label
PROG ... END PROG	Main program
SUB ... END SUB	Subroutine
EXIT SUB	Exit a subroutine

9-2- Arithmetic

+	Addition
-	Subtraction
*	Multiplication
/	Division

9-3- Mathematical

FRAC	Fractional part
INT	Integer part
MOD	Modulus

9-4- Logic

<<	Shift left
>>	Shift right
AND	AND operator
NOT	NOT operator

OR	OR operator
XOR	Exclusive OR operator

9-5- Test

<	Less than
<=	Less than or equal
<>	Not equal
=	Equal
>	Greater than
>=	Greater than or equal
IF	Conditional test

9-6- Motion control

- **Axis control :**

ACC	Acceleration
ACC%	Acceleration in percent
AXIS	Axis loop control
AXIS_S	Axis loop state
BUFMOV_S	Number of waiting movements
CLEAR	Zero the axis position
CLEARMASTER	Zero the master position
DEC	Deceleration
DEC%	Deceleration in percent
FE_S	Following error
FEMAX_S	Following error limit
HOME	Move to home position
HOME_S	Home state
LOOP	Virtual mode

MERGE	Merge movements
MOVE_S	Movement state
ORDER	Movement order number
ORDER_S	Current order number
POS	Target position
POS_S	Actual position
VEL	Speed
VEL%	Speed in percent

- **Positioning :**

MOVA	Move absolute
MOVR	Move relative
SSTOP	Stop axis (without waiting for zero speed)
STOP	Stop axis
STTA	Start an absolute movement
STTI	Start an infinite movement
STTR	Start a relative movement

- **Synchronisation :**

GEARBOX	Electronic gearbox
GEARBOXRATIO	Modify the ratio of an electronic gearbox
STARTGEARBOX	Start an electronic gearbox
STOPGEARBOX	Stop an electronic gearbox

- **Capture**

CAPTURE1 and CAPTURE2	Start a position capture
REGPOS1_S and REGPOS2_S	Read a captured position
REG1_S and REG2_S	Capture state

9-7- PLC

- **Digital I/O**

CAMBOX	Cam box
CAMBOXSEG	Cam box segment
INP	Read an input
INPB	Read a block of 8 inputs
INPW	Read a block of 16 inputs
OUT	Write an output
OUTB	Write a block of 8 outputs
STARTCAMBOX	Start a cam box
STOPCAMBOX	Stop a cam box
WAIT	Wait for a condition

- **Analogue I/O**

ADC(1)	Read analogue input 1
ADC(2)	Read analogue input 2
DAC	Write analogue output

- **Timing**

DELAY	Passive wait
LOADTIMER	Load a timer value into a variable
TIME	Time base
TIMER	Compare a variable with TIME

- **Counters**

COUNTER	Initialise a counter value
SETUPCOUNTER	Configure a counter
COUNTER_S	Read the state of a counter

9-8- Task management

CONTINUE	Continue the execution of a task
HALT	Stop a task
RUN	Start a task
SUSPEND	Suspend a task
STATUS	Read task state

9-9- Miscellaneous

DISPLAY	7 segment display
LOADPARAM	Load parameters from Flash
LOADVARIABLE	Load variables from Flash into RAM
RESTART	Restart the drive
SAVEPARAM	Save parameters from RAM into Flash
SAVEVARIABLE	Save variables VR0..VR63, VL0..VL63
SECURITY	Define safety actions
VERSION	Read the Operating System version

9-10- Liste aplhabétique

9-10-1- Addition

Syntax : <Expression1> + <Expression2>

Data types : Byte, Integer, Long-integer, Real

Description : This operator adds two expressions and returns a value of the same type as the operands.

Remarks : <Expression1> and <Expression2> must be valid expressions and must be of the same type.

Example :

```
VL1=10
VL2=5
VL3=VL1+VL2      'Result : VL3=15
```

See also : `-`, `*' and `/'.

9-10-2- Subtraction (-)

Syntax : <Expression1> - <Expression2>

Data types : Byte, Integer, Long-integer, Real

Description : This operator subtracts <Expression2> from <Expression1> and returns a value of the same type as the operands.

Remarks : <Expression1> and <Expression2> must be valid expressions and must be of the same type.

Example :
```
VL1=10
    VL2=5
    VL3=VL1-VL2          'Result : VL3=5
```

See also : '+', `*' and `/'.

9-10-3- Multiplication (*)

Syntax : <Expression1> * <Expression2>

Data types : Byte, Integer, Long-integer, Real

Description : This operator multiplies <Expression1> by <Expression2> and returns a value of the same type as the operands.

Remarks : <Expression1> and <Expression2> must be valid expressions and must be of the same type.

Example :
```
VL1=10
VL2=5
VL3=VL1*VL2        'Result : VL=50
```

See also : '+', '-' and `/'.

9-10-4- Division (/)

Syntax : <Expression1> / <Expression2>

Data types : Byte, Integer, Long-integer, Real

Description : This operator divides <Expression1> by <Expression2>

Remarks : <Expression1> and <Expression2> must be valid expressions and must be of the same type. <Expression2> must not be zero. This operator always returns a real value.

Example : VL1=10

VL2=5

VL3=VL1/VL2 'Result : VL3=2

See also : '+', '-', `*'.

9-10-5- Less than (<)

Syntax : <Expression1> < <Expression2>

Data types : Byte, Integer, Long-integer, Real

Description : This operator tests if <Expression1> is less than <Expression2>.

Remarks : <Expression1> and <Expression2> must be valid expressions and must be of the same type.

Example : VL1=10

IF VL1 < VL 2 ...

See also : '=', '>', '>=', '<=', '<>'.

9-10-6- Less than or equal to (<=)

Syntax : <Expression1> <= <Expression2>

Data types : Byte, Integer, Long-integer, Real

Description : This operator tests if <Expression1> is less than or equal to <Expression2>.

Remarks : <Expression1> and <Expression2> must be valid expressions and must be of the same type.

Example : VL1 =10

IF VL1<= VL1 ...

See also : '=', '>', '>=', '<', '<>'.

9-10-7- Shift left (<<)

Syntax : <Expression1> << <Expression2>

Data types : Byte or Integer

Description : This operator shifts <Expression1> to the left by <Expression2> bits.

Remarks : <Expression2> represents the number of bits to shift by. The shifting is not circular.

Example : VL1 = 4

VL2= VL1 << 2 'Result VL2= 16

See also : ' >> '.

Caution : Leave a space before and after the operator symbol.

9-10-8- Not equal to (<>)

Syntax : <Expression1> <> <Expression2>

Data types : Byte, Integer, Long-integer, Real

Description : This operator tests if <Expression1> and <Expression2> are different.

Remarks : <Expression1> and <Expression2> must be valid expressions and must be of the same type.

Example : VL1=10

IF VL2<> VL1 ...

See also : '=', '>', '>=', '<', '<='

9-10-9- Equals

Syntax : <Expression1> = <Expression2> or <Variable>=<Expression2>

Data types : Bit, Byte, Integer, Long-integer, Real

Description : This operator assigns <Variable> equal to <Expression2> or tests if <Expression1> is equal to <Expression2>.

Remarks : <Expression1> and <Expression2> must be valid expressions and must be of the same type.

Example : VL1=1

Loop :

VL1 = VL1 + 1

IF VL1 =10 GOTO Next

GOTO Loop

Next :

See also : '>', '>=', '<', '<=', '<>'

9-10-10- Greater than (>)

Syntax : <Expression1> > <Expression2>

Data types : Bit, Byte, Integer, Long-integer, Real

Description : This operator tests if <Expression1> is greater than <Expression2>.

Remarks : <Expression1> and <Expression2> must be valid expressions and must be of the same type.

Example : IF VL1 > VL2 ...

See also : '=', '>=', '<', '<=', '<>'

9-10-11- Greater than or equal to (>=)

Syntax : <Expression1> >= <Expression2>

Data types : Bit, Byte, Integer, Long-integer, Real

Description : This operator tests if <Expression1> is greater than or equal to <Expression2>.

Remarks : <Expression1> and <Expression2> must be valid expressions and must be of the same type.

Example : IF VL1 >= VL2 ...

See also : '=', '>', '<', '<=', '<>'.

9-10-12- Shift right (>>)

Syntax : <Expression1> >> <Expression2>

Data types : Byte or Integer

Description : This operator shifts <Expression1> to the right by <Expression2> bits.

Remarks : <Expression2> represents the number of bits to shift by. The shifting is not circular

Example : VL1 = 48

VL2 = VL1 >> 3 'Result VL2 = 12

See also : ' << '.

Caution : Leave a space before and after the operator symbol.

9-10-13- ACC - Acceleration

Syntax 1 : ACC = <Expression>

Syntax 2 : <Variable> = ACC

Units : User-defined units per s² (e.g. mm/s², degrees/s², revs/s² etc.)

Data types : Real

Description : This instruction reads or modifies the current acceleration value.

Remarks : <Expression> must be a valid real expression. The current acceleration can be read or modified at any time.

Example : ACC = 500

VR0 = 1000

ACC = VR0

See also : DEC, POS and VEL

9-10-14- ADC(1) – Read analogue input 1

Syntax : <Variable>= ADC(1)

Unite : Variable : Volt

Limits : Variable : +/- 10V

Data types : <Variable> : Real

Description : This function returns the voltage on analogue input 1.

Example : VR1=ADC(1)

See also : DAC, ADC(2)

9-10-15- ADC(2) – Read analogue input 2

Syntax : <Variable>= ADC(2)

Unite : Variable : Volt

Limits : Variable : +/- 10V

Data types : <Variable> : Real

Description : This function returns the voltage on analogue input 2.

Example : VR2 =ADC(2)

See also : DAC, ADC(1)

9-10-16- ACC% - Acceleration in percent

Syntax : ACC% = <Expression>

Data types : Byte

Data limits : 1 to 100

Description : This instruction modifies the current acceleration as a percentage of the acceleration parameter.

Remarks : The acceleration parameter can be set on screen Motion control / Configuration / Speed profile.

Example :
```
ACC%=10        'Set the current acceleration to 10%
VB = 50
ACC%=VB0
```

See also : DEC%

9-10-17- AND – And operator

Syntax : <Expression1> AND <Expression2>

Data types : Bit, Byte, Integer

Description : This function performs a binary AND between two expressions and returns a value of the same type as the operand.

Remarks : <Expression1> and <Expression2> must be of the same type.

Example :
```
VB3=1001111b
VB4=1111110b
VB2=VB3 AND VB4                ‘VB2=1001110b
```

See also : OR, NOT, XOR and IF

9-10-18- AXIS – Axis loop control

Syntax : AXIS ON | OFF

Description : This instruction is used to open and close the control loop.

Remarks : When the axis is in closed loop (AXIS ON), all of the movement instructions are transmitted to the axis via an intermediate movement buffer and are executed. If the axis is in open loop (AXIS OFF), the movement buffer is cleared and the instructions MOVE_S and FE_S return a value of 0.

Example :
```
AXIS ON          'closed loop control
MOVA=1000        'move to position 1000
OUT(3)=1         'set output 1
MOVA=2000
OUT(3)=0
```

Attention : See also the enable mode on screen Parameters / Digital Inputs Outputs.

See also : AXIS_S, SECURITY

9-10-19- AXIS_S – Read the state of the control loop

Syntax : AXIS_S

Description : This instruction is used to reads the state of the control loop and returns a value of 1 or 0.

Remarks : This instruction can be used at any time to see if the axis is enabled.

Example :
```
MOVA=100
If AXIS_S = 0 GOTO Error     'Error since the axis has
                             'changed to open loop.
```

See also : AXIS

9-10-20- BUFMOV_S

Syntax : <Variable>=BUFMOV_S

Data types : Byte

Description : This function returns the number of movements waiting in the buffer. The movement being currently executed is not counted by this function.

Remarks : This function can be used after having launched several movements to see if a movement is finished. When the movement buffer is full the task is blocked until a place becomes available.

Example :

```
STTR=100

STTR=50

STTR=50

WAIT BUFMOV_S<2          'Wait until the end of the first move.
```

9-10-21- CALL – Call a subroutine

Syntax : CALL <Name>

Description : This instruction is used to call a subroutine defined by a block SUB. <Name> is the name of the subroutine block.

Remarks : A subroutine cannot call itself. The execution of this instruction causes the multi-tasking controller to move on to the next task.

Example : CALL Movement

See also : SUB

9-10-22- CAMBOX

Syntax : CAMBOX (<BoxNo>, <Source>, <Segments>)

Limits : Box number : 1 to 2

Source : 0 for motor, 1 for master encoder

Segments : 1 to 4

Data types : Box number : Byte

Segments : Byte

Description : This function defines a cam box. All segments previously defined by CAMSEG are erased.

Remarks : < BoxNo > cam box number

< Segments > is the number of segments in the box. If this value is zero, the cam is destroyed and must be redefined before reuse.

Example : CAMBOX(1,1,4) 'Cam box 1, master encoder, 4 segments

See also : CAMBOXSEG

9-10-23- CAMBOXSEG – Cam box segment

Syntax : CAMBOXSEG (<BoxNo >, < SegNo >, <OutputNo>, <Start>,<End>)

Limits : Box number : 1 to 2

Segment number : 1 to 4

Output number : 1 to 10

Units : Start, End : User-units

Data types : Box number, Segment number, Output number : Byte

Start, End : Real

Description : This function defines one segment of a cam box.

Remarks : The output is set to 1 between <Start> and <End>.

Example : CAMBOXSEG(1,2,4,0,90) 'The second segment of box 1 sets output 4 between 0 and 90° (the user units having been defined as degrees) .

See also : CAMBOX

9-10-24- CAPTURE1

Syntax : CAPTURE1 (<Source>, <InputNo>, <Window>, <Min>, <Max>, <Inside>)

Description : The instructions CAPTURE1 and CAPTURE 2 are used to register the actual position of the axis or the master encoder on the rising edge of an input.

When the rising edge is detected, the position is stored in variable REGPOS1_S. The flag REG1_S is also set to true.

Data types : <Source> 0 for motor position, 1 for master encoder.

<InputNo> The input used to detect the rising edge (1 to 16)

<Window> If window is true, the input is only tested between the positions <Min> and <Max>.

<Inside> Defines whether the test is performed inside or outside the limits of the window <Min> and <Max>.

<Min> must always be less than <Max>.

Example : CAPTURE1(0,4,1,10,20,1) 'Capture motor position on the rising edge of input 4 when the axis is between 10 and 20.

```
WAIT REG1_S = 1          'Wait for the capture
VR1 = REGPOS1_S          'VR1 = captured position
```

See also : REG1_S or REG2_S, REGPOS1_S or REGPOS2_S

9-10-25- CLEAR – Clear the axis position

Syntax : CLEAR

Description : This instruction sets the axis position to zero.

Example :

```
CLEAR
VR1=POS_S   'Result : VR1=0.0
```

9-10-26- CLEARMASTER – Set the master encoder position to zero

Syntax : CLEARMASTER

Description : This instruction zeros the position of the master encoder.

Example :

```
CLEARMASTER
```

9-10-27- CONTINUE – Continue the execution of a task

Syntax : CONTINUE <TaskNo>

Description : This instruction is used to continue the execution of a suspended task.

Remarks : <TaskNo> is the number of the suspended task. This function has no effect on a stopped task or a running task.

Example :

```
Wait Inp(9)
RUN 2
Begin:
    Wait Inp(9)
    SUSPEND 2
    Wait Inp(8)
```

```
                CONTINUE 2
            Goto Begin
```

See also : RUN, HALT, SUSPEND

9-10-28- COUNTER - Initialise counter with a value

Syntax : COUNTER(1 or 2) = <Value>

Data types : <Value> : value between 0 and 65535

Description : The instruction COUNTER(1 or 2) is used to write a value to counter 1 or 2.

Example : COUNTER(2)=VL1+1000

See also : SETUPCOUNTER

9-10-29- COUNTER_S – Read a counter

Syntax : <Variable>=COUNTER_S(<CounterNo>)

Description : The instruction COUNTER_S reads the value of a counter.

Data types : <Variable> Integer between 0 and 65535

<CounterNo> counter number (1 or 2)

Example : VI0 = COUNTER(1)

9-10-30- DAC – Analogue output

Syntax : DAC = <Expression>

Units : Volts

Limits : -10 to +10

Data types : Real

Description : This function sets the voltage on the analogue output.

Remarks : The value on the analogue output can also be read.

Example : DAC=5.2

IF ADC(1)>DAC ...

See also : ADC(1), ADC(2)

9-10-31- DEC - Deceleration

Syntax 1 : DEC = <Expression>

Syntax 2 : <Variable> = DEC

Units : User-defined units per s² (e.g. mm/s², degrees/s², revs/s² etc.)

Data types : Real

Description : This instruction reads or modifies the current deceleration value.

Remarks : <Expression> must be a valid real expression. The current deceleration can be read or modified at any time.

Example : DEC = 500.

VR0 = 10000

DEC = VR0

See also : ACC, VEL

9-10-32- DEC% - Deceleration in percent

Syntax : DEC% = <Expression>

Data types : Byte

Data limits : 1 to 100

Description : This instruction modifies the current deceleration as a percentage of the acceleration parameter.

Remarks : The deceleration parameter can be set on screen Motion control / Configuration / Speed profile.

Example : DEC% = 10 'Set deceleration to 10 %

VB0 = 50

DEC% = 50

See also : ACC% and VEL%

9-10-33- DELAY – Passive wait

Syntax : DELAY <Duration>

Units : milliseconds

Data types : Integer

Description : This function initiates a passive delay for the specified duration. The task is blocked by this instruction, which passes execution on to the next task.

Example : DELAY 500 'Delay of 0.5 s.

or

VI12=500

DELAY VI12

9-10-34- DISPLAY – 7 segment display

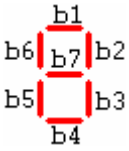
Syntax : DISPLAY <Expression>

Data types : Expression : Byte

Description : This instruction sets one or more of the individual segments of the LED display.

Remarks : Each bit of <Expression> represents a segment. The MBS is not used.

Example : Display 109 ' Equivalent to Display 01101101b or « 5 »



9-10-35- EXIT SUB – Exit a subroutine

Syntax : EXIT SUB

Description : This instruction exits a subroutine.

See also : SUB

9-10-36- FEMAX_S – Following error limit

Syntax : FEMAX_S

Description : This flag is set to 1 when the following error exceeds the level in the *following error parameter,* accessible from the menu Parameters / Supervision / Position.

Remarks : This function can be used to determine if a following error fault has occurred. If the instructions SECURITY(0) or SECURITY(1) have been used, it is recommended that this flag be monitored in a dedicated error-handling task.

The flag is reset to zero:

- If input 1 is configured as *NONE*, FEMAX_S is set to 0 with an Axis On instruction in a task or on the rising edge of the enable button in the main DPL window.
- If input 1 is configured as *ENABLE*, FEMAX_S is set to 0 on the rising edge of this input.
- If input 1 is configured as *ENABLE+DPL*, FEMAX_S is set to 0 is input 1 = 1 and an Axis On instruction has been executed in a task.

Example : IF FEMAX_S = 1 GOTO Error

GOTO Start

Error :

See also : FE_S, SECURITY

9-10-37- FE_S – Following error

Syntax : FE_S

Description : This function returns the value of the actual following error.

Remarks : This can be used the verify the performance of the axis control in real time.

Example : VR1 = FE_S

See also : FEMAX_S

9-10-38- FRAC – Fractional part

Syntax : FRAC(<Expression>)

Data types : Real

Description : This function returns the fractional part of <Expression>.

Remarks : The result is real.

Example : VR2=3.0214

VR1=FRAC(VR2) 'Result VR2=0.0214

See also : INT

9-10-39- GEARBOX

Syntax : GEARBOX(<Numerator>, <Denominator>)

Description : This instruction provides a gearbox function between a master encoder and the motor (slave axis).

Data types : <Numerator> Integer or value between 0 and 8

<Denominator> Integer or value between 0 and 32767

<Numerator> / <Denominator> defines the ration between the master encoder and the slave motor.

Remarks : This instruction does not block the task (unless the movement buffer is full). So long as the link between the master and slave is not broken, the instruction MOVE_S will give a value of 1 (even if the slave is stopped).

Example : GEARBOX (1, 2) 'Ratio 0.5

See also : GEARBOXRATIO, STARTGEARBOX, STOPGEARBOX

9-10-40- GEARBOXRATIO

Syntax : GEARBOXRATIO(<Ratio>)

Description : This instruction modifies the ratio of an electronic gearbox.

Data types : <Ratio> 0 to 65535. The ratio of the gearbox is defined by

(<Ratio> × <Numerator>) / (<Denominator> × 4096).

<Numerator> and <Denominator> are parameters the GEARBOX instruction.

Remarks : The instruction is non-blocking and allows the ratio to be changed at any time without stopping the gearbox.

Example : GEARBOXRATIO(2048)

See also : GEARBOX, STARTGEARBOX, STOPGEARBOX

9-10-41- GOTO – Jump to a label

Syntax : GOTO <Label>

Description : Jump to a label

Remarks : A label is a name followed by a ":". The execution of this instruction causes the multi-tasking controller to move on to the next task.

Example : GOTO Begin

...

Begin :

See also : IF

9-10-42- HALT – Stop a task

Syntax : HALT <TaskNo>

Description : This instruction is used to stop a running task or a suspended task..

Remarks : This function has no effect on a task already stopped. It does not affect current movements or the movement buffer.

Example : Begin :

Wait Inp(8)=On

RUN 2

Wait Inp(8)=Off

HALT 2

Goto Begin

Warning: After HALT function, it is recommend to wait the completely stop of the task: Wait Status (n°task) =0

See also : RUN, SUSPEND, CONTINUE

9-10-43- HOME – Go to home datum

Syntax : HOME(<Type>,[Reference])

Description : This function forces the axis to return to its home position using the method defined by <Type>. This instruction blocks the task until the homing is complete and also causes execution to transfer to the next task. Homing uses the speed set on the screen Motion control / Home.Values for <Type> are :

0 : immediate

1 : On Top Z : no movement is doing, drive calculate his position on Top Z, the new position various between +/- ½ motor rev.

2 : On sensor input (without release), positive direction

3 : On sensor input (with release), positive direction

4 : On sensor input (without release), negative direction

5 : On sensor input (with release), negative direction

6 : On sensor and Top Z (without release), positive direction

7 : On sensor and Top Z (with release), positive direction

8 : On sensor and Top Z (without release), negative direction

9 : On sensor and Top Z (with release), negative direction

[Reference] optional home position value

Remarks : Use AXIS Off to stop a homing operation. If <Type> is not specified, the value is the type defined in the Home set-up menu.

Example : VR0=100

HOME (3,VR0) 'Go home using mode 3 and a home position of 100

Note : If the [Reference] value is not given it is 0.

HOME(2) 'is equivalent to VR0=0 and HOME(2,VR0)

See also : HOME_S

9-10-44- HOME_S – Read homing status

Syntax : HOME_S

Description : This function reads the homing status

Remarks : This function shows if the homing has been completed or not. During a homing cycle the HOME_S flag is forced to 0. When the cycle is complete the HOME_S flag becomes a 1.

Example : IF HOME_S = OFF GOTO Next

Next :

See also : HOME

9-10-45- IF - IF…

Syntax : IF <Condition> GOTO {<Label>}

Description : Performs a conditional jump to a label based on the evaluation of an expression. If <Condition> is true then jump to <Label>.

Remarks : <Condition> must be a Boolean expression.

Example : IF VR1=150 GOTO SUITE

9-10-46- INP – Read a digital input

Syntax : INP (<InputNo>)

Data types : Value from 1 to 16.

Description : This function returns the state of a digital input.

Remarks : <InputNo> represents the number of the digital input. The returned data type is Bit.

Example : VF1 = INP(11)

See also : INPB, INPW, OUT, OUTB

9-10-47- INPB – Read a block of 8 inputs

Syntax : INPB (<BlockNo>)

Data types : Value 1 or 2.

Description : This function returns the state of a block of 8 digital inputs.

Remarks : <BlockNo> represents the input block number. The returned data type is Byte.

Example : VB1=INPB(2)

See also : INP, INPW, OUT, OUTB

9-10-48- INPW – Read 16 digital inputs

Syntax : INPW

Description : This function returns the state of the block of 16 digital inputs.

Remarks : The returned data type is Integer.

Example : VI2=INPW

See also : INP, INPB, OUT, OUTB

9-10-49- INT – Integer part

Syntax : INT (<Variable>)

Data types : Real

Description : This function returns the integer part of < Variable >.

Example : VR1=25.36

VR2=INT(VR1) 'Result : VR2=25

See also : FRAC

9-10-50- LOADPARAM – Reload the drive parameters

Syntax : LOADPARAM

Description : Transfers the drive parameters, saved in Flash memory, into the working RAM.

See also : SAVEPARAM

9-10-51- LOADVARIABLE – Load saved variables

Syntax : LOADVARIABLE

Description : Transfers the variables VR0 to VR63 and VL0 to VL63, saved in Flash memory, into the working RAM.

See also : SAVEVARIABLE

9-10-52- LOADTIMER – Load a variable with a timer value

Syntax : LOADTIMER(<VL n°XX>)=<Value>

Data types : Value : Long-integer

Description : The instruction LOADTIMER can be used to provide an active wait. Variable VLXX is loaded with the sum of **Time** + <Value>

Remarks : Up to 256 timers can be used simultaneously.

Example : LOADTIMER(VL129)=4688 ‘Load a time of 3000ms in variable VL129

See also : TIMER

9-10-53- LOOP – Virtual mode

Syntax : LOOP ON/OFF

Description : This function puts the axis into a virtual mode and allows a program to be tested with neither an encoder nor a motor. In this mode do not supply power to connector X10

9-10-54- MERGE – Chain movements

Syntax : MERGE ON | OFF

Description : This instruction is used to activate or deactivate the chaining of consecutive movements.

Example :

```
MERGE ON
TRAJA(1000,500)          'Movements chained without
TRAJA(1500,200)          'passing through zero speed
MERGE OFF
TRAJA(1800,700)          'Pass through zero speed at position 1500
```

9-10-55- MOD - Modulus

Syntax : <Expression1> MOD <Expression2>

Data types : Byte, Integer, Long-integer

Description : This operator returns the remainder from an integer division.

Example :

```
VI10=5
VI10=VI10 MOD 2    'Result : VI10=1
```

9-10-56- MOVA – Move absolute

Syntax : MOVA = <Distance>

Units : User-defined units, e.g. mm, degrees

Data types : Real

Description : Move the axis to an absolute position. This instruction causes execution to transfer to the next task.

Remarks : The task waits for the end of the movement (MOVE_S=0) before executing the next instruction. The axis uses the current values of speed, acceleration and deceleration.

Example : MOVA = 1200.00

See also : MOVR, STTA, STTR, STTI and MOVE_S

9-10-57- MOVE_S – Movement status

Syntax : MOVE_S

Data types : Bit

Description : This function indicates if the axis is moving..

Remarks : If the axis is open loop (AXIS OFF), the instruction MOVE_S = 0. If the axis is closed loop, MOVE_S is equal to 0 if the 4 following points are true :

The current movement is complete.

The following error is within the positioning window.

The movement buffer is empty.

In the case of a slave axis linked by a gearbox function, the link must already have been broken.

If one of these points is false, the instruction MOVE_S returns a value of 1.

Example :

```
STTA = VR10

WAIT MOVE_S = OFF        'Wait until the axis is stopped
```

9-10-58- MOVR – Move relative

Syntax : MOVR = <Distance>

Data types : Real

Description : Move the axis to a relative position. This instruction causes execution to transfer to the next task.

Remarks : The task waits for the end of the movement (MOVE_S=0) before executing the next instruction. The axis uses the current values of speed, acceleration and deceleration.

Example :

```
MOVR = VR1
```

See also : MOVA, STTA, STTR, STTI, MOVE_S

9-10-59- NEXTTASK

Syntax : NEXTTASK

Description : This instruction causes the multi-tasking controller to move on immediately to the next task.

9-10-60- NOT – Complement operator

Syntax : NOT(<Expression>)

Data types : Bit, Byte, Integer

Description : The NOT returns the complement of the expression..

Example :
```
VB1=15
VB2=NOT VB1          'Result VI2=140
```

See also : AND, OR, XOR

9-10-61- OR – Or operator

Syntax : <Expression1> OR <Expression2>

Data types : Bit, Byte, Integer

Description : This function performs a binary OR between two expressions and returns a value of the same type as the operand.

Remarks : <Expression1> and <Expression2> must be of the same type.

Example : VI12=VI12 OR 000FFh

See also : AND, NOT, XOR and IF

9-10-62- ORDER – Movement order number

Syntax 1 : ORDER = <Value>

Syntax 2 : ORDER

Data types : Value between 0 and 65535

Description : This instruction sets the order number of the next movement or reads the order number of the last movement.

Remarks : This instruction can be used with the ORDER_S function.

Example :
```
ORDER = 0
STTA = 50
VB1 = ORDER          'Result : VB1=1
```

See also : ORDER_S

9-10-63- ORDER_S – Current order number

Syntax : ORDER_S

Data types : Integer

Description : This function returns a value for the order number of the movement currently being executed.

Remarks : This function can be used to determine the state of a movement.

Example : ORDER=0

STTA = 50

STTA = 100

STTA = 50

IF ORDER_S=2 ...'The second movement has started

See also : ORDER

9-10-64- OUT – Write a digital output

Syntax : OUT (<OutputNo>) = <Expression>

Data types : Expression : Bit

Description : This function sets the state of a digital output.

Remarks : <OutputNo> represents the number of the digital output, 1 to 10

Example : OUT(10) = ON

See also : INP, INPB, INPW, OUTB

9-10-65- OUTB – Write a block of 8 outputs

Syntax : OUTB (<BlockNo>) = <Expression>

Data types : <Expression> : Byte

<BlockNo> : 1 or 2

Description : This function sets the states of 8 digital outputs.

Example : OUTB(1)=15

See also : INP, INPB, INPW, OUT

9-10-66- POS – Target position

Syntax 1 : POS = <Expression>

Syntax 2 : POS

Data types : Real

Description : This function returns or sets the target position in the chosen units.

Remarks : This function can be used to change the target position during the course of a movement. The position can be changed at any time.

Example :

```
STTA = 5000                 'Start the axis
WAIT INP(10) = On           'Wait for an input
POS = POS_S+50.             'Stop 50mm after the sensor input
WAIT MOVE_S = OFF           'Wait until the axis is stopped
```

See also : ACC, DEC, VEL

9-10-67- POS_S – Actual position

Syntax : <Expression> = POS_S

Data types : Real

Description : This function returns the actual position of the axis.

Remarks : With this you can obtain the axis position in real time.

Example :

```
STTA = 100                  'Start the axis
OUT(5) = 1                  'Set output 5
Loop :
        VR1=POS_S
        IF VR1<50 GOTO Loop
OUT(5) = 0                  'Clear output 5
```

See also : VEL_S

9-10-68- PROG .. END PROG – Main program block

Syntax : PROG

Description : This keyword defines the start of the main program block. When used in conjunction with END it is used to define the end of the main program block.

Remarks : Only one PROG - END PROG block can be defined in a task.

Example :
```
PROG
...
END PROG
```

9-10-69- READPARAM – Read a parameter

Syntax : <Variable> = READPARAM (<Index>, <Sub-Index>)

Data types : <Variable> Long-integer

<Index> Integer

<Sub-Index> Byte

Description : This function allows a task to read the status and parameters of the drive via the CANopen dictionary.

Example :
```
VL0 =  READPARAM(8448,1) ‘Read the drive fault number.
```

9-10-70- REG1_S

Syntax : <VFx>=REG1_S

Description : This function indicates if a position capture has taken place..

Remarks : The returned value is only true once per capture. REG1_S is automatically reset to 0 after a read operation and also on re-launching another capture.

Example :
```
CAPTURE1(0,4,1,10,20,1)   'Capture the motor position
                          ‘on the rising edge of input 4
                          ‘when the axis is between 10 and 20
WAIT REG1_S = 1           ‘Wait for the capture
VR1 = REGPOS1_S           ‘VR1 = captured position
```

See also : CAPTURE1 or CAPTURE2, REGPOS1_S or REGPOS2_S

9-10-71- REGPOS1_S

Syntax : <VR XX>=REGPOS1_S

Description : This function returns the last position captured by execution of the instruction CAPTURE1.

Example :

```
CAPTURE1(0,4,1,10,20,1)   'Capture the motor position
                          ‘on the rising edge of input 4
                          ‘when the axis is between 10 and 20
WAIT REG1_S = 1           ‘Wait for the capture
VR1 = REGPOS1_S           ‘VR1 = captured position
```

See also : CAPTURE1 or CAPTURE2, REG1_S or REG2_S

9-10-72- RESTART – Restart the system

Syntax : RESTART

Description : restart the system in the same way as at power-on.

9-10-73- RUN – Start a task

Syntax : RUN <TaskNo>

Description : This instruction is used to start a stopped task, e.g. a task declared as 'Manual'.

Remarks : This function has no effect on a suspended task or a task already started.

Example :

```
Start:

Wait Inp(11)=On

RUN  3

Wait Inp(11)=Off

HALT 3

Wait Status (3) =0

Goto Start
```

Warning: After HALT function, it is recommend to wait the completely stop of the task: Wait Status (n°task) =0

See also: CONTINUE, HALT, SUSPEND

9-10-74- SAVEPARAM - Save drive parameters

Syntax : SAVEPARAM

Description : The drive parameters in the working RAM are saved in Flash memory.

Remarks : The Flash memory has a life-time limit of 5000 write cycles.

See also : LOADPARAM

Attention : Excessive execution of this instruction can cause the premature degradation of the Flash memory.

9-10-75- SAVEVARIABLE – Save variables

Syntax : SAVEVARIABLE

Description : Variables VR0 to VR63, VL0 to VL63 in the working RAM are saved in the Flash memory.

The drive automatically passes to AXIS OFF

Remarks : The Flash memory has a life-time limit of 5000 write cycles.

See also : LOADVARIALBE

Attention : Excessive execution of this instruction can cause the premature degradation of the Flash memory.

9-10-76- SECURITY – Defines security actions

Syntax : SECURITY(<Level>)

Description : This instruction is used to define how the system will react when a following error is detected. <Level> determines the level of security. At power-on, the default value is SECURITY(2)

Level	Err. 12 on display	Flag Femax	Axis_S	S1 (ready)
0	No	1	Axis_s = On	1
1	No	1	Axis_s = Off	1
2	Yes	1	Axis_s = Off	0

.

Remarks : If the SECURITY instruction is used, the level of security can be reduced by a task. It is recommended not to use this instruction.

Example : SECURITY(0) ' The drive remains enabled with an excess following error.

Note : The flag Femax_S is reset to 0 each time the axis is enabled (Axis On).

9-10-77- SETUPCOUNTER – Configure a counter

Syntax : SETUPCOUNTER(<1 or 2>, <InputNo>, <Filter>)

Data types : <Filter> : Bit

Description : This instruction configures counter 1 or 2

Remarks : <InputNo> : Input number from 1 to 16

<Filter> : Filter activation : 0 for no filter, 1 for a filter.

See also : COUNTER

Attention : If the filter is not active, the maximum frequency is 781Hz (1.24ms), otherwise it depends on the Filter parameter in Parameters / Digital Inputs Outputs.

9-10-78- SSTOP – Stop the axis

Syntax : SSTOP

Description : This function stops the axis using the current deceleration. This function does not block the task.

Remarks : The axis stops even if the axis is linked by the GEARBOX function.

The instruction SSTOP empties the movement buffer and stops the axis using the current deceleration.

Example : SSTOP

See also : STTA, STTR, STTI, GEARBOX,

9-10-79- STARTCAMBOX – Start a cam box

Syntax : STARTCAMBOX(<BoxNo>)

Description : This instruction starts a previously defined cam box.

Remarks : If the cam box has not been defined, the instruction has no effect. <BoxNo> is the number used in the instruction CAMBOX.

Example : STARTCAMBOX(1)

See also : CAMBOX

9-10-80- STARTGEARBOX – Start electronic gearbox

Syntax : STARTGEARBOX(<Acceleration>)

Description : This instruction starts the electronic gearbox using the specified acceleration and the ration previously defined by the instruction GEARBOXRATIO.

Data types : <Acceleration> 0 to 65535

Remarks : The acceleration phase is : (Ratio × 640µs) / Acceleration, with Ratio defined by GEARBOXRATIO.

Example : STARTGEARBOX(512) 'Start a gearbox with an acceleration phase

… 'of Ratio×640µs/512

See also : GEARBOX, GEARBOXRATIO, STOPGEARBOX

9-10-81- STATUS – Task status

Syntax : STATUS (<TaskNo>)

Description : This function returns the state of a task

Remarks : Possible values are :

0 : The task is stopped

1 : The task is suspended

2 : The task is running

Example : Run 2

Wait Status(2)=0

9-10-82- STOP - Stop the axis

Syntax : STOP

Description : This function stops the axis using the current deceleration. This function blocks the task until the axis has stopped.

Remarks : The axis stops even if the axis is linked by the GEARBOX function.

The instruction STOP empties the movement buffer and stops the axis using the current deceleration. This instruction blocks the task until MOVE_S is 0.

Example : STOP

See also : STTA, STTR, STTI, GEARBOX

9-10-83- STOPCAMBOX – Stop a cam box

Syntax : STOPCAMBOX(<BoxNo>)

Description : This instruction stops a previously defined cam box.

Remarks : <BoxNo> is the number used in the instruction CAMBOX. This function does not destroy the cam box.

Example : STOPCAMBOX(1)

See also : CAMBOX, CAMBOXSEG, STARTCAMBOX

9-10-84- STOPGEARBOX – Stop electronic gearbox

Syntax : STOPGEARBOX(<Deceleration>)

Description : This instruction stops the electronic gearbox using the specified deceleration and the ration previously defined by the instruction GEARBOXRATIO.

Data types : <Deceleration> 0 to 65535

Remarks : The deceleration phase is : (Ratio × 640µs) / Deceleration, with Ratio defined by GEARBOXRATIO.

Example : STOPGEARBOX(256) ‘Stop a gearbox with an deceleration phase

```
WAIT MOVE_S=0
VR0=POS_S
HOME (0,VR0) ‘update theoretical position
```

See also : GEARBOX, GEARBOXRATIO, STARTGEARBOX

9-10-85- STTA – Start absolute movement

Syntax : STTA = <Distance>

Data types : Real

Description : Start a movement to an absolute position

Remarks : The system does not wait for the end of the movement (MOVE_S=0) before executing the next instruction. The axis uses the current values of speed, acceleration and deceleration.

Example : STTA = 1200.00

WAIT MOVE_S = OFF

See also : MOVA, MOVR, STTR, STTI

9-10-86- STTI – Start infinite movement

Syntax : STTI(+ or -)

Description : Start an infinite movement.

Remarks : The system immediately execute the next instruction. To stop the movement you must use STOP or SSTOP. . The axis uses the current values of speed and acceleration.

Example : STTI (+) ' start an infinite movement in the positive direction

See also : MOVA, MOVR, STTA, STTR, STOP

9-10-87- STTR – Start a relative movement

Syntax : STTR = <Distance>

Data types : Real

Description : Start a relative movement.

Remarks : The system does not wait for the end of the movement (MOVE_S=0) before executing the next instruction. The axis uses the current values of speed, acceleration and deceleration.

Example : VR0 = 420

STTR = VR0

See also : MOVA, MOVR, STTA, STTI

9-10-88- SUB .. END SUB – Subroutine

Syntax : SUB <Name>

Description : This keyword defines the start of a subroutine. When used in conjunction with END it is used to define the end of a subroutine.

Remarks : SUB - END SUB blocks must be outside the main program block defined by PROG – END PROG.

Example :
```
SUB    Move
...
END SUB
```

9-10-89- SUSPEND – Suspend a task

Syntax : SUSPEND <TaskNo>

Description : This instruction suspends a running task.

Remarks : This instruction has no effect on stopped tasks. It does not affect current movements or the movement buffer.

Example :
```
Wait Inp(12)
RUN  4
Begin:
        Wait Inp(12)
        SUSPEND 4
        Wait Inp(12)
        CONTINUE 4
Goto Begin
```

See also : RUN, CONTINUE, HALT

9-10-90- TIME – Extended time base

Syntax : <VLx> = TIME

Description : The system variable TIME can be used to give an active wait. TIME is a long-integer that represents the number of 0.640 thousandths of second since the last power-on.

Example :
```
VL2=TIME + 7812          'Load a time of 5000ms
LOOP  :
VL3 = TIME
 IF VL3<VL2 GOTO LOOP
```

Warning : TIME don't work in a test.

9-10-91- TIMER – Compare a variable to Time

Syntax : TIMER(<VL XX>)

Description : This instruction compares the system variable TIME with the contents of variable VLXX :

TIMER(VLXX) =1 if Time<=VLXX (timing in progress).

TIMER(VLXX)=0 if Time>VLXX (timing over).

Data types : VL XX : Long-integer

Example : LOADTIMER(VL122)=4688 'Load a time of 3s

WAIT (TIMER(VL122)=0) 'Wait until the time has elapsed

9-10-92- TRAJA – Absolute trajectory

Syntax : TRAJA (<Position>,<Speed>)

Data types : Real

Description : This instruction can be used to produce a complex movement. This instruction causes execution to be switched to the next task.

Remarks : The axis uses current acceleration and deceleration values.

Example : MERGE On

TRAJA (1000.00, VR0) 'Move at slow speed to position 1000

TRAJA (1500.00, VR1) 'Change speed without passing through 0

MERGE Off

See also : STTA, MERGE, TRAJR

9-10-93- TRAJR – Relative trajectory

Syntax : TRAJR (<Position>,<Speed>)

Data types : Real

Description : This instruction can be used to produce a complex movement. This instruction causes execution to be switched to the next task.

Remarks : The axis uses current acceleration and deceleration values.

Example : MERGE On

TRAJR (200.00, VR0) Move at a slow speed

TRAJR (1000.00, VR0)'to position 1200.

TRAJR (1500.00, VR1)'Change speed without passing through 0

MERGE Off

See also : STTR, MERGE, TRAJA

9-10-94- VEL - Speed

Syntax : VEL = <Expression>

Units : User-defined units per second, e.g. mm/s, revs/s, degrees/s.

Data types : Real

Description : This value specifies the current speed in units per second.

Remarks : <Expression> must be a valid real expression. The speed value can be modified at any time.

Example : VEL = 2000

See also : ACC, DEC, POS

9-10-95- VEL% - Speed in percent

Syntax : VEL% = <Expression>

Data types : Byte

Limits : 0 to 100

Description : this function adjusts the current speed as a percentage of speed parameter in screen Motion control / Configuration / Speed profile.

Example : VB0 = 50

VEL% = VB0

See also : ACC%, DEC%

9-10-96- VERSION – OS (Firmware) version

Syntax : <VI _XX>=VERSION

Description : This function returns the version of the operating system.

9-10-97- WAIT – Wait for a condition

Syntax : WAIT <Condition>

Description : Waits until the condition is true.

Example : WAIT INP(11)=On 'Passive wait

9-10-98- WRITEPARAM – Write a parameter

Syntax : WRITEPARAM (<Index>, <Sub-Index>) = <Variable>

Data types : <Variable> Long-integer

<Index> Integer

<Sub-Index> Byte

Description : This function allows a task to write parameters to the drive via the CANopen dictionary.

Example : WRITEPARAM(9984,6) = 1 ‘Set the axis as modulo

9-10-99- XOR – Exclusive OR operator

Syntax : <Expression1> XOR <Expression2>

Data types : Bit, Byte, Integer

Description : This function performs a binary Exclusive OR between two expressions and returns a value of the same type as the operand.

Remarks : <Expression1> and <Expression2> must be of the same type.

Example : IF VL1 XOR 0FF00h ...

See also : AND, OR, NOT, IF

10- Appendix

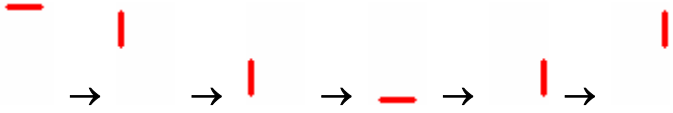
10-1- STATUS 7 segments display

10-1-1- Message descriptions :

- **On powering of the drive:**

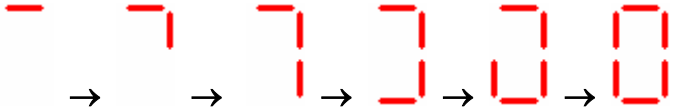
  - **BOOT initialisation phase :**

    All the segments of the display flash several times then light in the following order:

  - **OS initialisation phase :**

    The segments light in the following order but with different times:

    The complete initialisation lasts 7s.

  - **After initialisation :**

The output 'Drive Ready' (S1) is active. If DPL is in use : the automatic tasks are launched and there should remain only one point which flashes.

If DPL is not in use the segments of the display light in sequence as the motor shaft turns

If DPL is in use only the decimal point remains. The segments can be modified using the instruction 'Display' in a DPL task.

- **During drive operation :**

On the occurrence of an error:

The numbers of the errors are displayed in order.

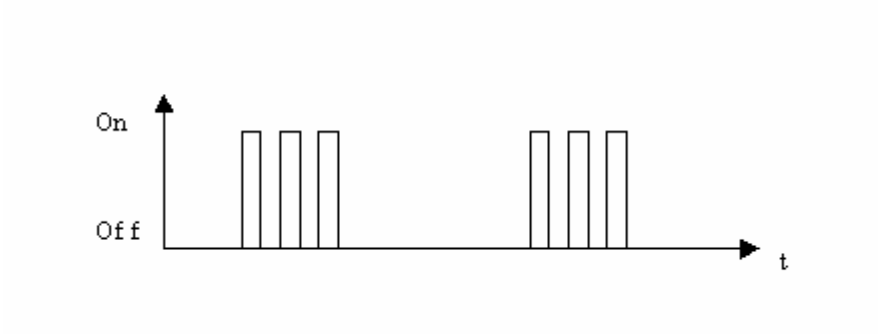
e.g. : For a motor temperature error E7 and an encoder error E8 we see :

On the removal of a fault:

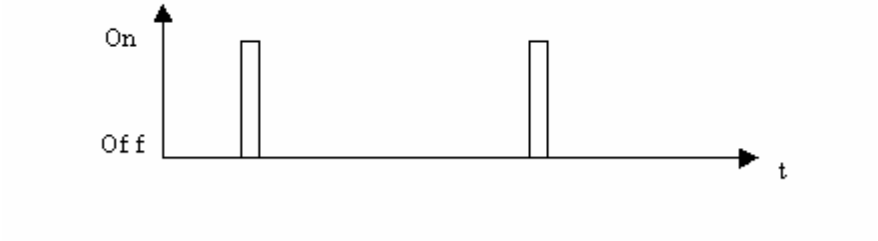
Removal of the error number and return to a normal display (as after the initialisation)

Flashing decimal point :

- If system serial connection present (RTS high) :



- If no system serial connection:



10-1-2- Error messages :

- **List of errors :**

DC Bus over-voltage : an over-voltage has been detected on the internal dc bus.
This fault can be due either to an over-voltage on the supply or to the braking resistance being insufficient.

DC Bus under-voltage : an under-voltage has been detected on the internal dc bus.
This condition is only monitored when the drive is active (Enable = ON).

I²t motor : I²t motor detected.

Over-current : a current greater than the maximum current has been detected.

Short-circuit : a short-circuit between phases or between a motor phase and earth has been detected.

Temperature IGBT : maximum temperature attained in the drive.

Temperature motor : maximum motor temperature attained.

Resolver fault : Resolver feedback signals defective.

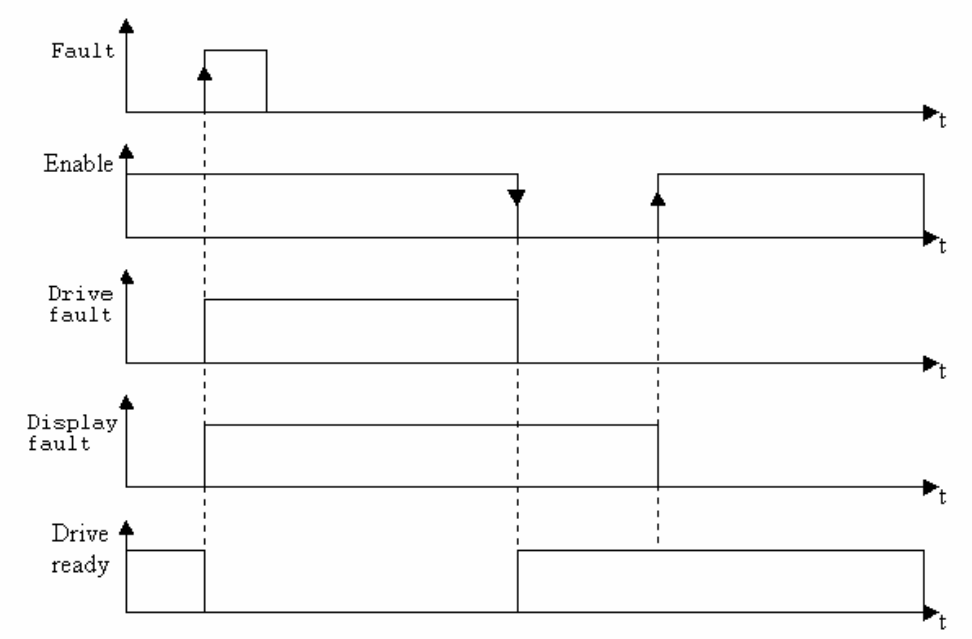
Invalid parameters : checksum error on the drive parameters.

Drive type error : the parameter file does not correspond to the drive type.

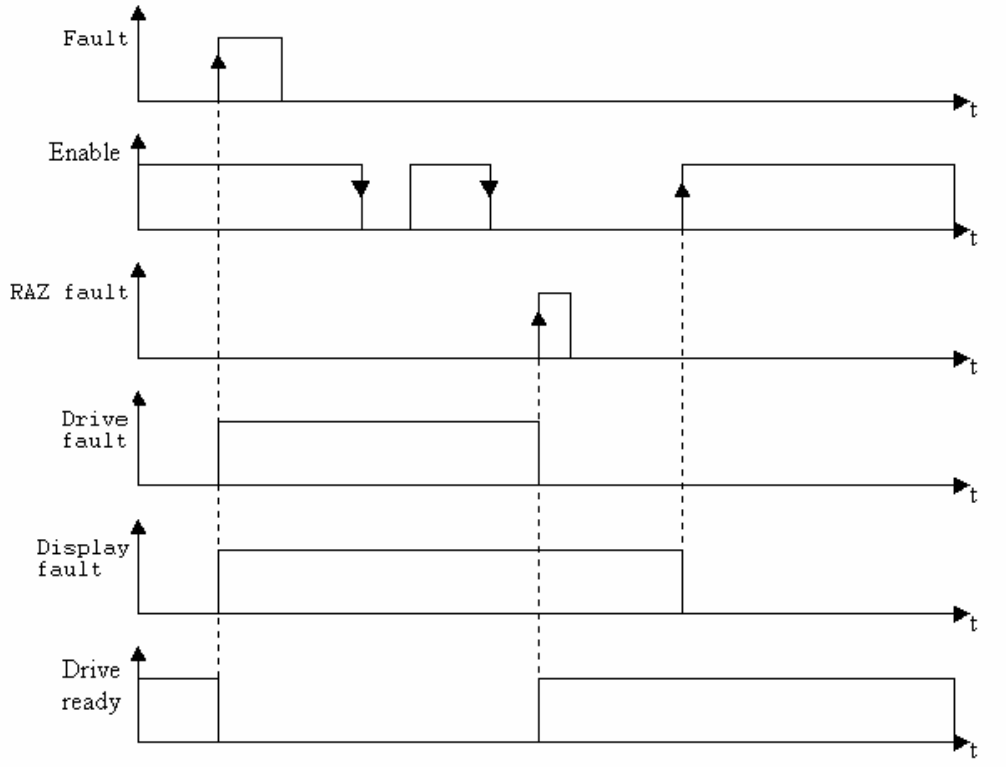
DPL error : an error has been detected during the execution of the DPL tasks.

Following error : the maximum following error has been exceeded.

- **Fault reset :**
  - If input E4 is not configured as Fault Reset, proceed as follows :



- If input E4 is configured as Fault Reset, proceed as follows :



10-2- CANopen

10-2-1- Definition

A) Introduction

The CAN (Controller Area Network) bus appeared in the middle of the 80's to respond to the requirements of data transmission in the automobile industry. This type of bus makes it possible to obtain high data transfer rates.

The CAN specifications define 3 layers in the model OSI : the physical layer, the data link layer and the application layer. The physical layer defines the mode of data transmission. The data link layer represents the core of the CAN protocol since this layer is responsible for controlling the transmission, bus arbitration, error detection, etc. The last layer is the application layer also referred to as CAL (CAN Application Layer). This is therefore a general description of the language for the CAN network that offers a number of communication services.

CANopen is a type of network that is based on a serial link and on the CAL application layer. CANopen only supports part of the communication services offered by CAL. The advantages are that this only needs a low-performance processor with low memory requirements.

CANopen is, therefore, an application layer standardised by the CIA (CAN In Automation) specifications : DS-201…DS-207.

The network manager allows for simplified network initialisation. The network can be extended to contain any other necessary components.

The CAN bus is a multi-master bus. Unlike in other field-buses, the messages are identified and not the connected modules. The network elements are allowed to send their messages each time the bus is free. Bus conflicts are resolved by a priority level given to each message. CAN bus messages are divided into 2032 priority levels. All elements of the network have the same rights and so this form of communication is only possible without a bus master.

Each element decides for itself when data is to be sent. It is, however, possible to send data by another means. This demand is made by the remote device.

The CANopen specifications (DS-201…DS-207) define the technical and functional characteristics required by any device connected to the network. CANopen makes a distinction between devices that are servers and clients.

B) CANopen communication

The CANopen communication profile allows information for the data exchange and the parameters to be specified in real time. CANopen uses services optimised for different types of data.

↳ PDO (Process Data Object)

⇨Exchange data in real time

⇨High priority identifier

⇨ Synchronous or asynchronous transmission

⇨ Maximum of 8 bytes (one message)

⇨ Pre-defined format

↳ SDO (Service Data Object)

⇨ Access the objects dictionary of a device

⇨ Low priority identifier

⇨ Asynchronous transmission

⇨ Data distributed in multiple messages

⇨ Data addressed with an index

The information sent on the CAN are received and evaluated by all connected devices. Each service of a CAN device is configured by a COBID (Communication OBject Identifier). The COBID is an identifier that characterises the message. It is this parameter that indicates to a device whether or not the message must be treated. For each service (PDO or SDO), it is necessary to specify a COBID during the transmission (send a message) and a reception COBID (receiving a message). For the first SDO server the COBID is fixed and cannot be modified remotely. Moreover, it is calculated from the NODE-ID. The NODE-ID is the parameter that characterises the device and permits a unique access to it.

PDO (Process Data Object)

This is a data exchange arbitrated between two modules. The PDO can transfer in turn controlled synchronizations or events to carry out the message sending request. With the controlled events mode, the bus loading can be reduced to a minimum. A devices can therefore obtain a high performance with a low transfer rate.

Data exchange with the PDO uses the advantages of CAN :

- Sending messages can be done from an asynchronous event (controlled event).
- Sending messages can be done from the reception of a synchronizing event.
- Recovery from a remote frame.

SDO (Service Data Object)

This is a point-to-point data exchange. A device asks for access to the list of SDO objects. The SDO replies with information corresponding to the type of request. Each SDO can be client or server. An SDO server cannot send a request to another SDO, it can only respond to a request from a client SDO. Unlike a PDO, the SDO must follow a particular communication protocol. Each message is composed of 8 bytes :

- Domain Protocol (Byte 0) : Defines the command (Upload, Download,….).
- Index - 16 bits (Bytes 1 and 2) : Defines the dictionary address of the object.
- Sub-index - 8 bits (Byte 3) : Defines the element of the selected object.
- Parameter (Bytes 4 to 7) : Defines the value of the parameter, read or written.

The network manager has a simplified mode for starting up the network. Network configuration is not required in all cases. The default parameter configuration is sufficient in many cases. If the user wants to optimise the CANopen network or increase its functionality, he can modify these parameters. In CANopen networks all devices have the same rights and data exchange is directly regulated between each participating device.

The profile of a device defines the parameters necessary for communication. The contents of this profile are specified by the device manufacturer. Devices with the same profile are directly interchangeable. Most parameters are described by the manufacturer. The profile may also contain empty slots for future extensions to the functionality by the manufacturer.

In most master/slave buses, the efficiency of the master determines the behaviour of the entire network. Moreover, slaves cannot communicate directly with each other. Such characteristics increase the number of transmission errors. CANopen eliminates all of these disadvantages. The timing characteristic can be specified individually for each task of the participating devices. So the entire communication system does not need to have the same efficiency if this is only required by certain devices. Moreover, an automatic task can be separated for each device. Thus the performance available to the network manager can be used in an optimised way and can be increased at any time by adding new devices.

10-2-2- Dictionary

A) CANopen dictionary

The drive can only use the mode SDO to allow reading from and writing to the parameters and variables.

The dictionary contains the various parameters and variables of the drive.

See the contents of the file ..\DPL\DATA\ Modbus and CANopen.xls (Opened preferably using Excel)

Parameter list of MD series								
OS Version : 1.0.3.0								
					Lecture CANOpen		Ecriture CANOpen	
Adress	Modbus Adr	Name	Decription	Size	Read Index	Read SubIndex	Write Index	Write SubIndex
600	400601	_RESTART_DRIVE		1	24673	0	8192	0
601	400602	_PARAM_MODE	Mode	1	24673	0	24672	0
602	400603	_PARAM_DRIVE_MODEL	Modèle	1	25872	1	25872	1
603	400604	_PARAM_DRIVE_NODE	Node ID	1	25872	5	25872	5
604	400605	_PARAM_DRIVE_I_NOM	Courant nominal	2	25872	99	25872	99
606	400607	_PARAM_DRIVE_I_MAX	Courant max	2	25872	100	25872	100
608	400609	_PARAM_I_NOM	Courant nominal	2	24693	0	24693	0
610	400611	_PARAM_I_MAX	Courant max	1	24691	0	24691	0
611	400612	_PARAM_TORQUE_NOM	Couple nominal	2	24694	0	24694	0
613	400614	_PARAM_MAX_MOTOR_S	Vitesse maxi	1	24704	0	24704	0
614	400615	_PARAM_MOTOR_PAIR_I	Nombre de paire	1	24653	0	24653	0
615	400616	_PARAM_SENSOR_TYPE	Capteur de temp	1	25616	32	25616	32
616	400617	_PARAM_RESOLVER_PA	Nombre de paire	1	25616	17	0	0
617	400618	_PARAM_DEPHASAGE	Déphasage / mo	1	25616	16	25616	16
618	400619	_PARAM_EXCITATION	Calage excitatio	1	25616	18	25616	18
619	400620	_PARAM_CALAGE	Calage automati	1	25616	19	25616	19
620	400621	_PARAM_GAIN	Gain excitation	1	25616	20	25616	20
621	400622	_PARAM_FILTRE_RESOL	Retour résolveur	1	25872	16	25872	17
622	400623	_PARAM_FILTRE_RESOL	Consigne ana filt	1	25872	17	25872	17
623	400624	_PARAM_EMULE_CODEU	Emulation codeu	1	25872	32	25872	32
624	400625	_PARAM_CODEUR_RESO	Résolution codeı	2	25872	33	25872	33
626	400627	_PARAM_CODEUR_NUMI	Numérateur	1	25872	34	25872	34
627	400628	_PARAM_CODEUR_DIVIS	Diviseur	1	25872	35	25872	35
628	400629	_PARAM_INPUT_INVERT	Inversion des ent	1	25872	48	25872	48
629	400630	_PARAM_INPUT_FILTER_	Activation filtre	1	25872	49	25872	49
630	400631	_PARAM_INPUT_FILTER_	Période filtre	1	25872	50	25872	50

- Flag variables :

16 bits are exchanged at the same time in the form of an integer variable.

e.g. : Index 12288, Sub-index 0 corresponds to VF0 to VF15

- Byte variables :

2 bytes are exchanged at the same time in the form of an integer variable.

e.g. : Index 12544, Sub-index 0 corresponds to VB0 to VB1

- Integer variables :

The type exchanged is the same.

- Long-integer variables :

The type exchanged is the same.

- Real variables :

The values sent must correspond exactly with the units and the number of decimal places (precision) parameter set in the software using menu Options / Language DPL / Compiler.

Example : Precision parameter of 0.01

Units : mm

In order to load 100.5mm in variable VR0

Index 13312, Sub-index 0, Value 10050

WriteParam (13312,0) = 10050

VR1 = ReadParam (13312,0) 'is equivalent to VR1 = VR0

See : WRITEPARAM, READPARAM

10-3- MODbus

10-3-1- Definition

A) Introduction

MODBUS is a master/slave protocol used mainly in industrial applications. It allows supervisory equipment (Human Machine Interface, Supervisory Control and Data Acquisition), to communicate with various industrial devises (Programmable Logic Controllers, sensors, etc.).

This protocol functions using requests. These messages can be transmitted on a serial link such as RS232, RS422 or RS485.

To distinguish one slave from another each piece of equipment is given an address (Unit ID). Using this number, only the slave concerned will answer a request from the master.

The drive operates the protocol MODBUS RTU slave.

The serial link format is 8 data bits, 1 stop bit, no parity.

The transmission speed can be up to 57600 baud.

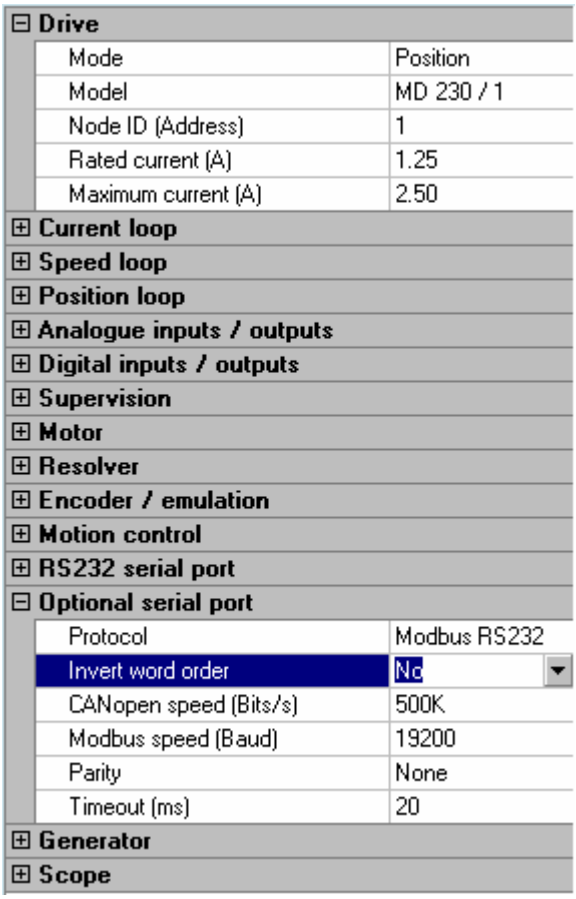
Functions for reading words (function n°3 or 4) and writing words (function n°16) are recognized by the drive.

B) Variables coded as 2 words

Drive parameters as well as some variables are coded as 2 words (32bits). As indicated in the Modbus standard, a double word has the following form :

Address :	Word :
n	MSW
n+1	LSW

The parameter « Invert word order » accessible in the parameter group Optional Serial Link allows the inversion of the coding of the double word for the variables type long and real.



The inversion has no effect on the drive parameters, which are always coded in accordance with the standard most significant, least significant.

Liaison :	System liaison	inversion parameter	VR & VL Coding version	parameters Coding version
RS232 serial port (Connector X1)	Enable	X	No	No
RS232 serial port (Connector X1)	Disable	No	No	No
RS232 serial port (Connector X1)	Disable	Yes	Yes	No
Optionnal serial port (Connector X4)	X	No	No	No
Optionnal serial port (Connector X4)	X	Yes	Yes	No

* X : don't care

If Invert Order = NO ⇒ Address n : most significant

Address n+1 : least significant

If Invert Order = YES ⇒ Address n : least significant

Address n+1 : most significant

10-3-2- MODBus dictionary

A) MODBus dictionary

The dictionary contains the various parameters and variables of the drive.

See the contents of the file ..\DPL\DATA\ Modbus and CANopen.xls (Opened preferably using Excel)

- Parameters are accessible between addresses 600 and 900
- Flag variables are accessible between addresses 57344 and 57359
- Byte variables are accessible between addresses 57360 and 57487
- Integer variables are accessible between addresses 57388 and 57743
- Long-integer variables are accessible between addresses 57744 and 58254
- Real variables are accessible between addresses 58256 and 58767

Index

H

I

L

M

N

O

P

R

S

T

V

W

X