Motion Drive

Digital drive for Brushless motors
IMD Series

User manual

Read manual before installing and follow all instructions 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

Read this manual first before installing the drive, non-observance may result in damage to property and in personal injuries.

Only suitable qualified personnel should undertake the mounting, installation, operation and maintenance of this equipment. The general set-up and safety regulations for work on power installations (e.g. DIN, VDE, EN, IEC or other national and international regulations) must be complied with.

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

The safety regulations are :

• 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 have gone to great lengths to ensure this documentation is correct and complete. However, since it is not possible to produce an absolutely error-free text. No responsibility will be assumed by SERAD for any damage caused by using this documentation and software.

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

1-2- IMD series drive description

1-2-1- General:

The IMD 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 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 :	230V to 400V AC ±10% three phase or 230V AC ±10% single phase
Auxiliary supply :	24 V DC ±10%, 0.4A typical (0.7A max if all options fitted)
Supply filter :	Integral
Switching frequency :	6.67 kHz sine-wave PWM
DC bus voltage :	310V to 680V
Braking resistance :	Integral : 75 ohms 60W Possibility to add an external resistor : Min value Max. cont. power Imp power 60Ω 5kW 10kW
Protection :	Short circuit between phases, phase to earth, over current, I2t Over voltage, under voltage Motor feedback fault
Motor feedback :	Resolver SinCos encoder Hiperface (option)
Master encoder :	Incremental encoder Absolute encoder SSI SinCos encoder Hiperface (option) Virtual
Encoder emulation :	Incremental : A, /A, B, /B, Z, /Z 1 to 100 000 points per rev
Diagnostic :	STATUS display
Communication :	RS 232 MODBUS RTU IMDBUS : for master/slave application CANopen (option): DS 402, SDO, PDO, master or slave PROFIBUS DP* (option) SERCOS 16Mb* (option)

Digital inputs :	4 inputs (with 2 fast inputs I3 and I4) 12 additional inputs with expansion module (with 2 fast inputs I15 and I16) Type: PNP, 24V DC, 8mA per input and 15mA per fast input Logic 0: Between 0 and 5 V Logic 1: Between 8 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, 500mA 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 : 16 bits on Input 1 12 bits on input 2
Analogue output :	1 output : Output voltage : ±10 V Maximum current : 5 mA Resolution : 8 bits
Architecture :	Processor :150 MHz DSP and 100 000 gates FPGA FLASH memory for programs and parameters RAM memory for data FRAM memory for variables Real-time, multi-tasking kernel
Control loops :	Current loop : 75 µs Speed loop : 150 µs Position loop : 150µs
Operating modes :	Torque mode Speed mode Position mode Stepper Mode (pulse input, direction) Motion functions (absolute, relative and infinite movements, S profile) Advanced motion functions (gearbox, CAM profiles, CAMBOX functions, triggered movement)
Operating temperature :	0 to 40°C
Storage temperature :	-10 to 70°C
Degree of protection :	IP 20
Weight	3.6 kg

Drive	Rated current	Peak current (2s)	Rated power	Dimensions w x h x d
IMD / 1	1.25 A rms	2.5 A rms	0.7 kVA	72 x 293 x 233
IMD / 2	2.5 A rms	5 A rms	1.4 kVA	72 x 293 x 233
IMD / 5	5 A rms	10 A rms	2.8 kVA	72 x 293 x 233
IMD / 10	10 A rms	20 A rms	5.6 kVA	72 x 293 x 233

1-3- iDPL software

1-3-1- General:

The iDPL 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: speed, current, torque, position
- Saving and printing 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 without being connected to a drive
- On-line help for each window

1-3-3- iDPL programming language:

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

The pseudo-basic language, iDPL, 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 badly earthed connection can damage electronic drive components.

↳ 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.

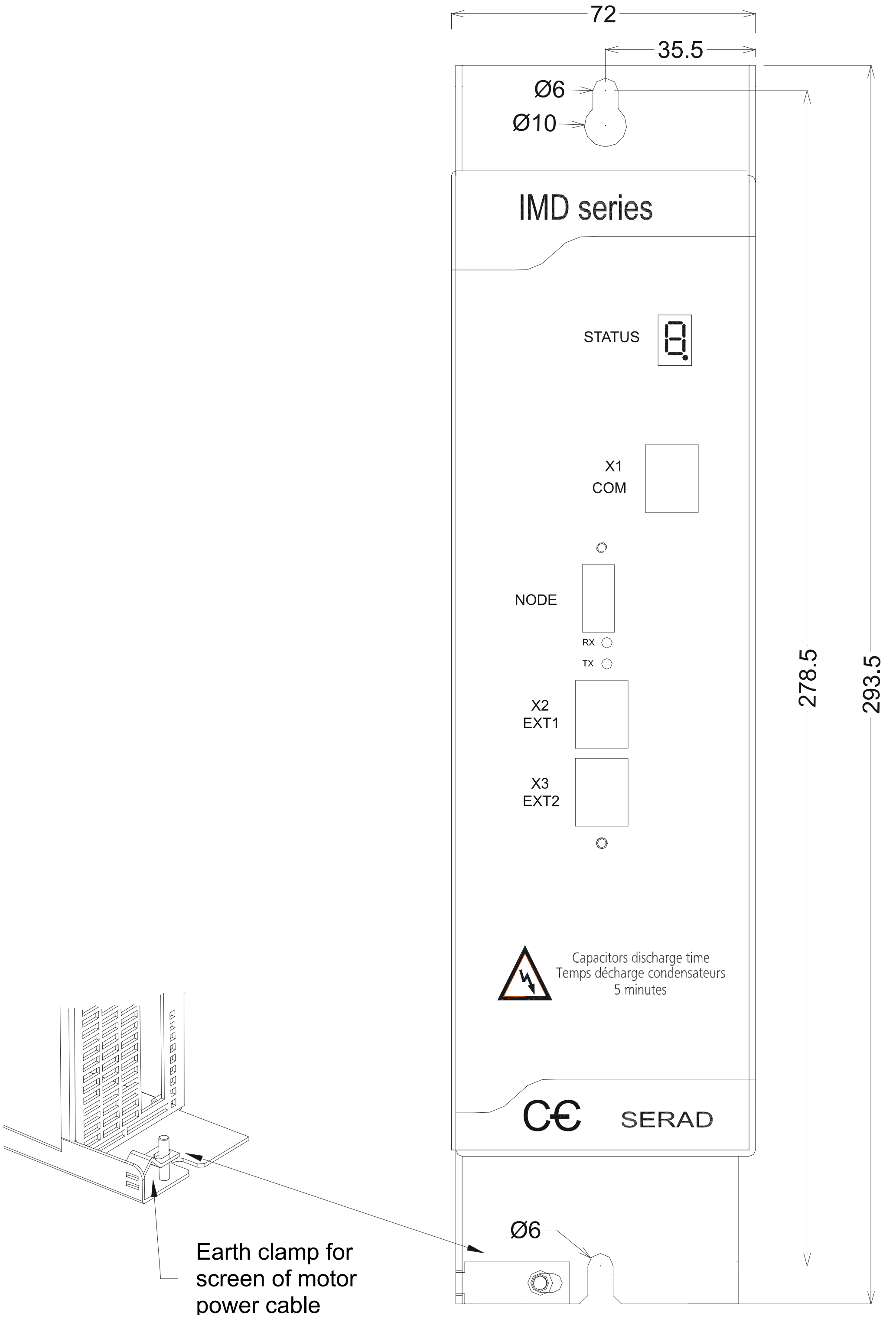
↳ In the case of axis over-travel, the over-travel limit switches must be connected to the limit inputs or in series with the emergency stop loop. It is also recommended to use the software limits.

↳ 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.

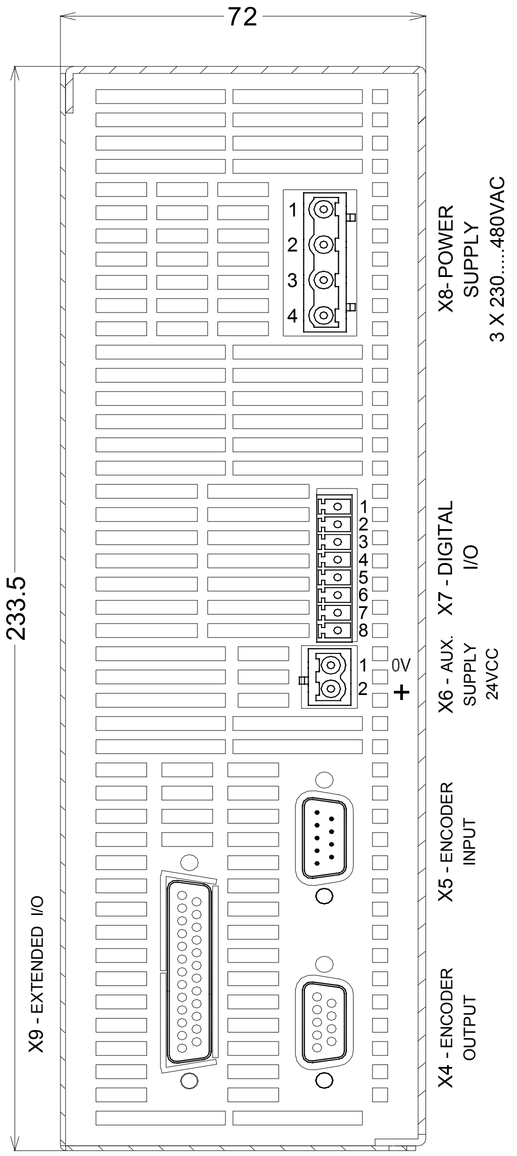
↳ If the drive contains an application program developed using iDPL, connect a signal ‘Cabinet supplies OK’ to one of the digital inputs and monitor it in a non-blocking safety task. On detection of an excess following error the drive will be put in open loop mode and the drive ready relay will be opened. 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	EXT1	Extension: Optional communications ports
X3	EXT2	Extension: Optional communications ports

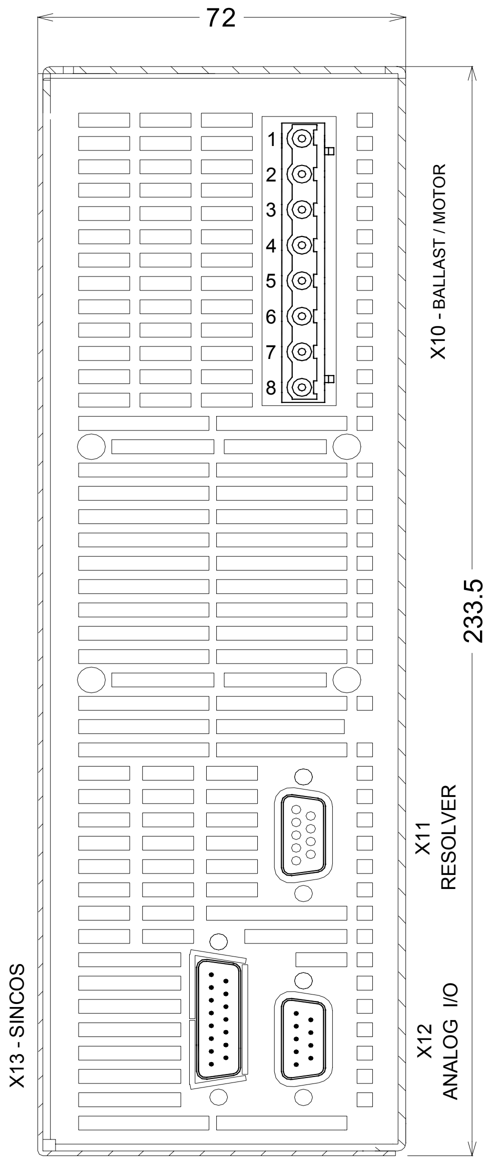
2-3- Top view



X4	ENCODER	OUTPUT	Multifunction encoder output
X5	ENCODER	INPUT	Multifunction encoder input
X6	24Vdc		Auxiliary 24V DC supply
X7	I/O		Digital I/O
X8	POWER SUPPLY		Single / Three-phase power supply
X9	EXT I/O		Option: I/O expansion board

The voltage on connector X8 can reach 480V!

2-4- Bottom view



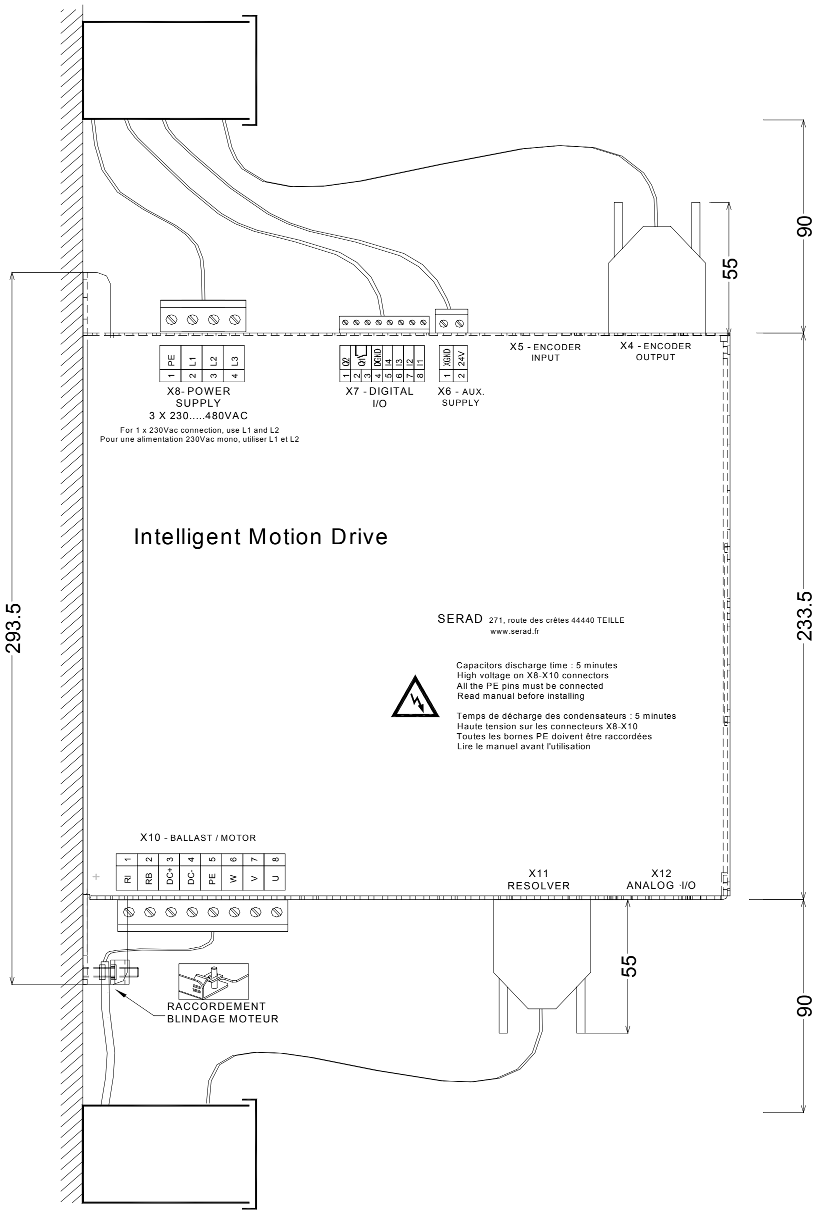
X10	RB / MOTOR	External braking resistor and motor supply
X11	FEEDBACK	Motor position feedback (resolver / encoder)
X12	ANALOG	Analogue I/O
X 13	SINCOS	Motor position feedback (if SINCOS encoder is used)

Care must be taken when making connections to connector X10. An incorrect connection can seriously damage the drive. Dangerous voltages are present on X10 (900V).

Wait at least 5 minutes to allow the capacitors to discharge before removing the connector.

2-5- Mounting

Several drives can be mounted side-by-side provided that enough space (at least 20mm) is left to ensure good natural convection. Leave a space greater than 90mm over and under the drives to allow for the various connectors and cables to be fitted.

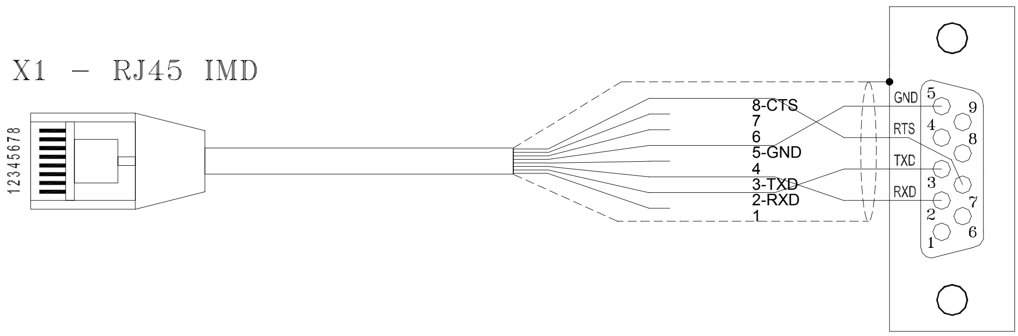


2-6- Connector pin assignments

2-6-1- X1: RJ45 serial port for downloading programs and parameters.

N°	Name	Type	Description
1			
2	RXD	Inp	Receive data
3	TXD	Out	Transmit data
4			
5	GND		0V
6			
7			
8	CTS	Inp	Clear to send
	SHIELD		Connect the shield to the shell of the connector

9 way SUBD socket



2-6-2- X2 & X3: Extension: Optional communications port

N°	Module RS 232	Module RS 422	Module RS 485	Module CANopen
1				
2	RXD	RX+		
3	TXD	RX-		
4				
5	GND	GND	GND	GND
6				
7		TX-	TRX-	CAN_L
8		TX+	TRX+	CAN_H
	Connect the shield to the shell of the connector			

- X2 and X3 are identical and have the same connections. This makes it easier to connect several drives to a network.
- Node Address : For RS422, RS485 and CANopen, the NodeID corresponds to the five firstly dipswitchs + 1

  Ex.: dipswitchs: 1 -> ON, 2 -> OFF, 3 -> ON, 4 -> OFF, 5 -> OFF

  Dipswitchs value = 1 + 4 = 5

  NodeID = 5 + 1 = 6

- Put on Dipswitch 6 to activate terminal resistor (120Ω).

  RS232 communication allows communication only with 1 device (ex: 1 PLC and 1 IMD drive).

2-6-3- X4: Multifunction encoder output:

- Encoder emulation output
- IMDbus output

The choice of the output is made in the iDPL software in the Multifunction encoder output window.

Connector : SUBD 9 way female

N°	Name	Type	Encoder emulation	IMDbus
1	A	Out	Channel A	Data
2	/A	Out	Channel A inverted	/Data
3	B	Out	Channel B	Clock
4	/B	Out	Channel B inverted	/Clock
5	Z	Out	Zero marker	NC
6	/Z	Out	Zero marker inverted	NC
7				
8	GND		0V	0V
9				
	SHIELD		**Connect the shield to the shell of the connector**	

NC (Not connected): It is forbidden to connect this pin.

2-6-4- X5: Multifunction encoder input:

- Incremental encoder input
- SSI encoder input
- Stepper input
- IMDbus input

TTL 5V encoder (0-5V, differential)

The choice of the input is made in the iDPL software in the Multifunction encoder input window.

Connector : SUBD 9 way male

N°	Name	Type	Incremental encoder	Codeur SSI	Stepper	IMDbus
1	A	Inp	Channel A	Data	Direction	Data
2	/A	Inp	Channel A inverted	/Data	/Direction	/Data
3	B	Inp	Channel B	NC	Pulse	Clock
4	/B	Inp	Channel B inverted	NC	/Pulse	/Clock
5	Z	I/O	Zero marker	Clock	NC	NC
6	/Z	I/O	Zero marker inverted	/Clock	NC	NC
7	+5Vdc	Out	Supply for external encoder, 100 mA max.*	NC	NC	NC
8	GND		0V	0V	0V	0V
9		Inp	NC	SSI selection : Connect pins 8 and 9	NC	NC
	SHIELD		Connect the shield to the shell of the connector			

* If the feedback is SINCOS then do not use the 5V power supply (pin 7 of connector X5) but an external power supply.

NC (Not connected): It is forbidden to connect this pin.

2-6-5- X6: 24V dc supply

Connector: Removable 2 way, 5.08mm pitch

N°	Name	Type	Description
1	XGND		0V
2	24Vdc	Inp	Control card supply, backup motor position

2-6-6- X7: Digital I/O

Connector: Removable 8 way, 3.81mm pitch

N°	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	Inp	Input 4, programmable
6	I3	Inp	Input 3, programmable
7	I2	Inp	Input 2, programmable
8	I1	Inp	Input 1, programmable:standard function ENABLE

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

2-6-7- X8: High voltage supply

Connector: Removable 4 way, 7.62mm pitch

N°	Name	Type	Description
1	PE		Supply earth
2	L1	Inp	Supply L1 for 230V and 400V
3	L2	Inp	Neutral for 230V or supply L2 for 400V
4	L3	Inp	Supply L3 for 400V

Care must be taken when making connection to connector X8.

Wait for at least 5 minutes to allow the capacitors to discharge before remove the connector.

2-6-8- X9: Option: Expansion module, 12 inputs / 8 outputs

Connector: SUBD 25 way female

N°	Name	Type	Description
1	I5	Inp	Input 5, programmable
2	I6	Inp	Input 6, programmable
3	I7	Inp	Input 7, programmable
4	I8	Inp	Input 8, programmable
5	I9	Inp	Input 9, programmable
6	I10	Inp	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 24Vdc**	Inp	External supply, 24 V dc
13	IO 24Vdc**	Inp	External supply, 24 V dc
14	I11	Inp	Input 11, programmable
15	I12	Inp	Input 12, programmable
16	I13	Inp	Input 13, programmable
17	I14	Inp	Input 14, programmable
18	I15	Inp	Input 15, programmable
19	I16	Inp	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

2-6-9- X10: Motor armature

Connector: Removable 8 way, 7.62mm pitch

N°	Name	Type	Description
1	RI		Internal braking resistor *
2	RB		Braking resistor *
3	DC Bus +	Out	DC bus +
4	DC Bus -	Out	DC bus -
5	PE		Motor earth
6	W	Out	Motor phase W
7	V	Out	Motor phase V
8	U	Out	Motor phase U

The shielded motor cable must be connected directly to the terminals of the drive.

Connect the shield (on drive side) using the clamp provided (see Front view of the drive).

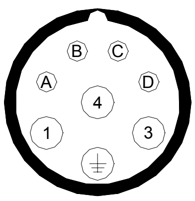
*Selection of the braking resistor:

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

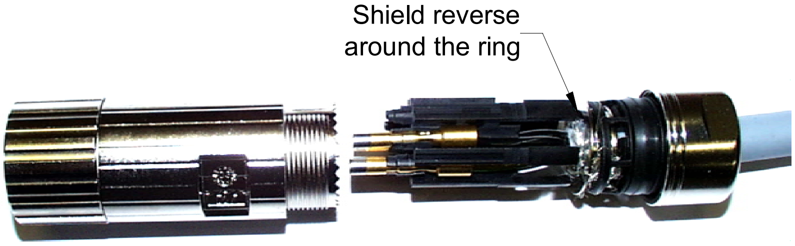
  Connect the external resistor between terminals 2 and 3

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

SERAD MOTOR



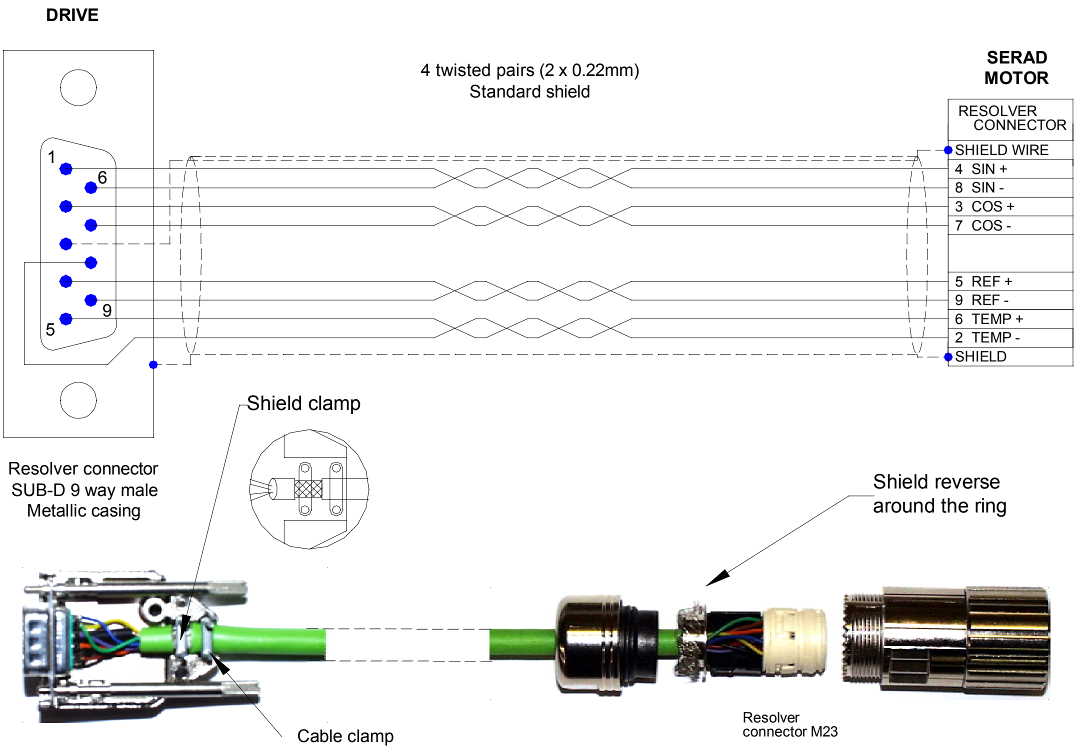
DESCRIPTION	
1	Phase U
4	Phase V
3	Phase W
2	Earth
C	Break +
D	Break -



2-6-10- X11: Motor position feedback (resolver)

Connector: SUBD 9 way female

N°	Name	Type	Description
1	S2	Inp	Sine Hi
2	S1	Inp	Cosine Hi
3	AGND		0V analogue
4	R1	Out	Reference Hi
5	°CM+	Inp	Motor temperature sensor Hi
6	S4	Inp	Sine Lo
7	S3	Inp	Cosine Lo
8	°CM-	Inp	Motor temperature sensor Lo
9	R2	Out	Reference Lo
⚠	**SHIELD**		**Connect the shield to the shell of the connector**



2-6-11- X12: Analogue I/O

Connector : SUBD 9 way male

N°	Name	Type	Description
1	IN2 -	Inp	Analogue input 2
2	IN2+	Inp	Analogue input 2 : assigned to torque limit
3	IN1-	Inp	Analogue input 1
4	IN1+	Inp	Analogue input 1 : assigned to speed or torque command
5	AGND		0V analogue
6	-12V	Out	-12V, 20 mA output
7	AGND		0V analogue
8	+12V	Out	+12V, 20 mA output
9	OUT	Out	Analogue output (function monitor)
⚠	**SHIELD**		**Connect the shield to the shell of the connector**

2-6-12- X13: Option: SinCos encoder input

Connector: SUBD 15 way male

N°	Name	Type	Description
1	°CM +	Inp	Motor temperature sensor Hi
2	AGND		0V analogue
3	/DATA	I/O	/DATA (In Dev*) /RS485 (HIPERFACE)
4	/CLK	Out	/CLOCK (In Dev*)
5	+5V	Out	+5V, 200 mA output (In Dev*)
6			
7	REFCOS	Inp	Cosine Hi
8	REFSIN	Inp	Sine Hi
9	°CM-	Inp	Motor temperature sensor Lo
10	+8,3V	Out	+8.3V, 150 mA output(HIPERFACE)
11	DATA	I/O	DATA (In Dev*) RS485 (HIPERFACE)
12	CLK	Out	CLOCK (In Dev*)
13			
14	COS	Inp	Cosine Lo
15	SIN	Inp	Sine Lo
⚠	**SHIELD**		**Connect the shield to the shell of the connector**

* In Dev - in development

2-7- Cables

We can supply all cables with connectors (standard, robotics ...), contacts us.

- RS 232 serial communication cable, X1:

  Screened cable, 4 core

  Connect the shield on each extremity, to the shell of the connector (RJ45 and SUBD).

- Encoder cable, X4/X5:

  Screened cable with 4 twisted pairs, 0.25 mm²

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

- Analogue cable, X12:

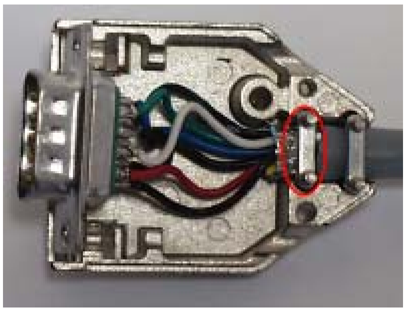
  Screened cable, 2 core, 0.25 mm² per analogue input.

  Connect the shield: on drive side to the screw provided (see 2-2 Front view) and on the other side to the shell equipment (ex. Motion controller …)

- Motor feedback cable (resolver), X11:

  Screened cable with 4 twisted pairs, 0.25 mm²

  Ground the shield of the feedback SUBD as shown below:



- Motor power cable, X10:

  Screened cable, 4 core, (+2 for a brake), 1.5 mm² for drives up to 8A otherwise use 2,5 mm²

  Connect the shield (on drive side) to the clamp provided (see Front view of the drive).

2-8- Connection diagrams / Protection

All connections must be made by qualified personnel. 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 4 of the connector X8).

Connect the motor earth to the drive (pin 5 of the connector X10) before applying any voltages.

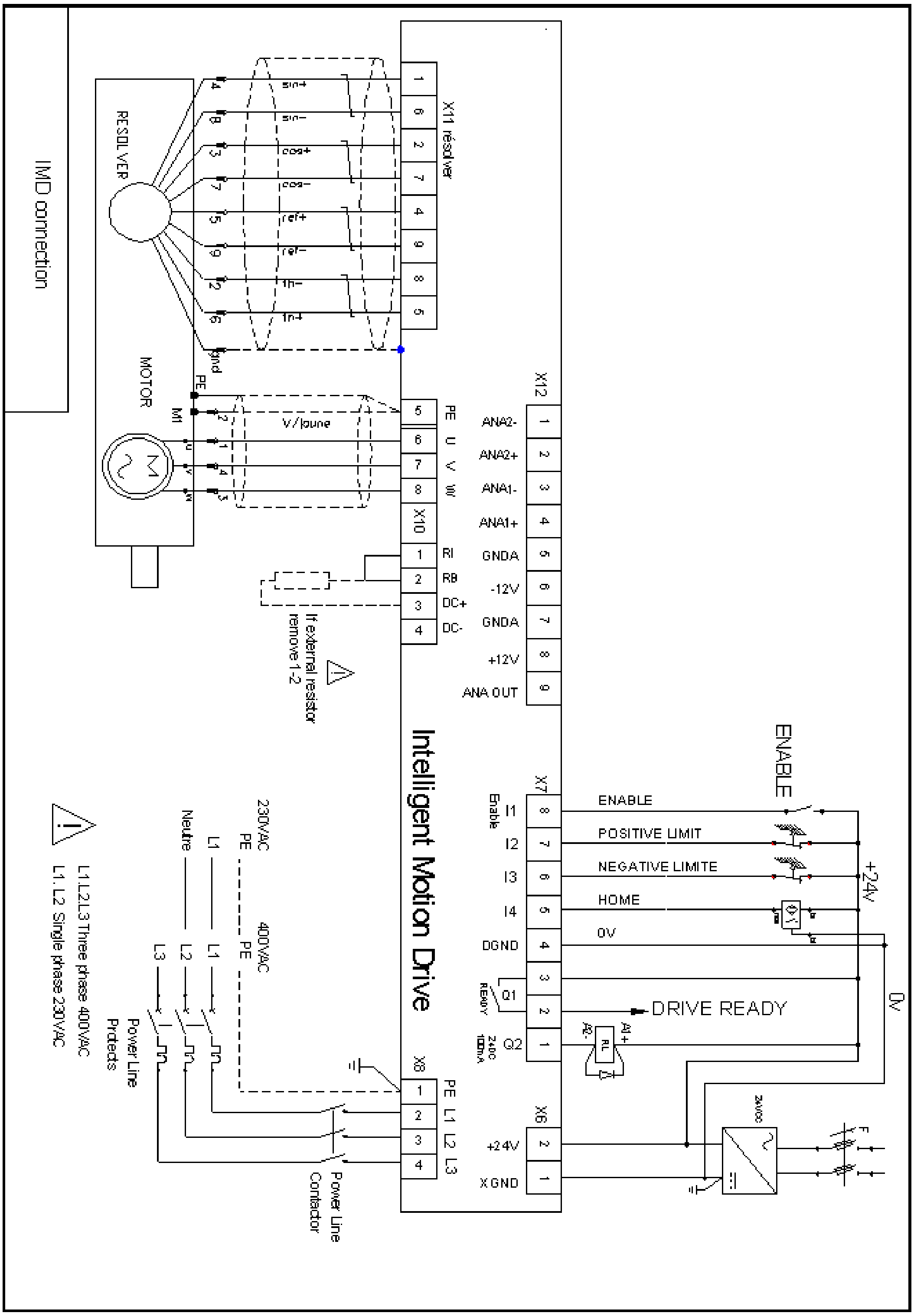
For the shielded cables, connect the screen to the chassis at each extremity via the shell of the connectors (for the SUBD) or the screws provided for this purpose (X7) in order to ensure an optimal equipotential.

Transient suppression measures should be taken on control panel components such as contactors (obligatory on brake) and relays using RC elements or diodes (e.g. 1N4007).

Drive	Input voltage	Maximal input current	Safety device: cutout curve C	Wire
IMD / 1	400V 3-phase	2.2A	10A max	1.5^2 mm
	230V 1-phase	3.5A	10A max	1.5^2 mm
IMD / 2	400V 3-phase	4.2A	10A max	1.5^2 mm
	230V 1-phase	7A	10A max	1.5^2 mm
IMD / 5	400V 3-phase	8.2A	10A max	1.5^2 mm
	230V 1-phase	14A	16A max	2.5^2 mm
IMD / 10	400V 3-phase	16A	20A max	2.5^2 mm

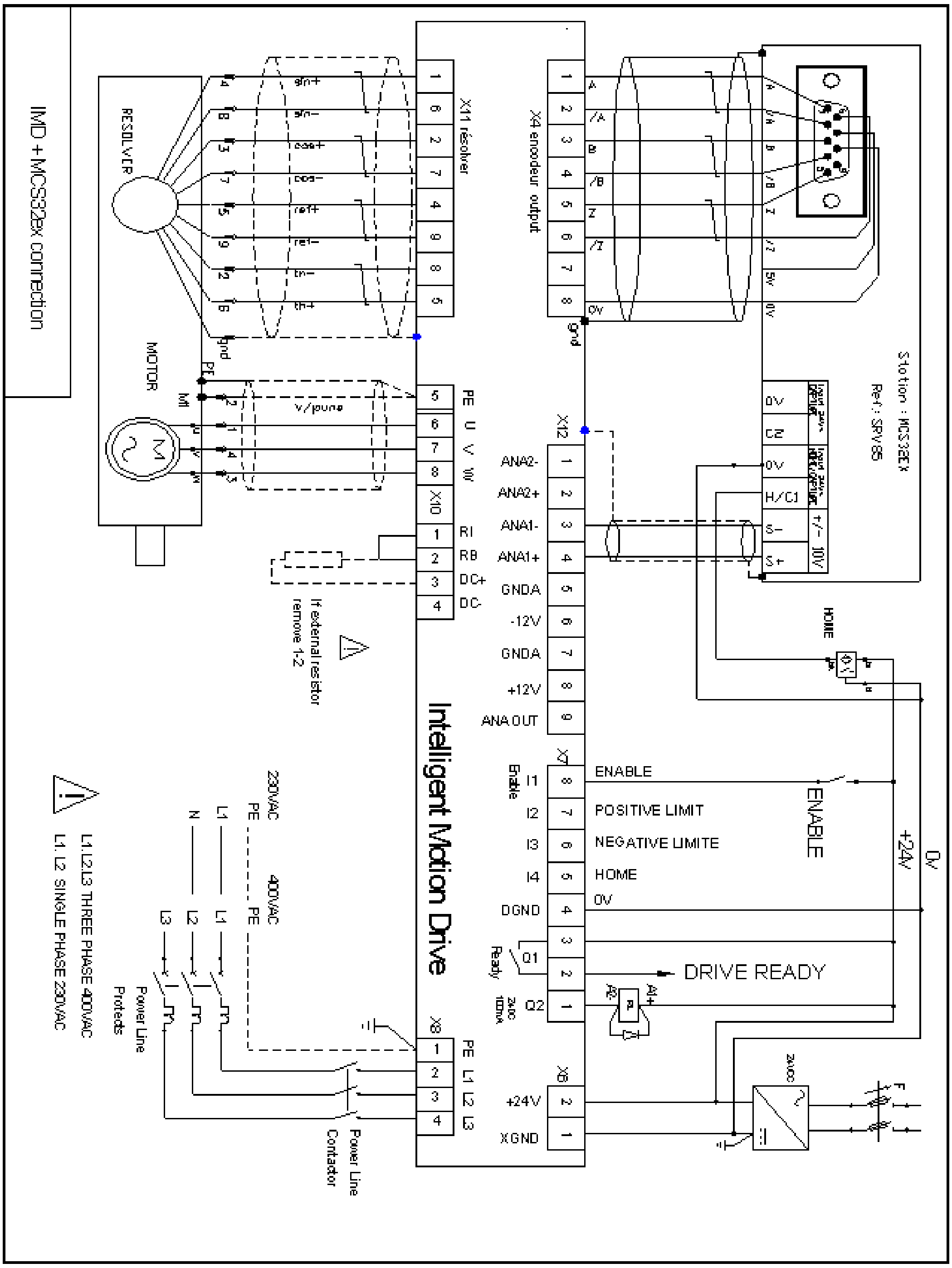
Caution: the in-rush current can reach 25A with a duration of 10ms.

A) Stand-alone drive



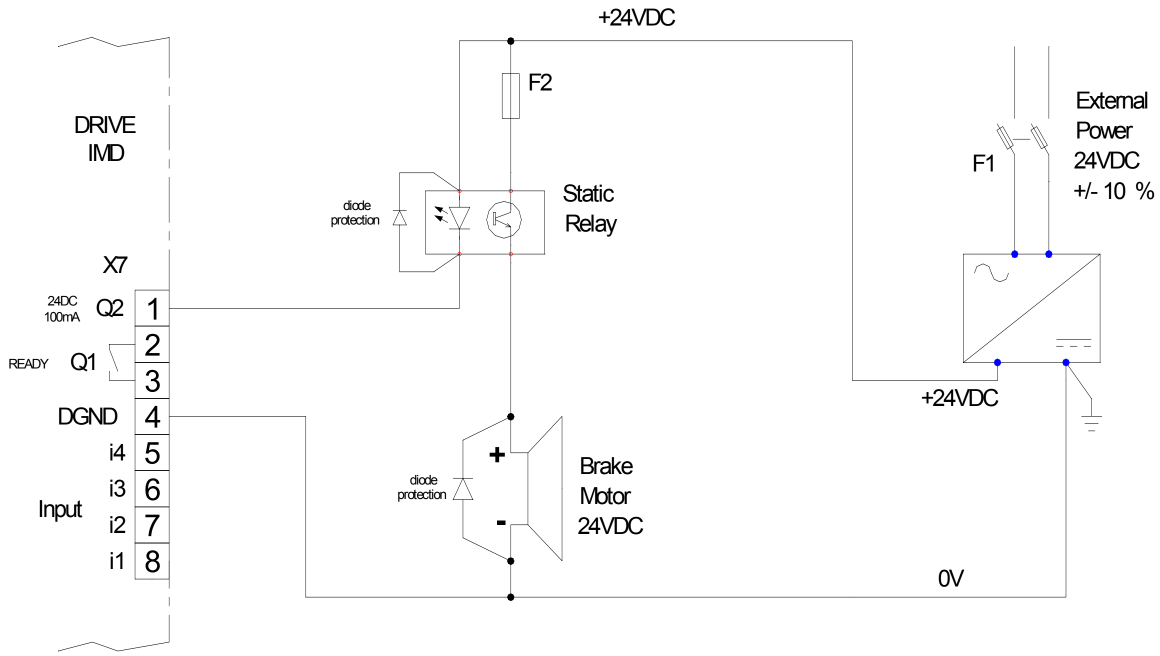
The output Q2 is NPN open collector, 100mA max. The load must be connected between Q2 and +24Vdc.

B) Drive controlled by a motion controller



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

C) Connecting a motor brake



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

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

It is obligatory to use the 2 protection diodes otherwise drive components can be damaged.

2-9- 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- iDPL software

3-1- iDPL software installation

3-1-1- System configuration

A) Minimum configuration:

⇒ Pentium II PC

⇒ 64M Byte RAM

⇒ Hard disk (35 M Bytes free)

⇒ Microsoft® Windows™ 98 SE, NT, 2000 and XP

⇒ CD-ROM (2X)

⇒ SVGA monitor

⇒ Mouse or other pointing device

B) Recommended configuration:

⇒ Pentium® II PC

⇒ 256M 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- iDPL installation procedure

The software package "Intelligent 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 progression is indicated with a bar graph.

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

3-1-3- Directories

The default installation folder for the software is:

C:\Program Files\SERAD\iDpl\

It contains 5 sub-directories:

- Data: containing the source files of the software.
- Help: containing the help files
- Lib: containing the various parameter files for the drive.
- Os: containing the drive operating system.
- Doc containing automatically generated documentation files (modbus.htm, EDS file...)

3-2- Presentation

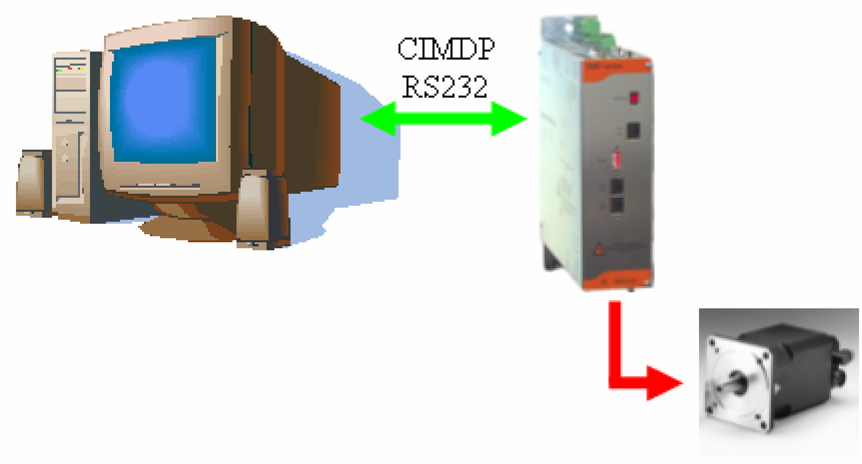
3-2-1- Communication methods

To communicate with the drive, you need the CIMDP cable that allows the drive to be connected to a PC.

When you connect a drive to a PC, all the drive parameters are transferred into the iDPL software

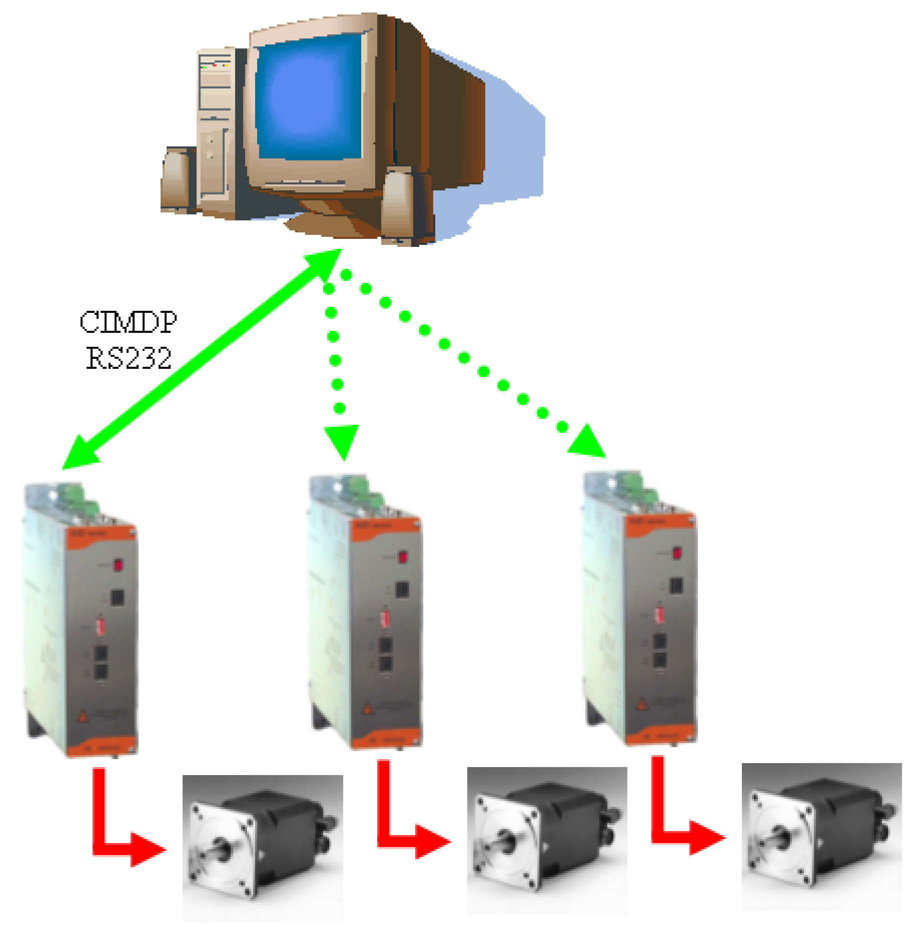
When you change a parameter in the iDPL software, you also change the drive parameters (but they are not saved if you restart drive).

A) Communication with one drive:



In iDPL software, create a new project with one drive.

B) Communication with several drives:

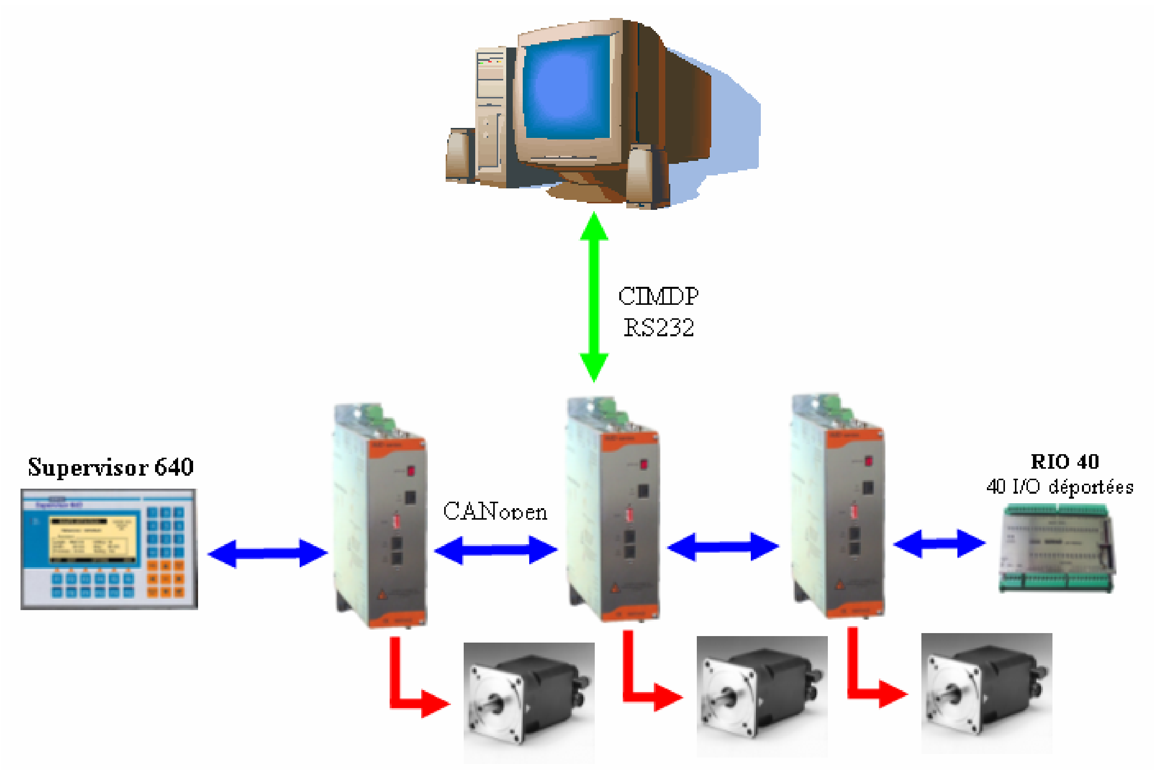


In iDPL software, create a new project with the drive number of the machine.

To connect to another drive, select a drive from the drive list in the iDPL software and then connect the CIMDP cable to the correct drive.

C) Multi drive communication:

Multi drive (several drives on a CAN network) allows you to program all drives without changing the CIMDP cable connection.



In the iDPL software, create a new project with the drive number of the machine.

To connect to another drive, select a drive in the drive list of iDPL software. The CIMDP cable can be connected to any drive.

PC uses node ID 1 so your CANopen network must start at address 2.

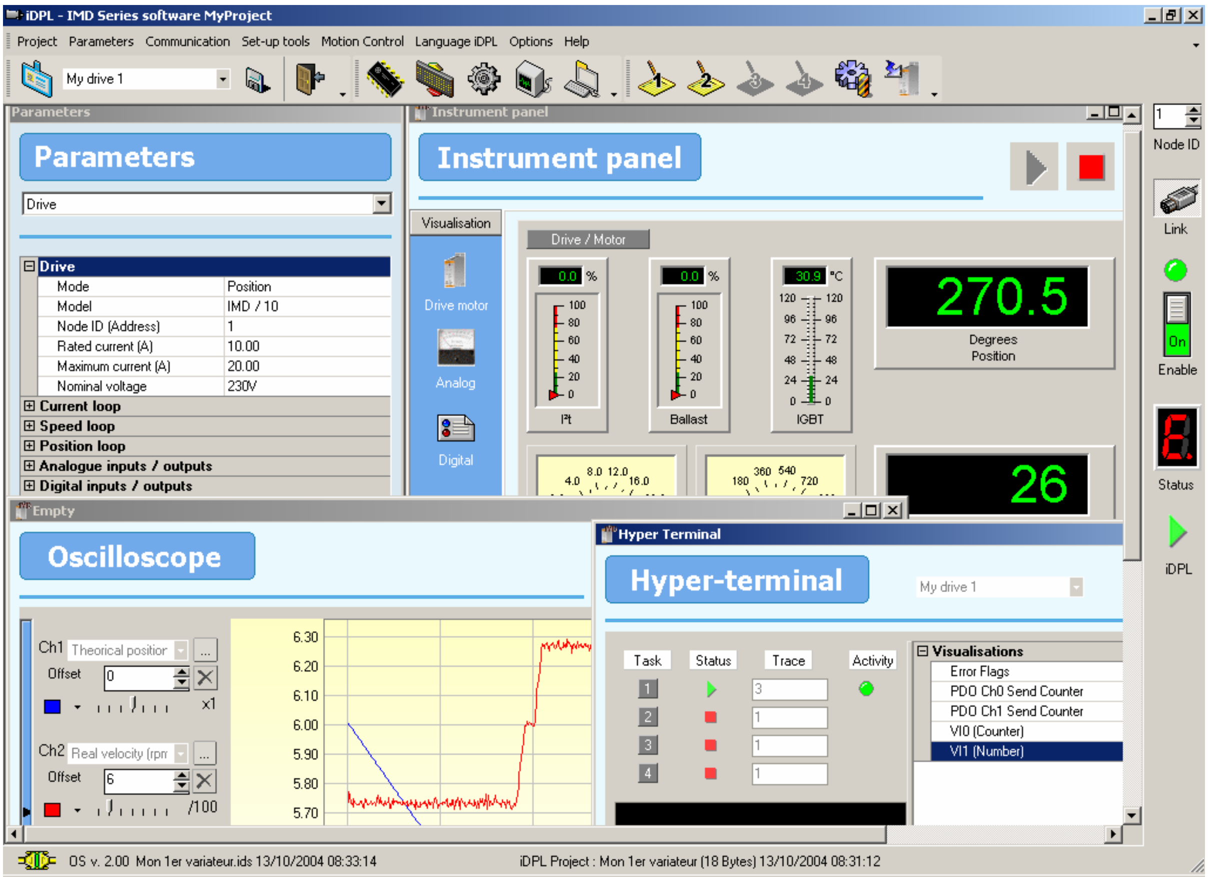
It is OBLIGATORY to be in system communication between PC and drives for multidrives project.

Default project is saved in the Project directory of the iDPL software.

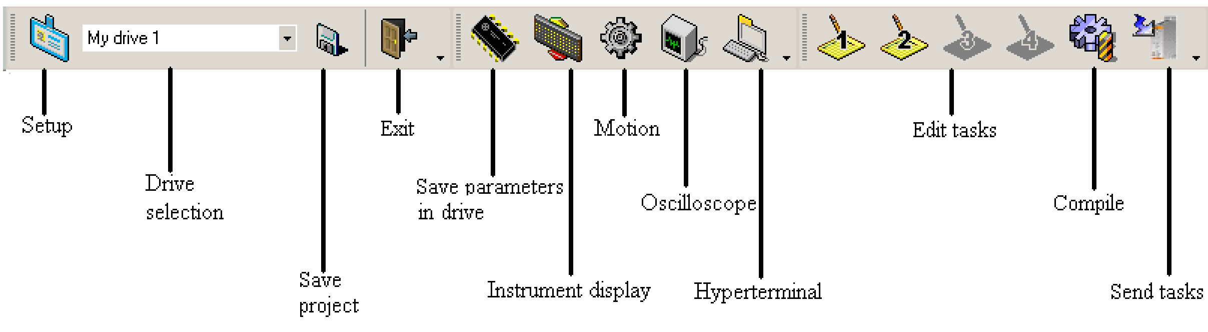
In off-line working, you must open a project and a parameter file.

3-2-2- Initial screen

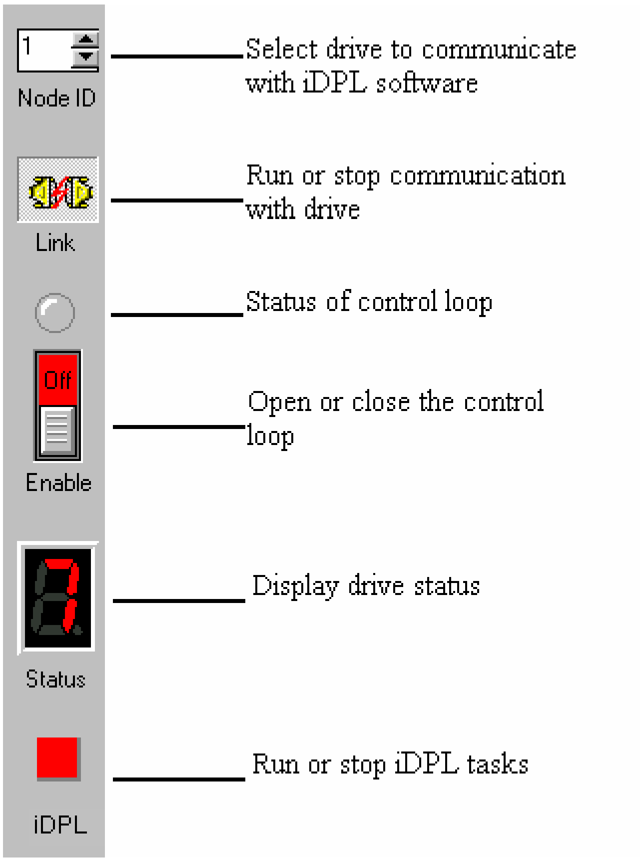
The iDPL software is characterized by a main window that contains a menu bar, icon bar and a number of selectable windows. The ability to have multiple windows allows the user to simultaneously view several aspects of the drive.



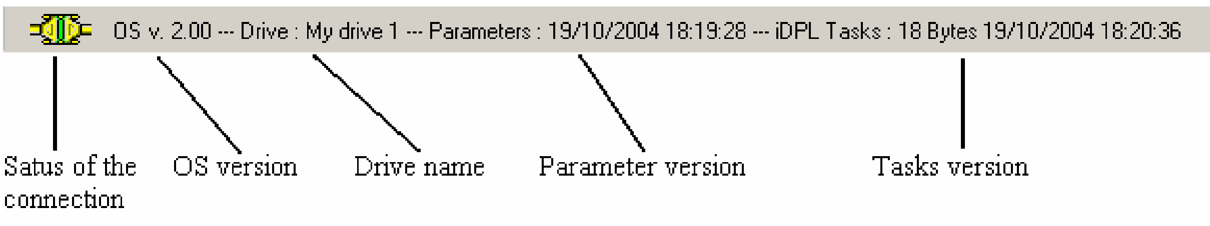
- Tool bar:



- Command bar :

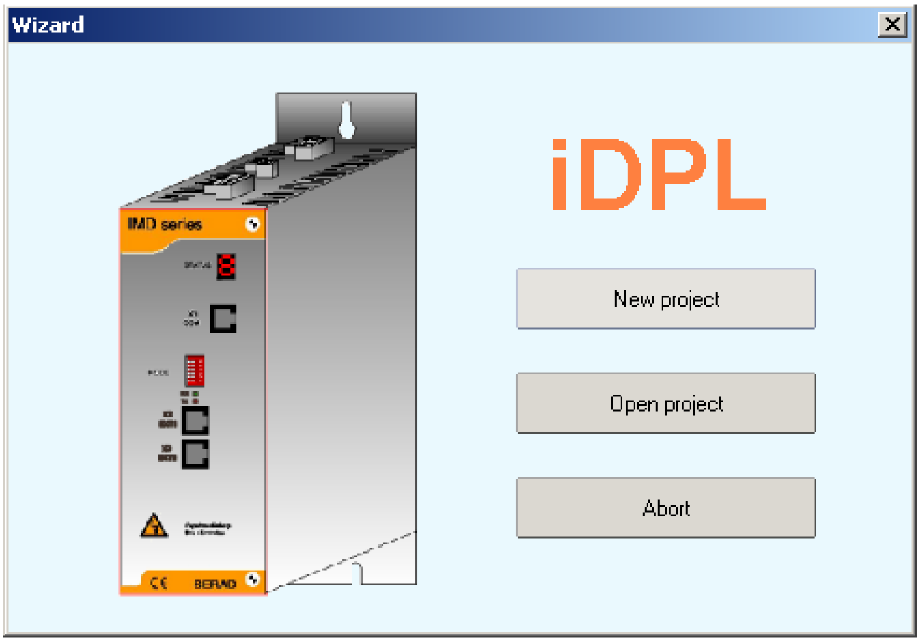


- State bar :

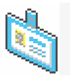


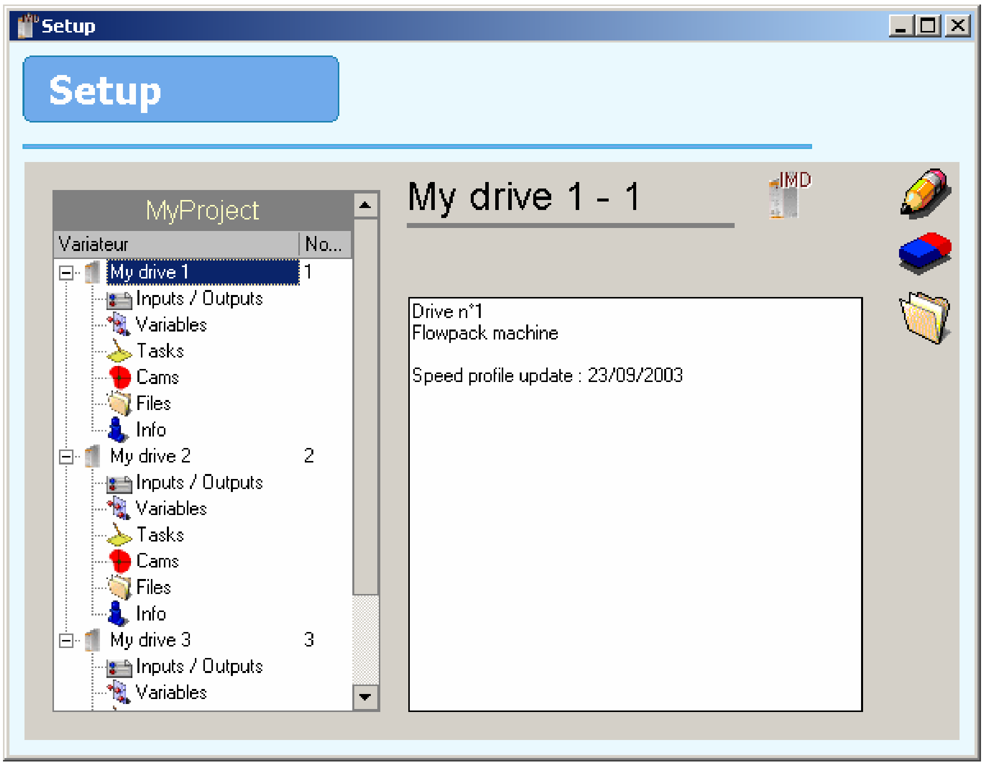
3-2-3- Project management

iDPL software starts with a wizard window:



It is obligatory to create and open a project to access a drive.

To access the project setup, click on  icon or choose **setup** in **project** menu.



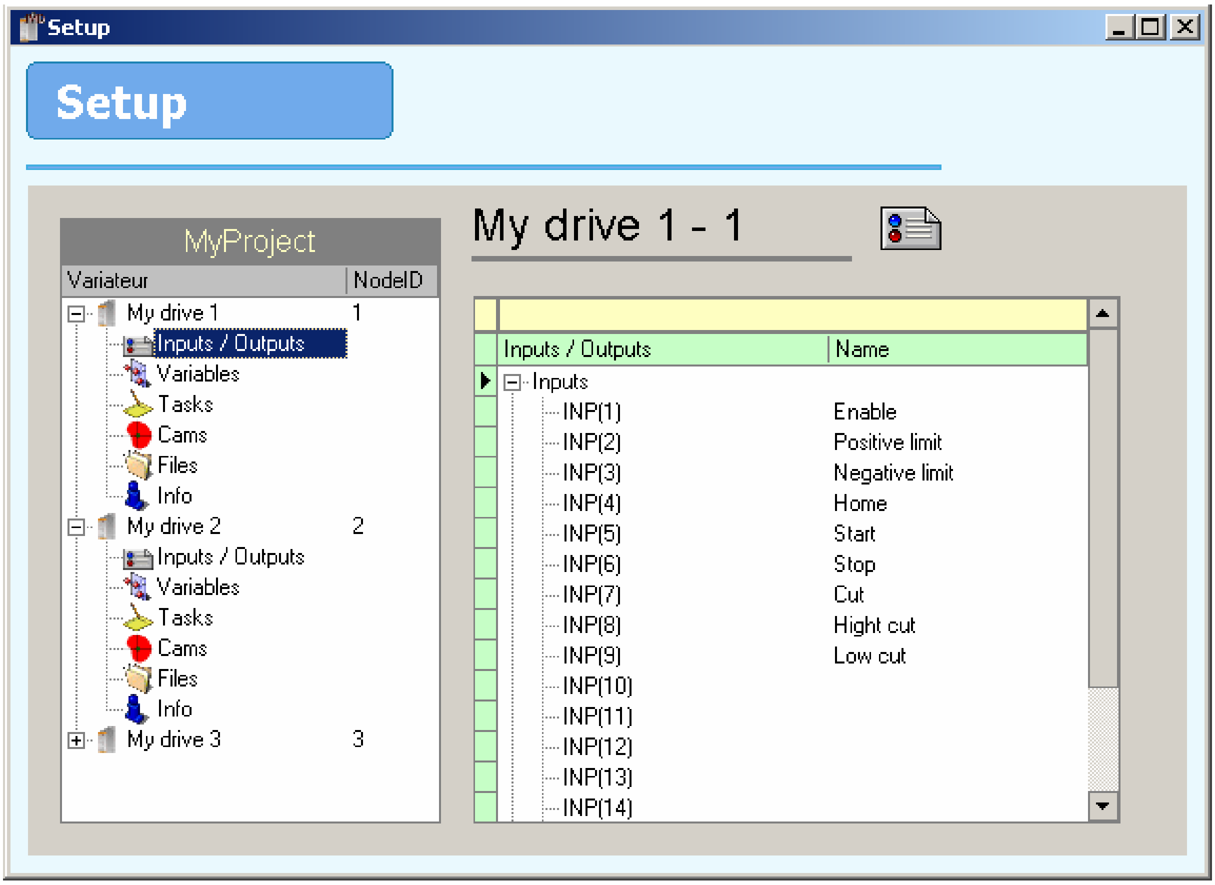
In this window, you can setup all drives of your project (parameters, I/O, variables, tasks, cams ...)

Double click on Node ID number to change it (must be the same as drive dipswitchs).

In the right area, programmer can let notes for next use.

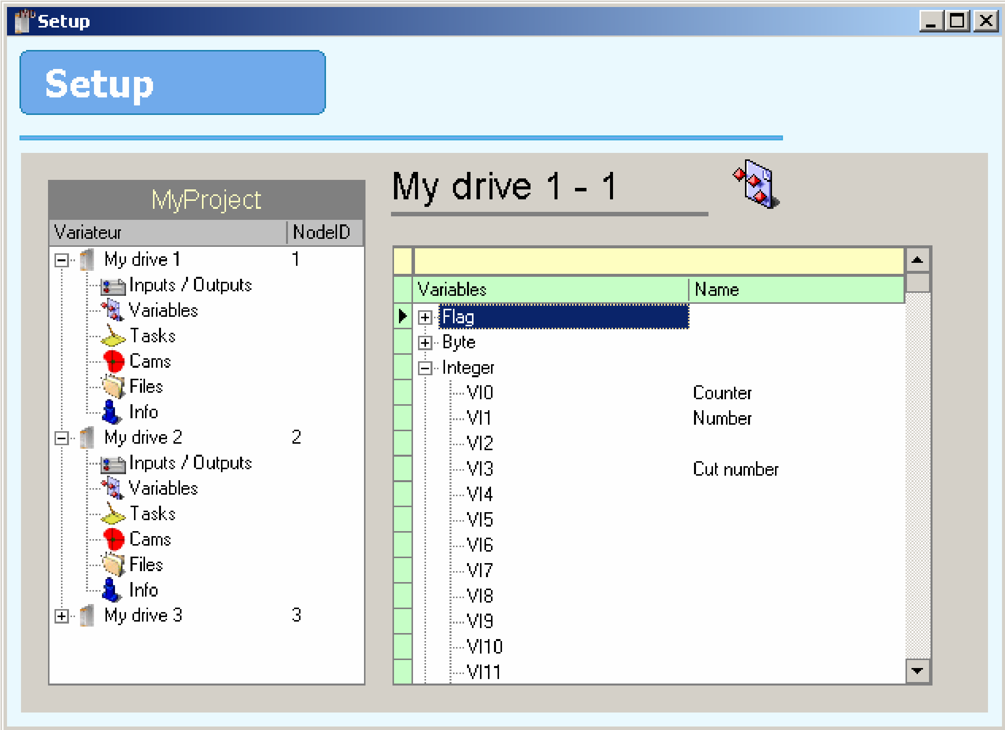
A project can have up to 127 drives.

A) I/O declaration:



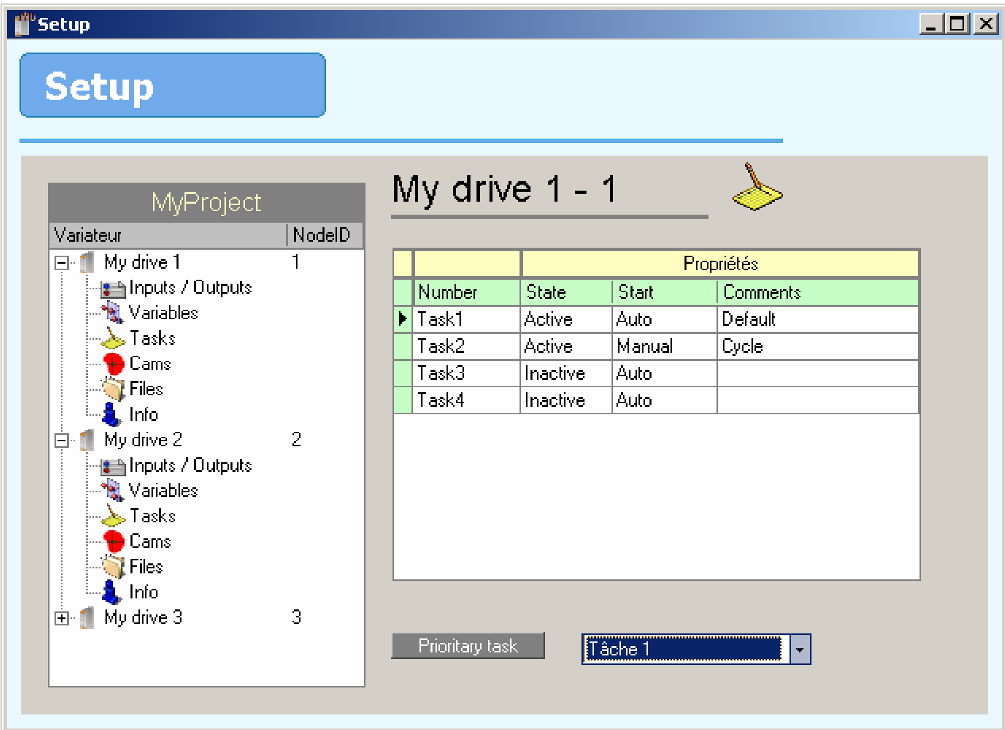
Allows I/O to be assigned names that can be used by the iDPL tasks.

B) Variable declaration :



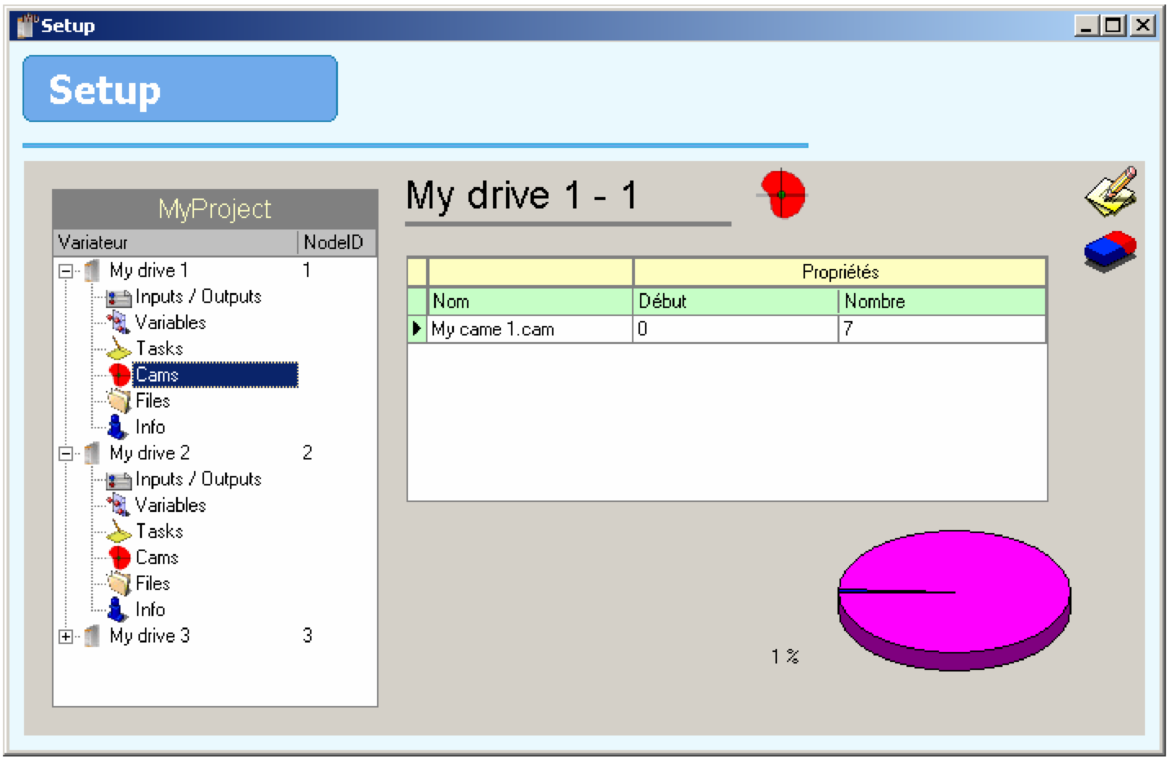
Allows variables to be assigned names that can be used by the iDPL tasks.

C) Task declaration :



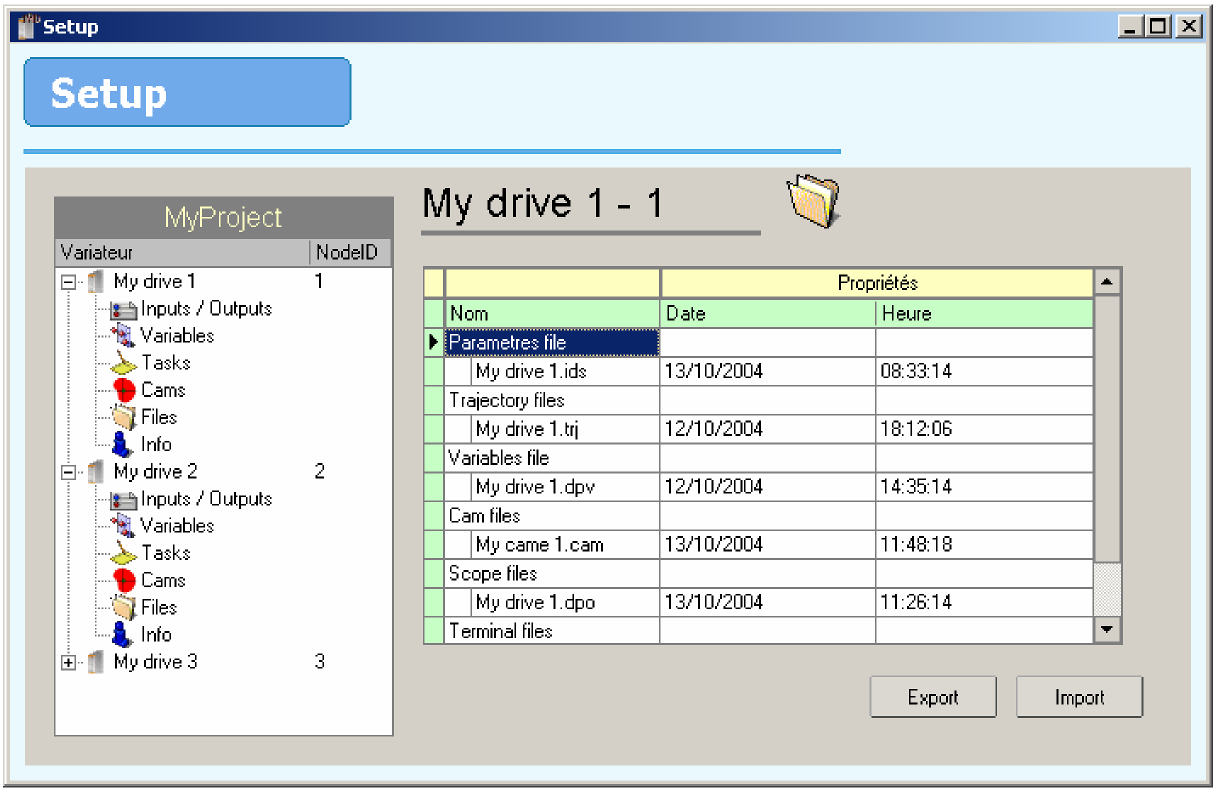
Allows the activation of tasks (at power-on or by run function) and defines a task priority.

D) Cam declaration :



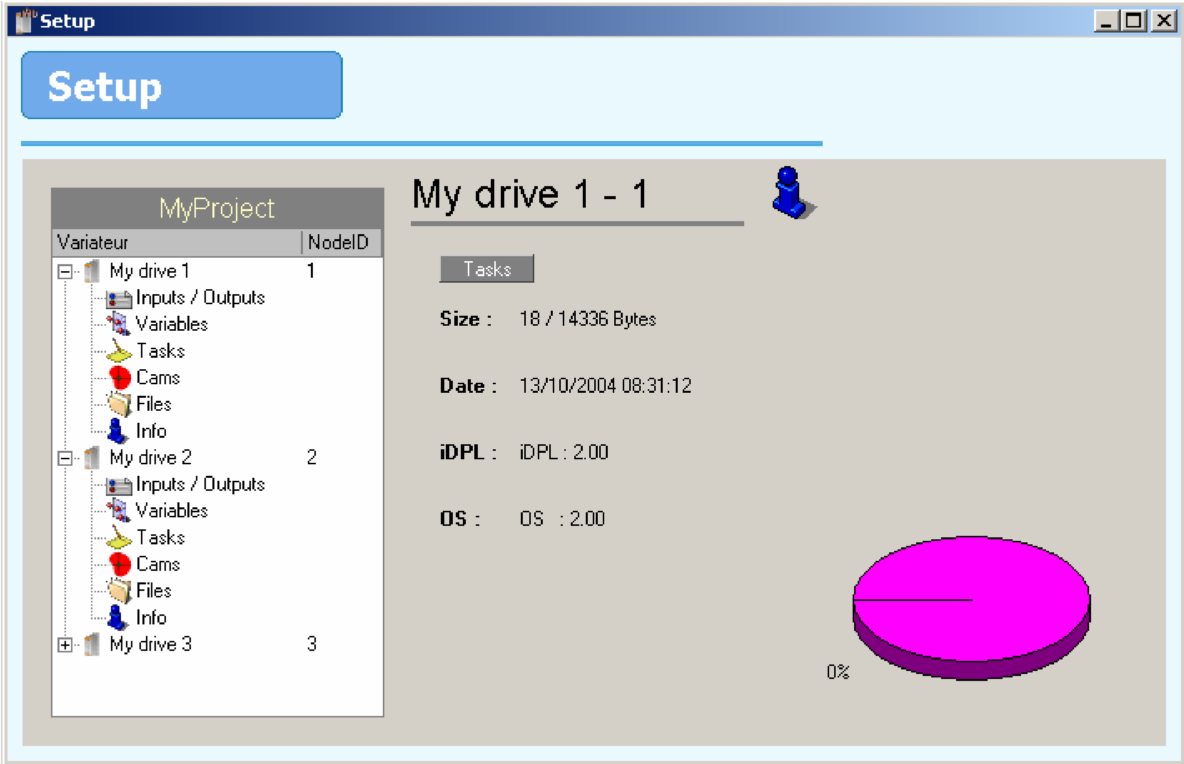
Allows the definition of cams in flash memory (a cam is defined by a starting position and a size).

E) Drive files information :



Shows all files for the drive and allows importing or exporting files from/to other projects.

F) Drive information :



Shows the OS and software version as well as the drive memory in use.

3-2-4- Project contents

A project comprises a file ProjectName.idw and a folder ProjectName.data.

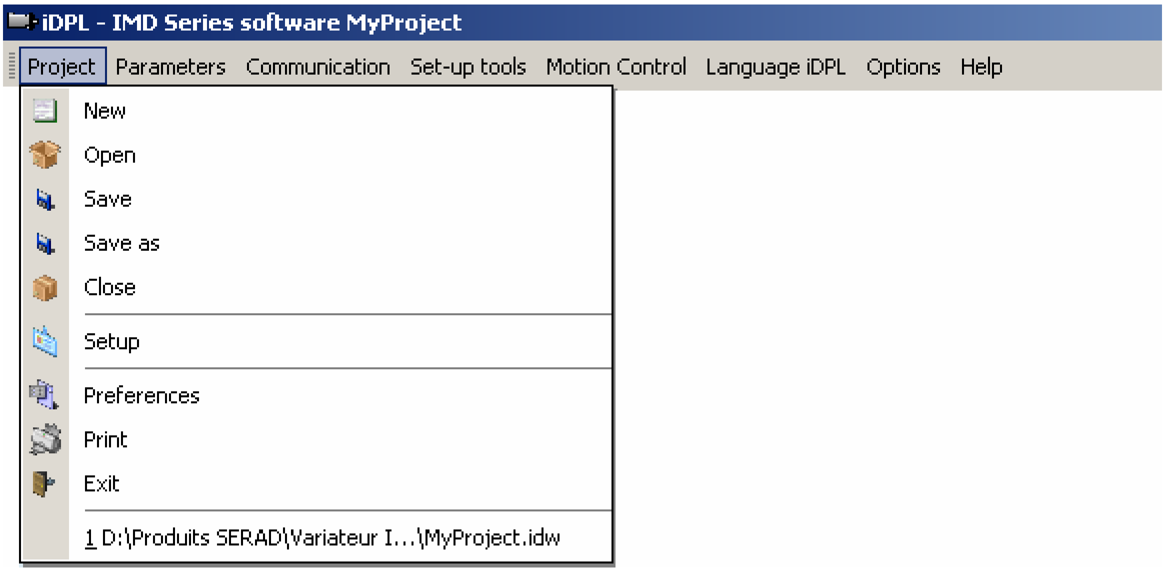
The folder contains:

- Files (DriveName.ids) containing the drive parameters in text format.
- Files (DriveName.idp) containing the drive information in text format.
  - I/O declaration
  - Variable declaration
  - Task declaration
- Folder (DriveName.data) containing the files:
  - Files (TaskX.dpl) containing the task code in text format.
  - A file (DriveName.dpi) containing information relating to the drive.
  - A file (DriveName.dpo) containing oscilloscope set-up relating to the drive

- A file (DriveName.dpv) containing a list of variables and their values.
- A file (DriveName.trj) containing trajectories relating to the drive.
- A folder (bin) containing the compiler output files and parameter files required by the drive
- Files (.dpt) containing hyper terminal setup
- Files (.cam) containing cam profile

3-3- Menus and icons

3-3-1- Project



A) New :

Icon : 

Action : Define a new project.

B) Open :

Icon : 

Action : Open an existing project.

C) Save :

Icon : 

Action : Save the entire contents of the project.

D) Save as :

Icon : 

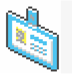
Action : Save the project under a different name. This command creates a file and a directory having the same name but with extensions .idw for the file and .data for the directory.

E) Close :

Icon : 

Action : Close the current project.

F) Setup :

Icon : 

Action : Setup the drives in the project (define I/O, variables, tasks of each drive)

See project management chapter

G) Preferences :

Icon : 

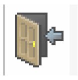
Action : Setup / alter the printing options (it is only possible to print in portrait mode).

H) Print :

Icon : 

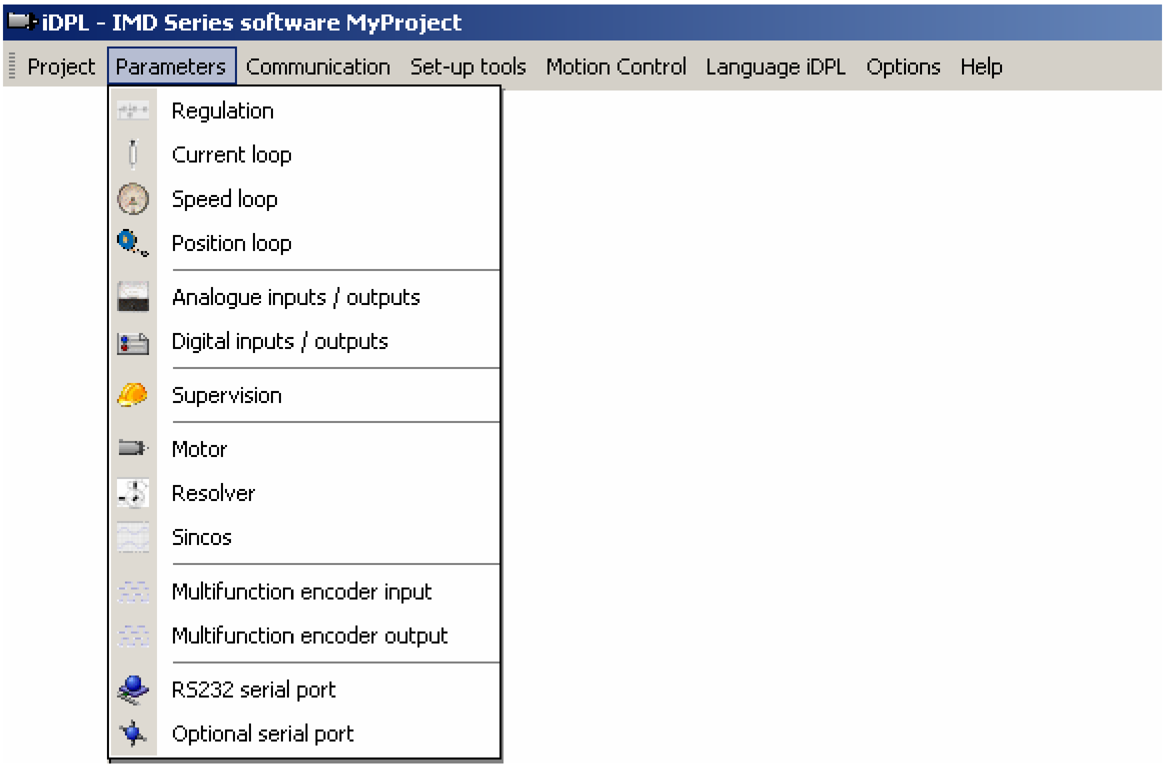
Action : Print the entire contents or selected items of a project.

I) Exit :

Icon : 

Action : Exit the program.

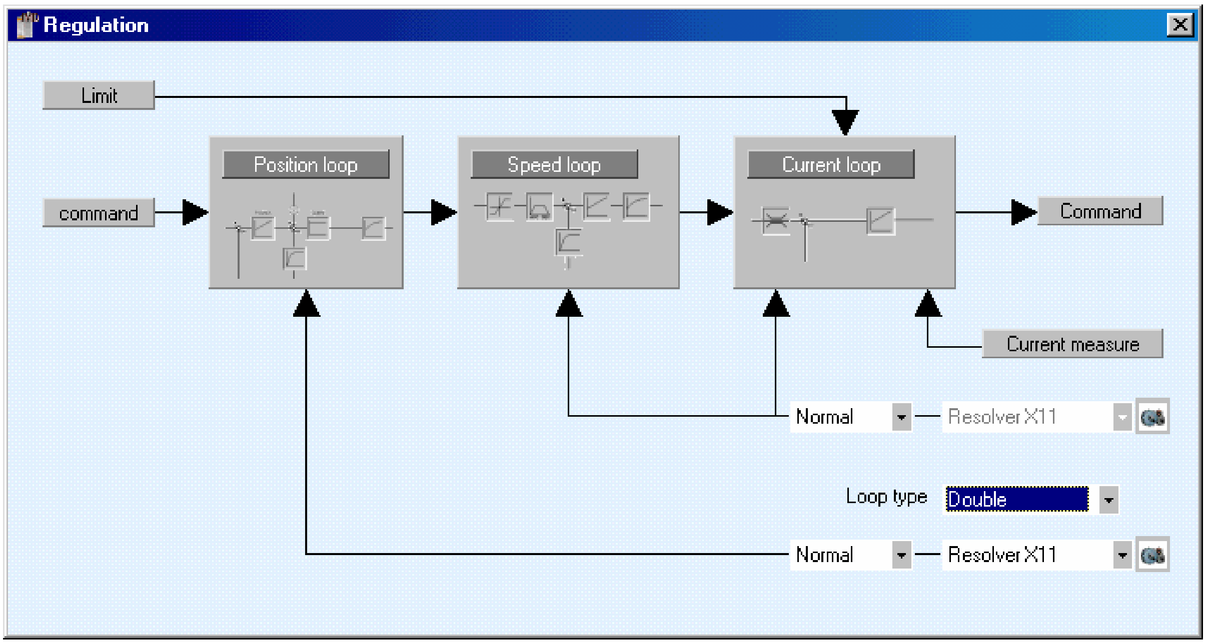
3-3-2- Parameters



A) Regulation :

Icon : 

Action : Principal window for the drive regulation allows access to all other regulation and configuration windows.



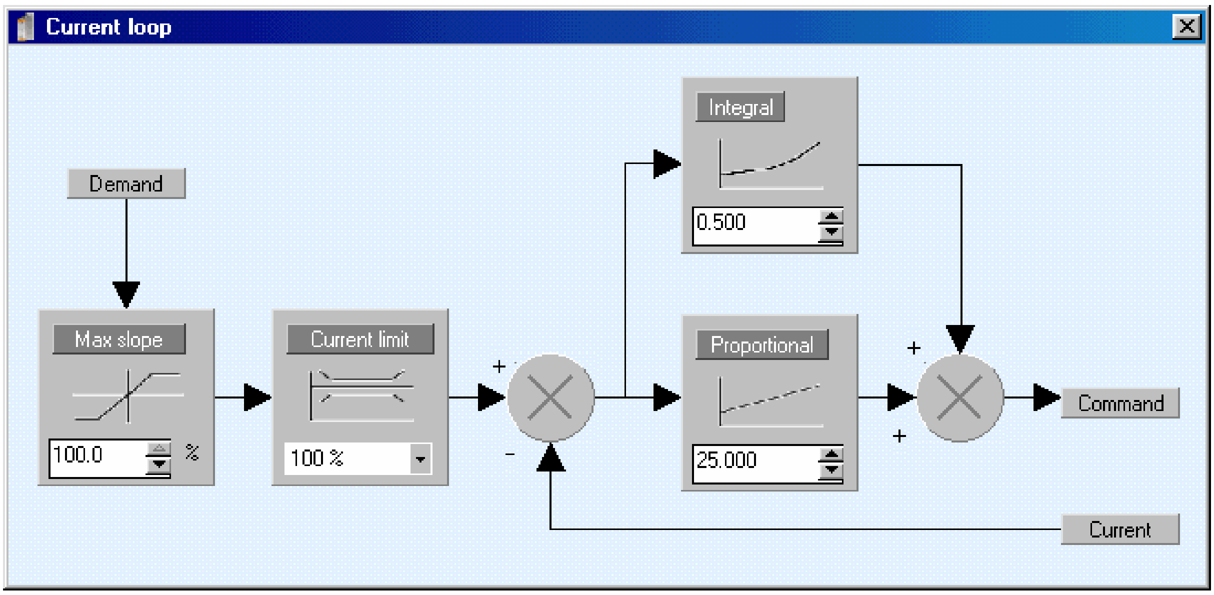
Simple loop: the three regulation loops use the same feedback (resolver or SinCos). It is possible in this screen to modify the position feedback signal.

Double loop: the position loop uses a feedback (resolver or SinCos) different from the two other loops. It is possible in this screen to modify position feedback signal.

B) Current loop :

Icon : 

Action : Configure the drive current loop parameters.



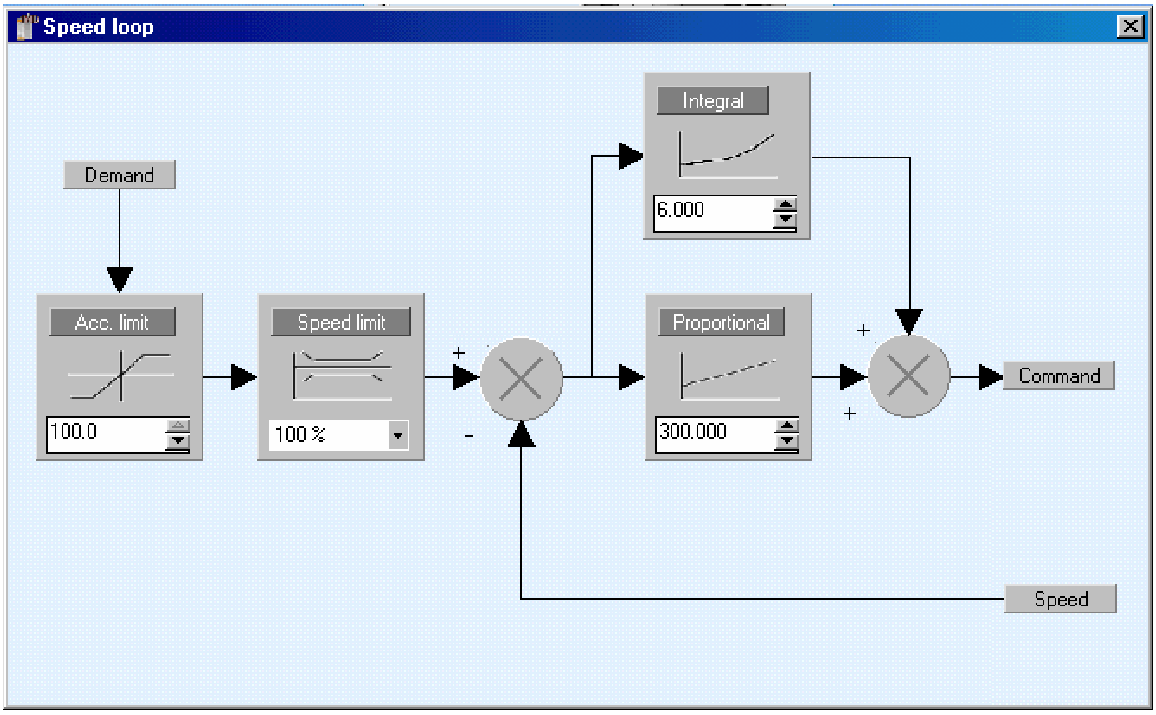
- Demand: Select the command source: value (expressed as a percentage of maximum motor current), analogue input, speed loop or RS232.
- Max slope : Limit the rate of change of current.
- Current limit : Limit the current as a percentage of the nominal value.
- Integral gain : Set the integral coefficient 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).

C) Speed loop :

Icon : 

Action : Configure the drive's speed loop parameters.



- Demand : Select the command source : value, analogue input, position loop, RS232 ....
- Acceleration limit : Limit the rate of change of speed.

Table showing relationship between acceleration limit percentage and time for speed to increase from zero to nominal motor velocity :

Percentage	Time
100%	no limit
50%	20 ms
10%	100 ms
1%	1s
0,10%	10s

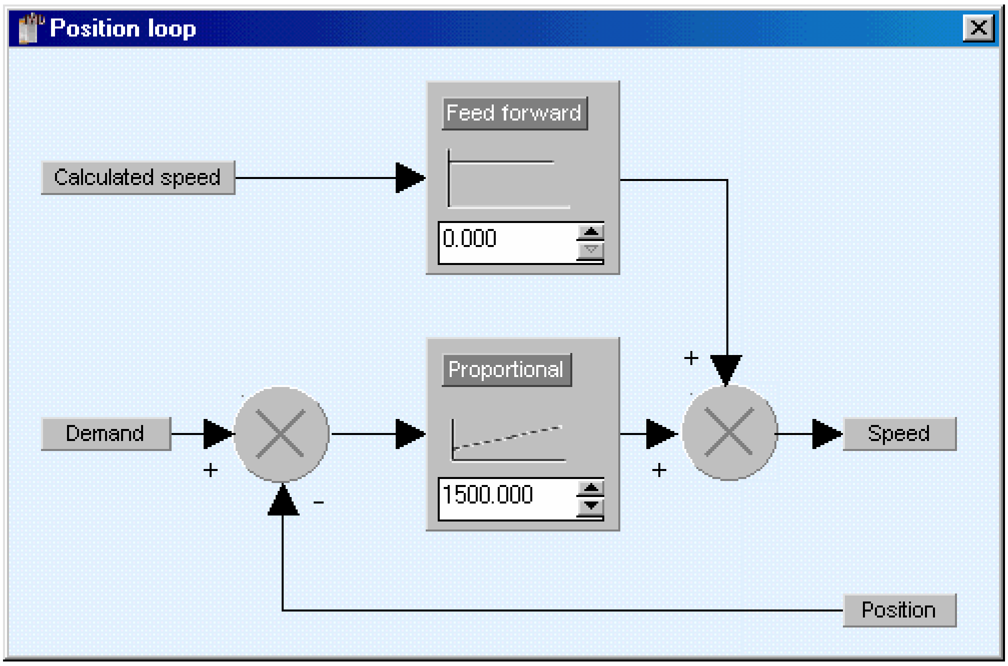
- Speed limit : Limit the speed as a percentage of the nominal value.
- Integral gain : Set the integral coefficient of the control loop.
- Proportional gain : Set the proportional coefficient of the control loop.

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

D) Position loop :

Icon : 

Action : Configure the drive position loop.

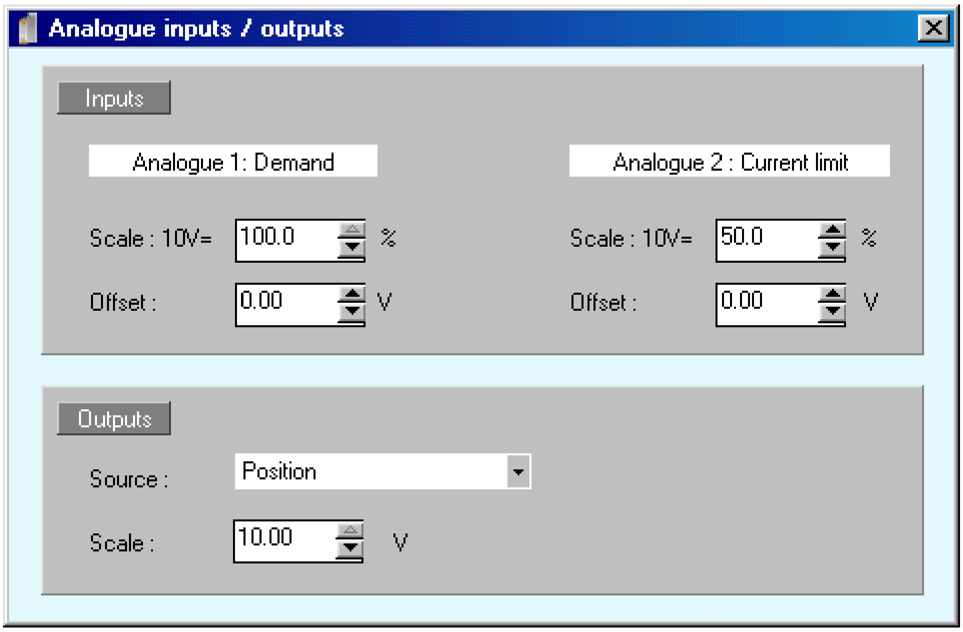


- Feed forward : The feed forward gain can be used to give a following error close to zero.
- Proportional gain : Set the proportional coefficient of the control loop.

E) Analogue inputs / output :

Icon : 

Action : Configure the analogue I/O.



- Analogue inputs :

  In current loop, Analogue 1 can be use as demand source and Analogue 2 as limit current with maximum value: Inom * Imax (see Parameter \ Motor)

  Scale : 10V= : assigns a percentage to the maximum 10V input signal (knowing that 100% is the maximum value of the current or the speed in speed loop).

  Ex : Nominal speed = 3000 rev/min

  Maximum speed = 110 %

  Voltage on analogue 1 -> ± 5V

  Then we have the maximum speed = 3300 rev/min and we will put 200% in the scale parameter so that 5V on Analogue 1 corresponds to maximum speed.

- Analogue output :

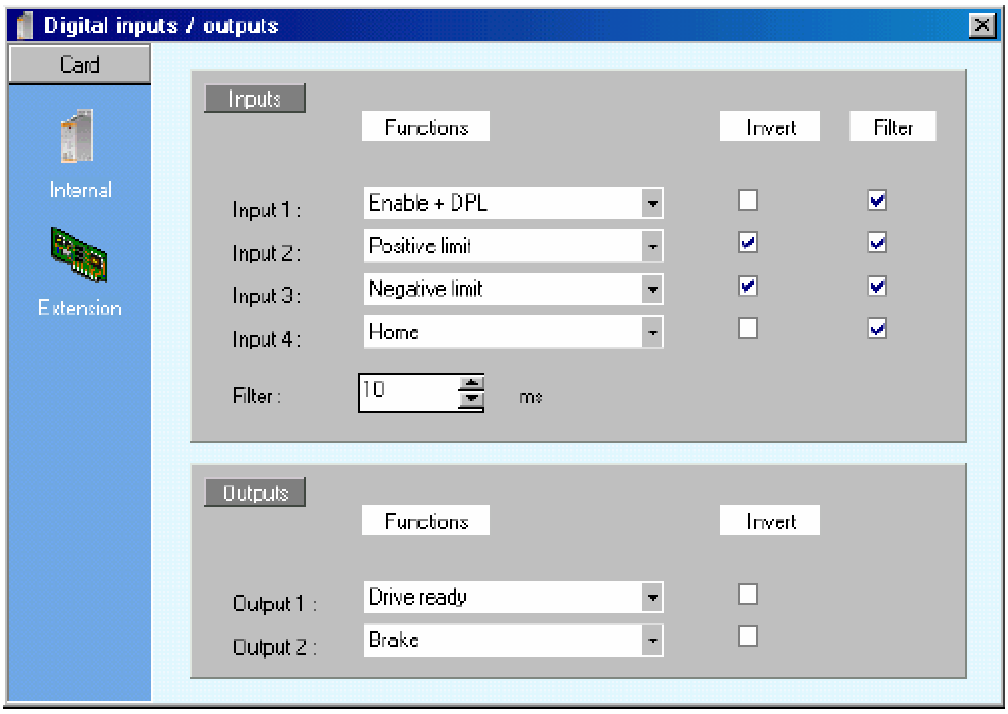
Output	Min. value	Max. value
None	-	-
Position	- 1/2 rev	+ 1/2 rev
Current demand	- Inom. * Imax.	+ Inom. * Imax.
Actual current	- Inom. * Imax.	+ Inom. * Imax.
Speed demand	- Spd. Nom. * Spd. Max.	+ Spd. Nom. * Spd. Max.
Actual speed	- Spd. Nom. * Spd. Max.	+ Spd. Nom. * Spd. Max.
Following error	- Following err.	+ Following err.

Scale : Selects a range for the analogue output.

F) Digital inputs / outputs :

Icon : 

Action : Configure the digital I/O.

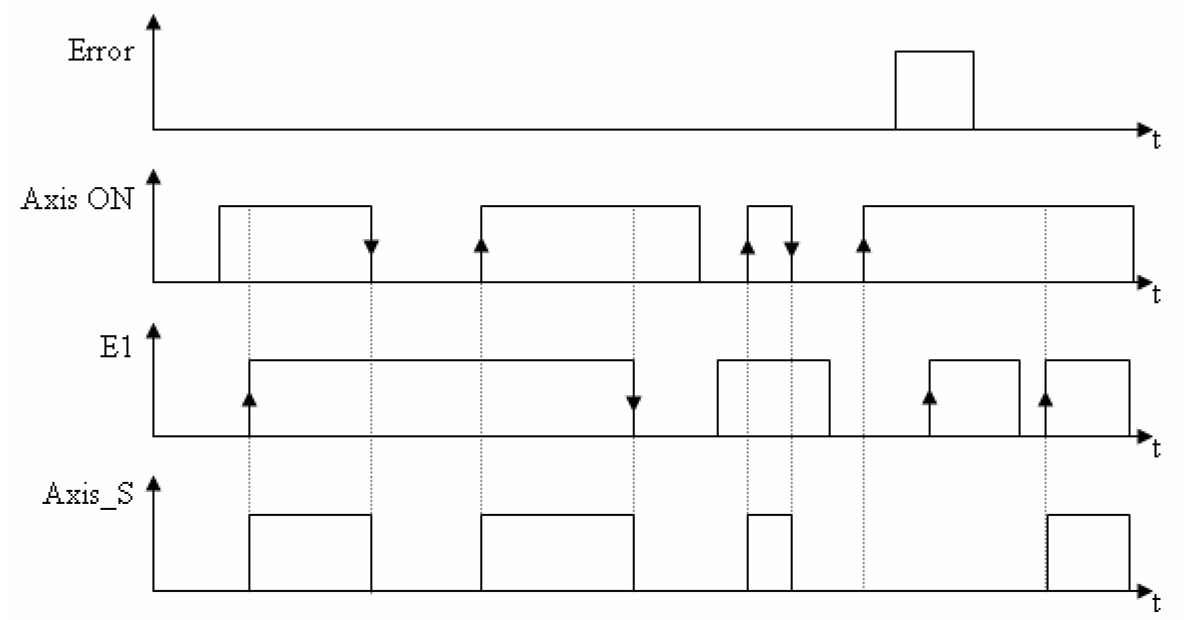


- Input 1 : Selection : Drive **Enable** or none.

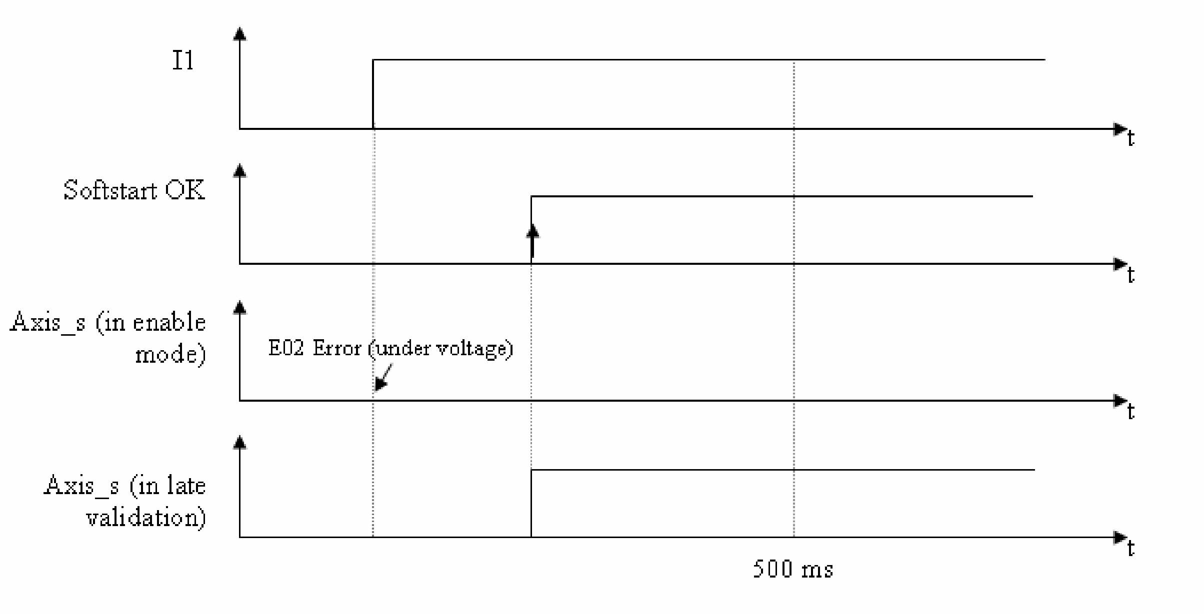
1. If **None**, the power stage of the drive is activated by the Enable button in the main iDPL window or by an Axis On / Axis Off instruction in a iDPL task.

2. If **Enable**, control is done on rising edge of the logical input E1.

3. If **Enable + iDPL**, control is done on rising edge of the logical input E1 and by the Axis Off instruction followed by Axis On of language iDPL.



4. If **Late validation**, control request is done on rising edge of the logical input I1 but control is done on Softstart and SINCOS (if used) validation, the timeout is 500ms.



- 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.

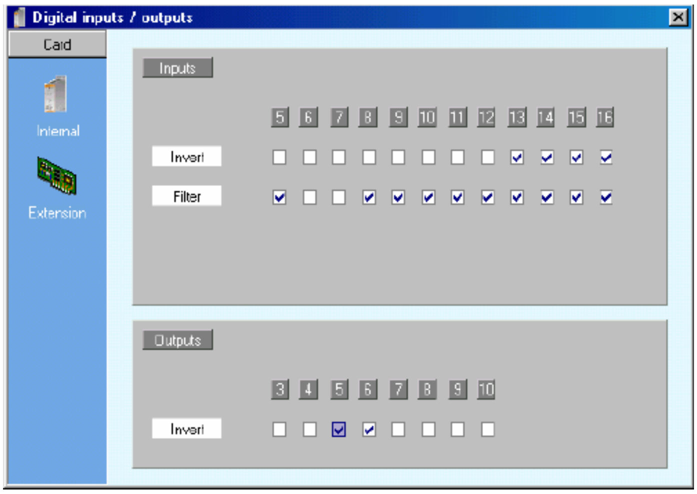
If the brake option is selected for output 2, it is necessary to add an external relay to control the brake as the output current from the drive is limited to 100mA.

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 sensors are active.

To use inputs 3, 4, 15 and 16 in fast mode, deactivate their filters.

With an extension card, you can have:



- 12 additional inputs that can be filtered and/or inverted (to use fast inputs 15 and 16 deactivate filtering).
- 8 additional outputs that can be inverted.

G) Supervision :

Icon : 

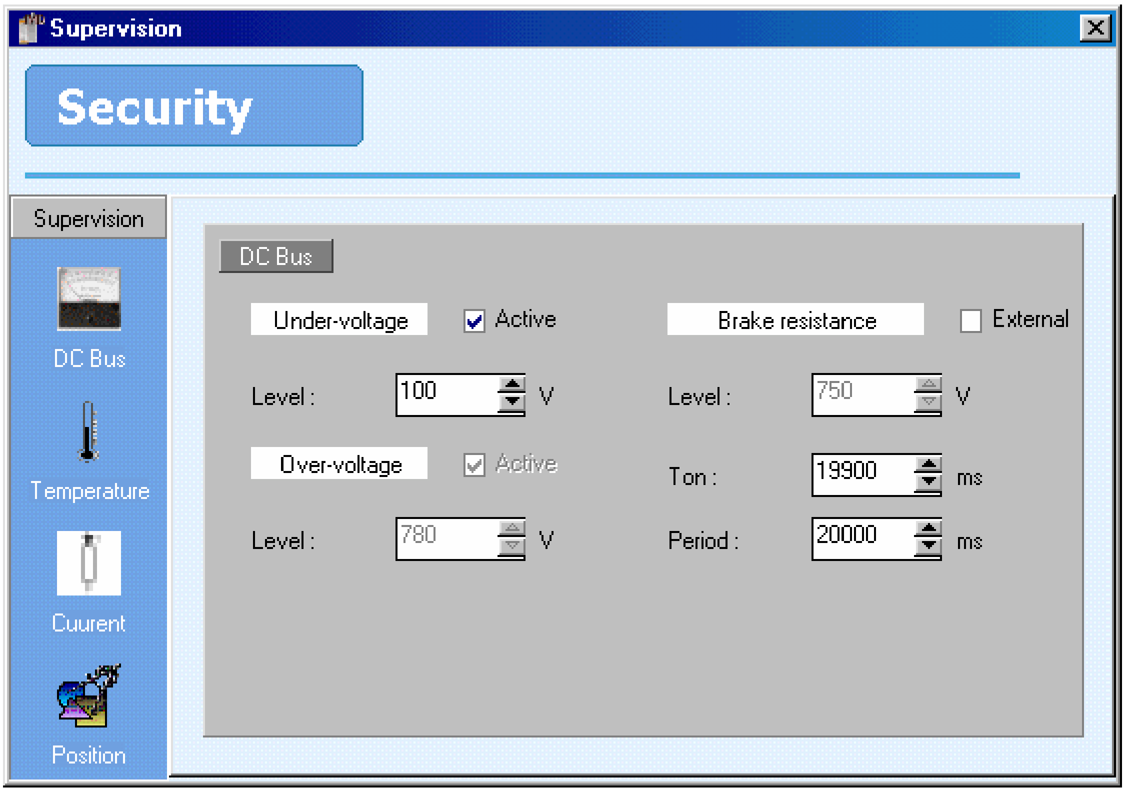
Action : Configure the security parameters.

a) DC Bus monitor :

 **Factory settings do not modify.**

When an external brake resistor has been used select the tick-box External (if it is unchecked, drive uses default parameters to control the ballast).

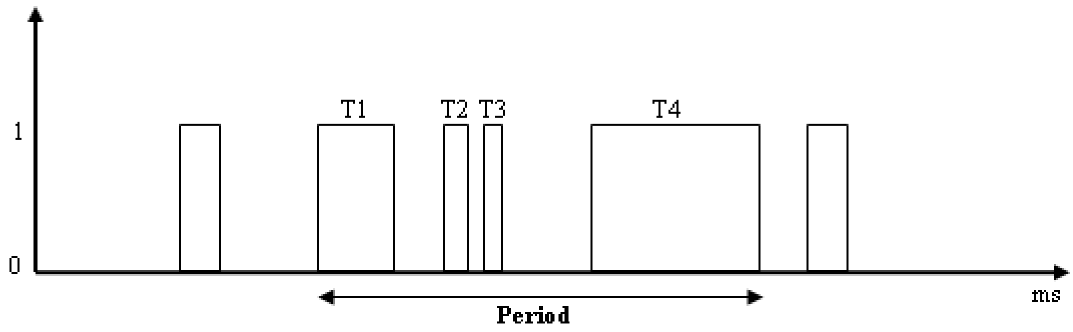
This resistance must be carefully chosen. The adjustments are only accessible when advanced parameters are selected.



- Under voltage: active by default, drive minimum voltage when drive enabled (gives Error E02 under voltage).
- Over voltage: active by default, drive maximum voltage (gives Error 01 over voltage).
- **Warning**: This parameter is only used if Nominal voltage parameter is « Other » (parameters window) else default values are used (390V for nominal voltage 230V, 780V for nominal voltage 400V).

- External brake resistance: check this box if you add an external brake resistance to the drive.
- Brake level: sets the low limit to activate the external brake resistance.
- **Warning**: This parameter is only used if External brake resistance box is checked or if Nominal voltage parameter is « Other » (parameters window) else default values are used (375V for nominal voltage 230V, 750V for nominal voltage 400V

- Ton and Period : allows you to define the duration and duty for the brake resistance :



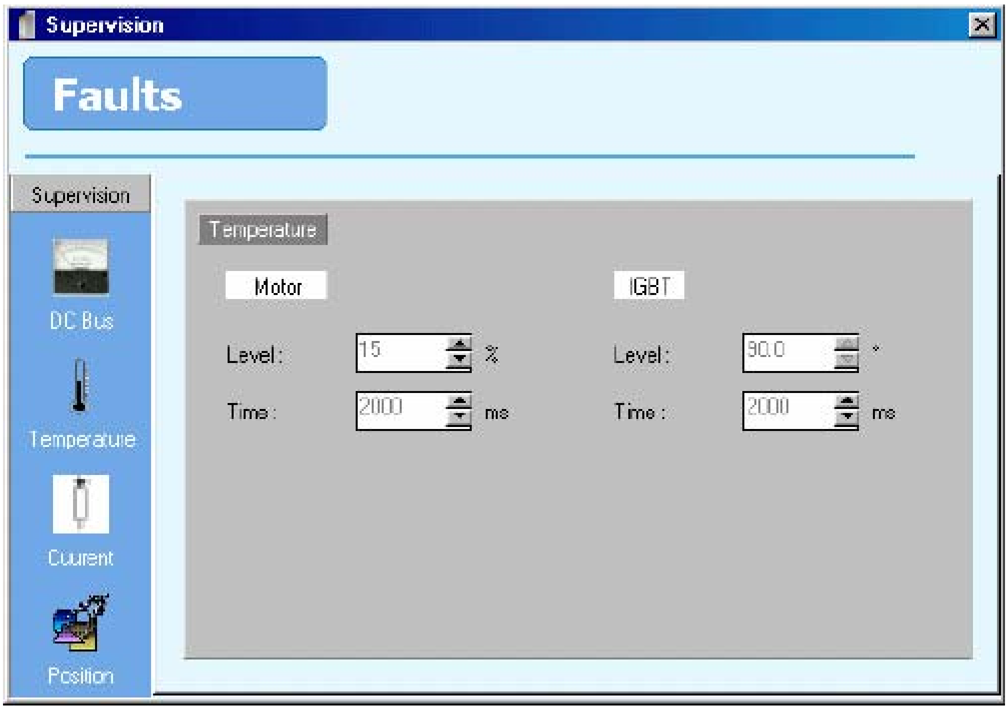
Ton = total of activation time (T1, T2 ...) of the resistor during the time **Period**

While the activation time is less than Ton length during a time interval set by Period, the DC bus over-voltage can be reduced by the brake resistance.

Warning: This parameter is only used if External brake resistance box is checked.

b) Temperature monitor :

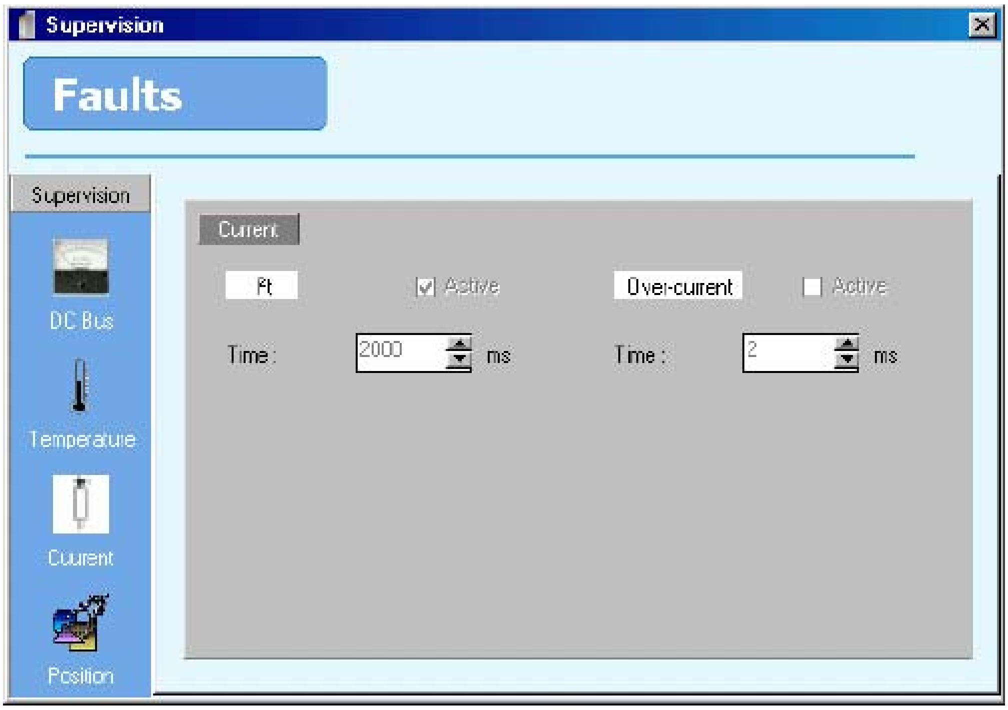
Factory settings, do not modify.



- Motor temperature: defines the level and length of the motor over-temperature and gives error E07.
- IGBT temperature: defines the level and length of the IGBT over-temperature and gives error E06.

c) Current monitor :

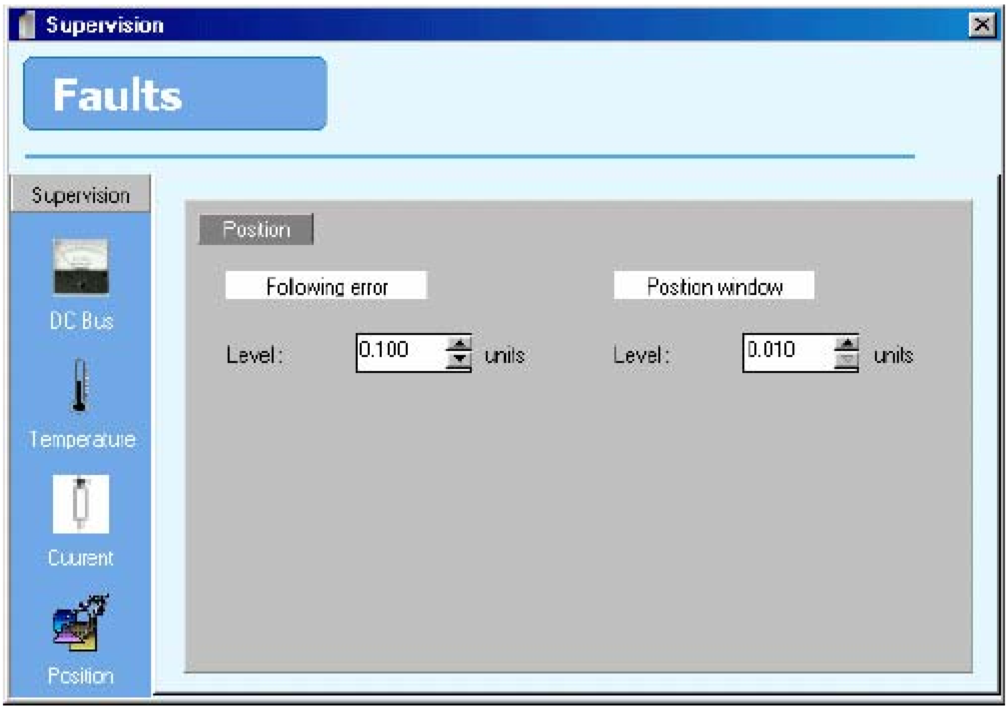
 **Factory settings do not modify.**



- I^2t: Brushless motors can accept peak currents (greater than Inom*2). I2t verifies if the average current is always less than Inom. In correct use, I2t must keep null.
- Time : defines the length of one control period.
- Over current : the drive always controls the current if it is within its range, if the current is out of limit during the time parameter then there is an error E04 over-current.

d) 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 20 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.

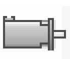
  The control of this value is very important: a value too small can lead to spurious errors; a value too large can reduce the overall safety margins of the machine.

  Warning: the value of the following error depends of the unit in the **Motion control \ Units** window.

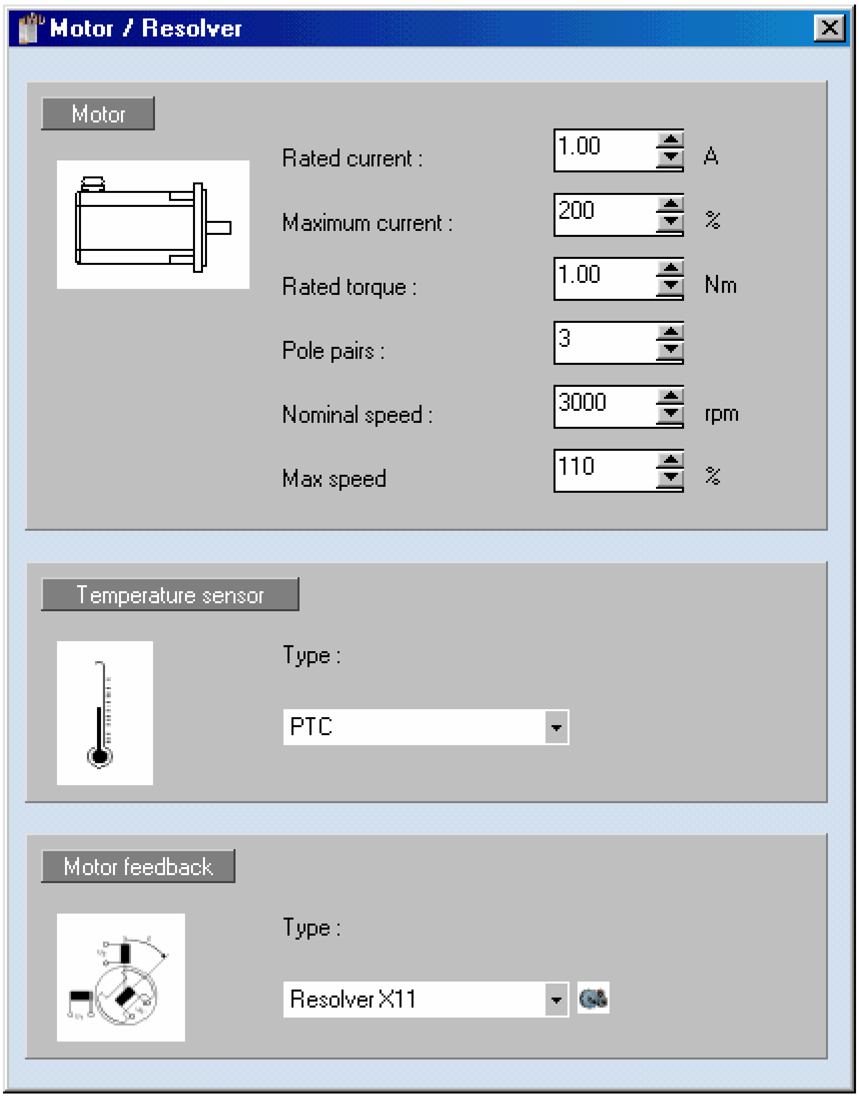
- Position window: A the end of a movement, the movement is considered to be completed only when the difference between the actual position and the theoretical position is less than the position window value.

  Warning: the value of the position window error depends of the unit in **Motion control \ Windows units**.

H) Motor :

Icon : 

Action : Configure the motor and resolver.



a) Motor :

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

Maximum current : A percentage of the rated current. Default value 200%.

Rated torque : Rated motor torque in Nm (only use for display).

Pole pairs : Must correspond to the motor being used.

Nominal speed : Nominal speed (rev/min)

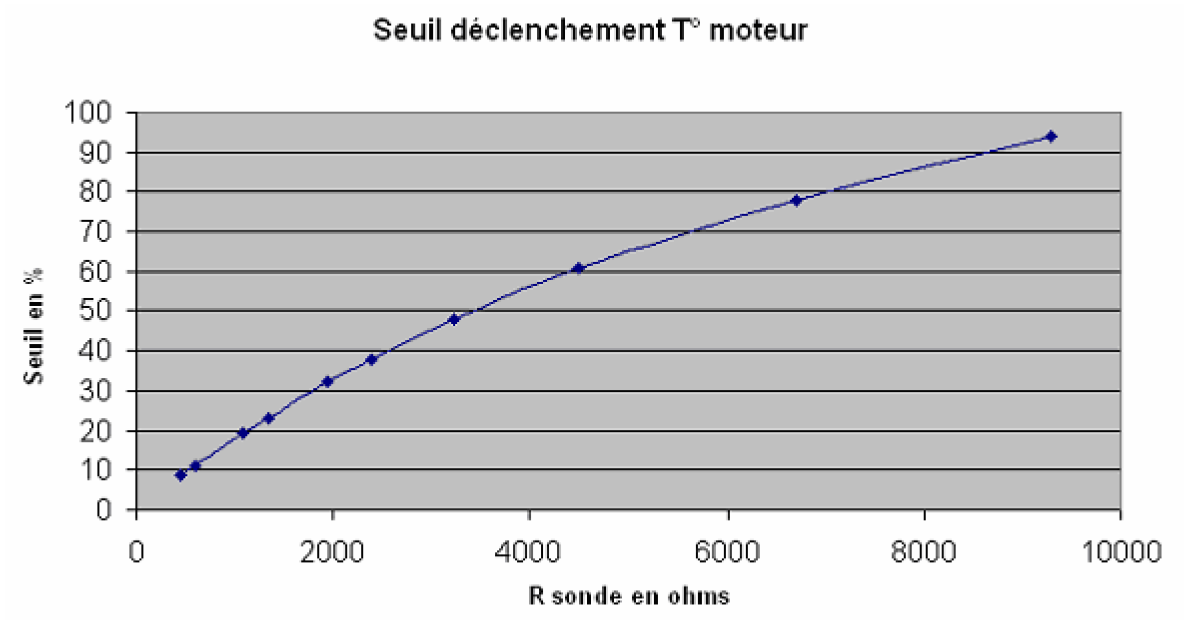
Maximum speed : A percentage of the nominal speed, use to limit the motor in speed loop.

b) Temperature sensor:

Type : PTC or NTC

PTC sensor: Error when the sensor in over drive sensor threshold.

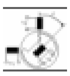
NTC sensor: Error when the sensor in under drive sensor threshold.



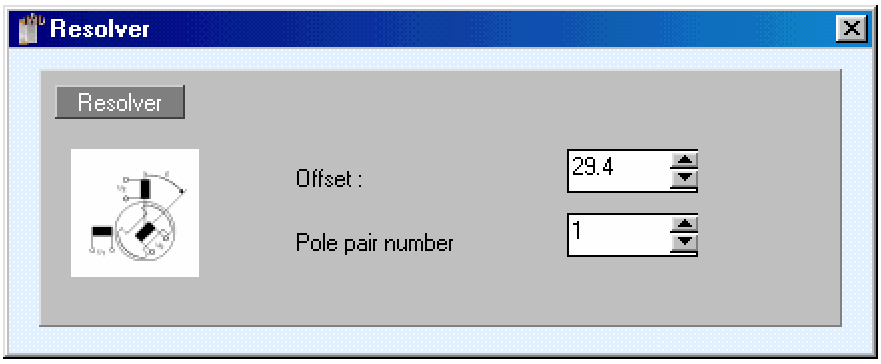
c) Motor feedback:

Type : choice of the motor feedback (resolver X11 or SinCos X13).

I) Resolver :

Icon : 

Action : Set up the resolver.

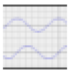


Offset : Resolver offset.

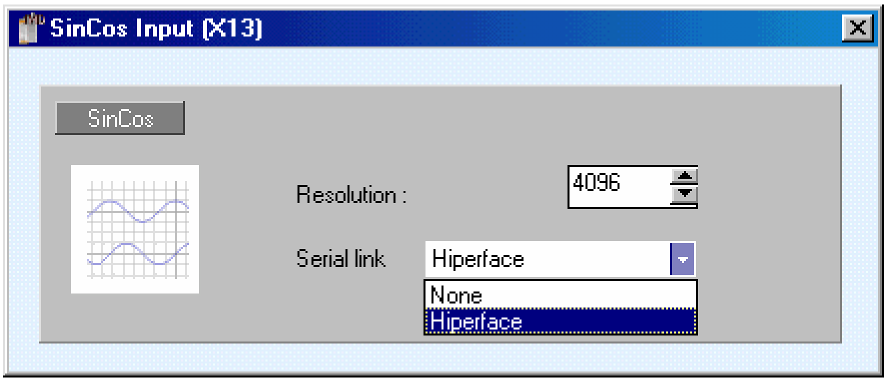
Pole pairs : Must correspond to the used resolver

For resolver with several pole pairs, you have several rotor position for 1 motor position (ex : 0, 120° or 240° for resolver with 3 pole pairs). So a HOME on TOP Z can have several physical position (offset = number of pole pairs / 360°).

J) SinCos :

Icon : 

Action : Set up the SinCos.



Resolution : Defines the number of encoder increments (4 increments by point). Ex : For a 500 line encoder, chose 2000 increments.

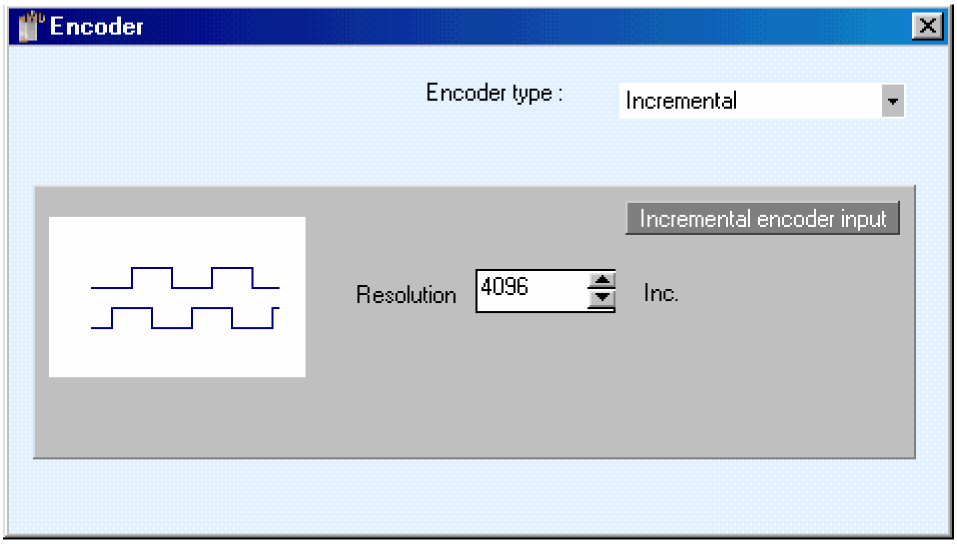
Serial link : If none is selected, then the feedback is relative, if Hiperface is selected then feedback is absolute.

If there is an fault on the serial link when the drive is activated, the drive gives an error E08.

K) Encoder input :

Icon : 

Action : Set up encoder input.

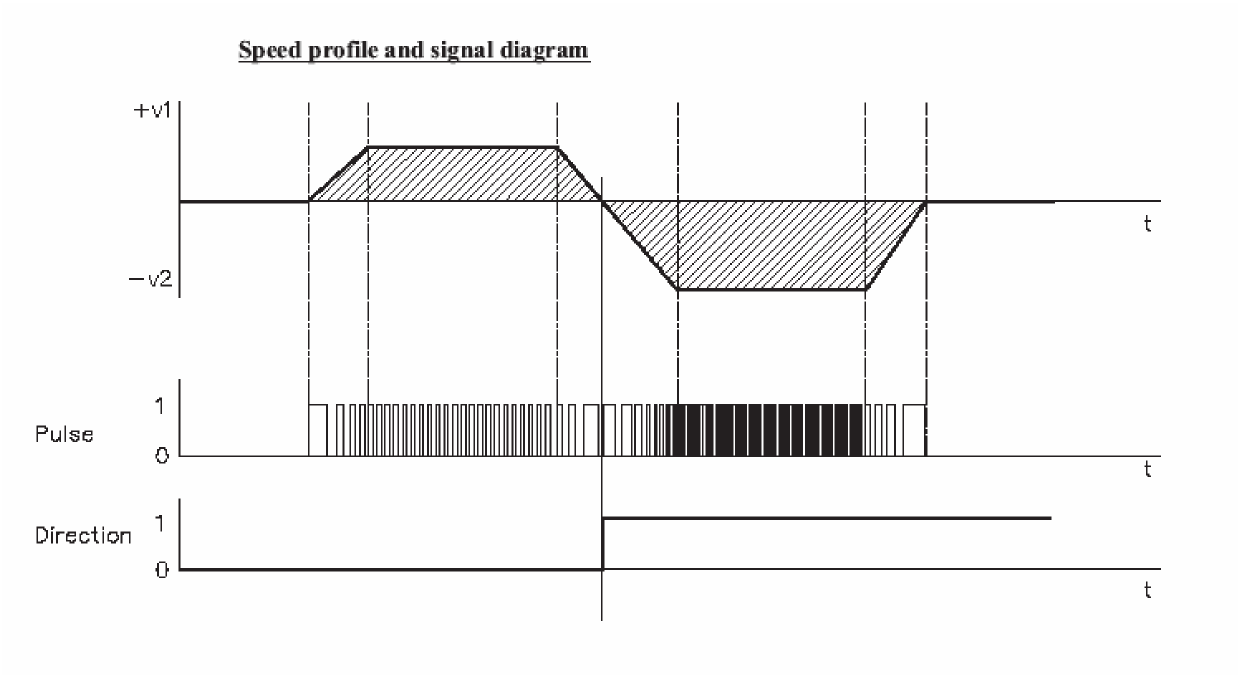


a) Incremental mode :

Resolution : Defines the number of encoder increments (4 increments by point).
Ex : For a 500 line encoder, chose 2000 increments.

b) Stepper mode :

Allows the IMD drive to be connected to a third-party stepper-motor controller. The number of steps and rotation direction can be changed.

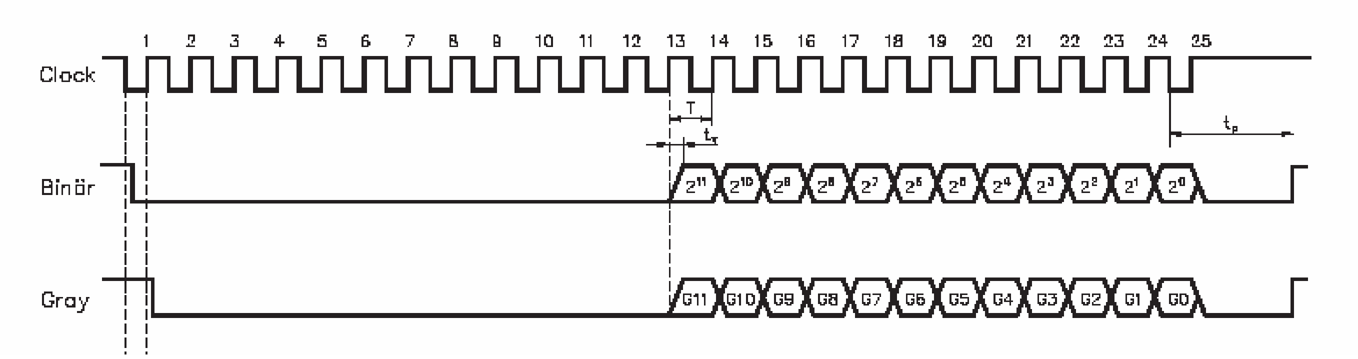


Resolution : input the resolution in increments (4 increments per line).
For example, for an encoder with 500 pulses per rev enter 2000 increments.

c) SSI mode :

Allows an SSI absolute encoder to be used for master functions or double loop regulation.

The position of the motor shaft is calculated from the cyclic-absolute signals of the resolver or encoder.



Bit : Number of bits for position information (from 2 to 31).

Frequency : Clock frequency (1,5 MHz max)

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

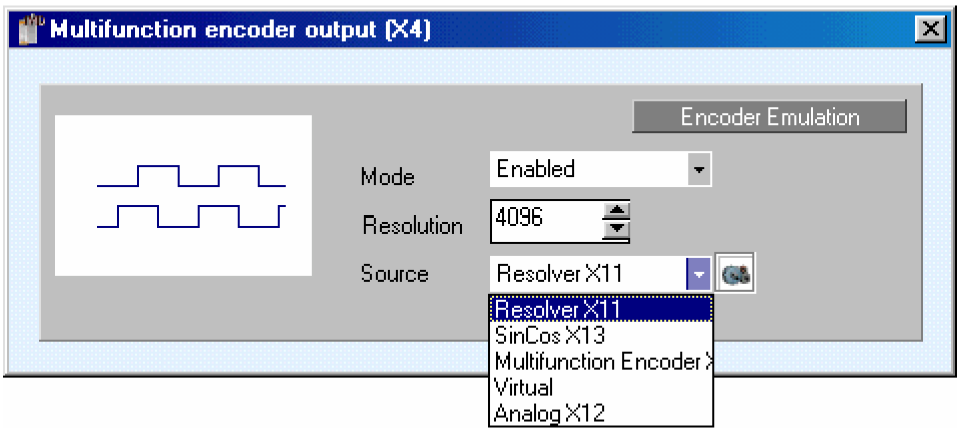
GRAY code: Yes/No (see encoder documentation)

Warning : The resolution must be less or equal to 2^nb Bit and maximum capture time (2^nb Bit / Frequency) must be under 100µs.

L) Encoder output:

Icon : 

Action : Set up encoder output.



Disable mode: The encoder output is not used.

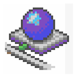
Enable mode: The encoder output returns an incremental signal using the selected source and resolution.

- Source : Resolver, SinCos, multi-function input (incremental, stepper, SSI), Virtual, Analogue
- Resolution: input the resolution in increments.

Bypass mode: copy the encoder input to encoder output.

IMD bus : Not available on this version.

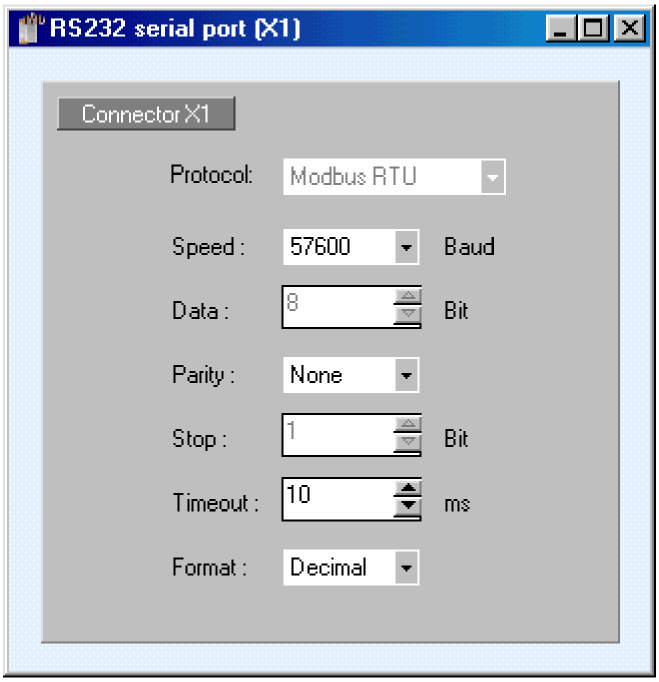
M) RS232 serial port (fitted as standard) :

Icon : 

Action : Configure the port for Modbus.

The drive uses this connection in Modbus RTU slave mode.

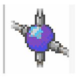
The data format is fixed as 8 bits, 1 stop bit, no parity.



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.

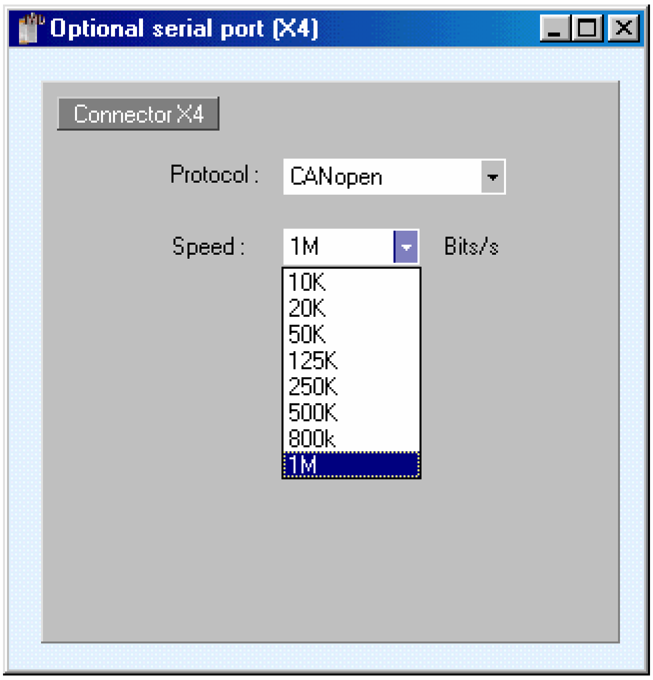
 With the system communication, the signal RTS from the PC is used and is forced to a logic 1.

N) Optional serial link :

Icon : 

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

- CANopen :



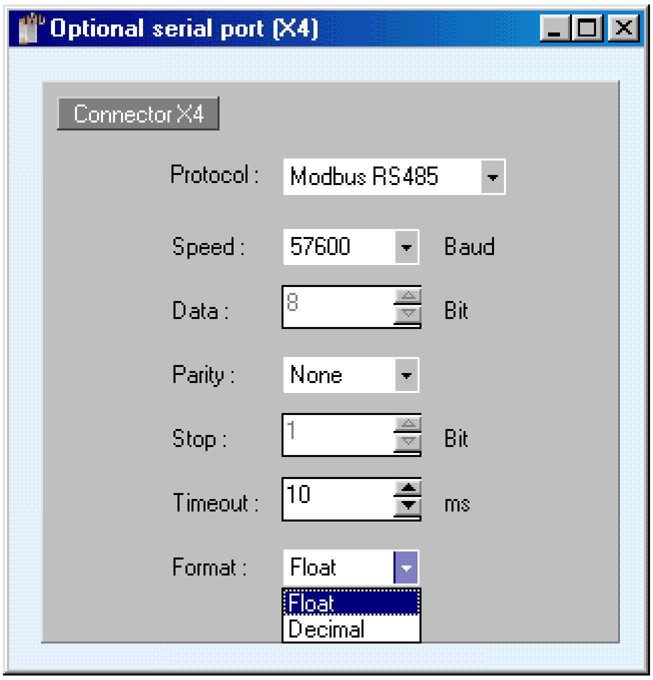
Speed : Defines the communication speed used by the CANopen bus.

For more information, see the appendix relating to CANopen .

- 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, no parity.



Settings:

Node Address: For the NodeID corresponds to the five first dipswitchs + 1

Ex: dipswitchs: 1 -> ON, 2 -> OFF, 3 -> ON, 4 -> OFF, 5 -> OFF

Dipswitchs value = 1 + 4 = 5

NodeID = 5 + 1 = 6

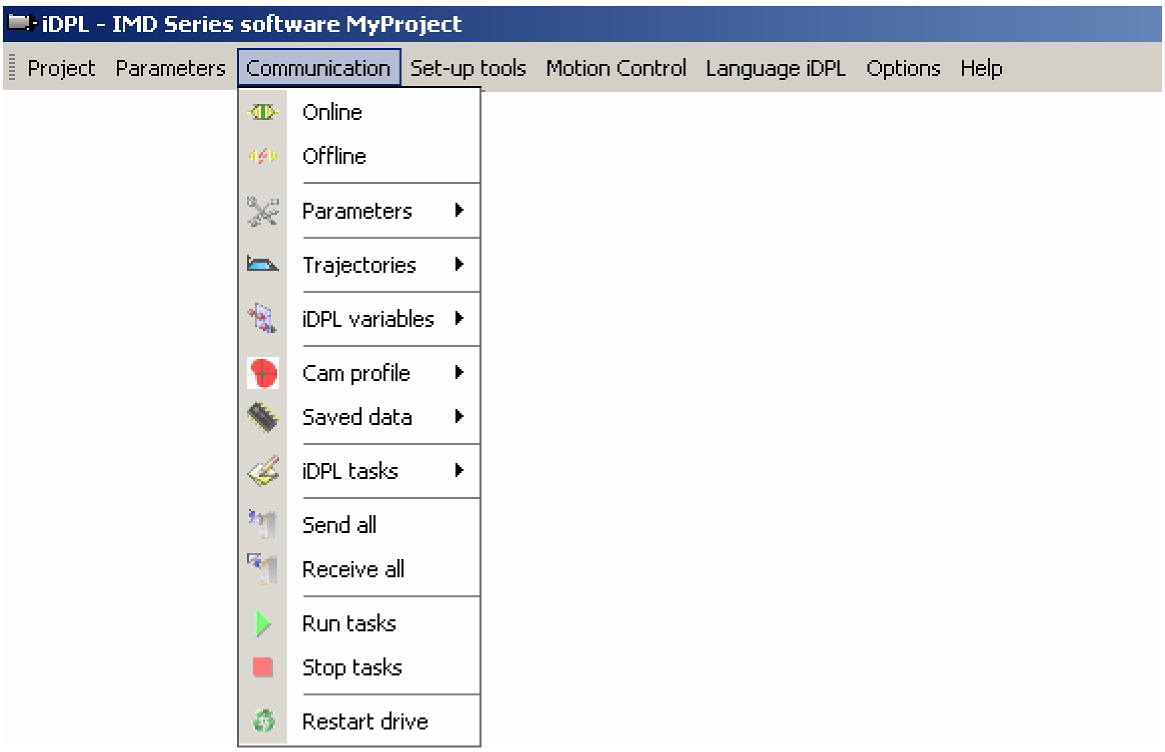
Speed: Set the communication speed of the port.

Timeout: Maximum time without a response.

Format: Select the real format of variables (VR0 .. VR255) or parameters (motor position)

- Floating : use by system communication
- Decimal: number of decimal places depends of the precision parameter in the options\language iDPL\Compiler.

3-3-3- Communication



A) Online :

Icon : 

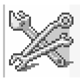
Action : Establish communication with the drive. All parameters shown on the screen correspond with the values stored in the drive.

B) Offline :

Icon : 

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

C) 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.
- ***Import from file and send:*** allow to send an external parameter file from PC to drive. These parameters are automatically saved in the drive.
- ***Save drive parameters*** : transfer the current drive parameters to Flash memory. This allows them to be restored automatically after a supply interruption.

D) Trajectories :

Icon : 

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

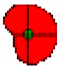
E) iDPL variables :

Icon : 

Action : Sends or receives the initial values of the variables to or from the drive.

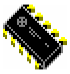
 Only variables VR0 toVR63 and VL0 to VL63 are applicable. At each power-on of the drive these 128 variables are loaded with these initial values.

F) Cam profiles:

Icon : 

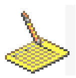
Action : Sends or receives Cam profiles in FRAM.

G) Saved data :

Icon : 

Action : Sends or receives data save in FRAM..

H) iDPL tasks :

Icon : 

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

I) Send all :

Icon : 

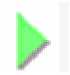
Action : Allows the user to send a package to the drive ; it is possible to select parameters, variables, cams, tasks

J) Receive all :

Icon : 

Action : Allows the user to receive a package from the drive ; it is possible to select parameters, variables, cams, tasks

K) Run iDPL :

Icon : 

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

L) Stop iDPL :

Icon : 

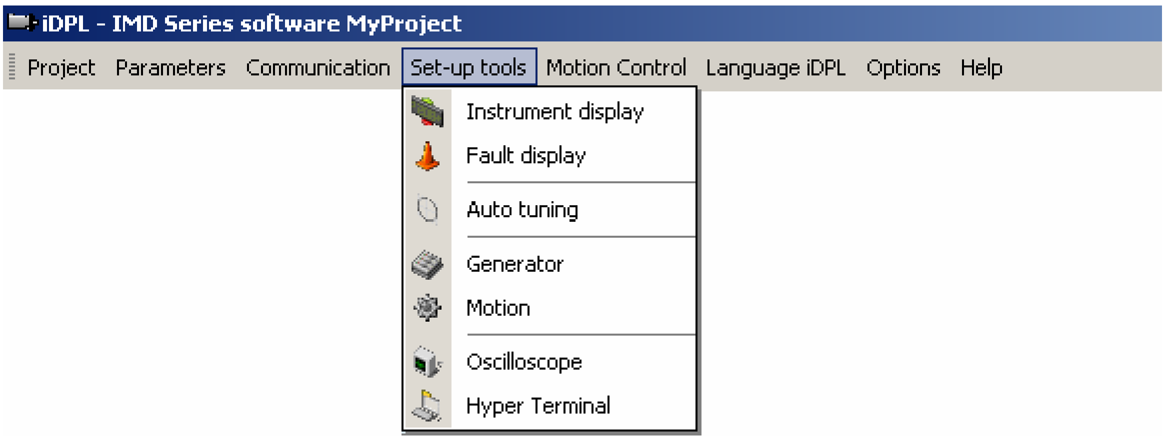
Action : Stops the execution of all of the tasks.

M) Restart :

Icon : 

Action : Restarts the drive.

3-3-4- Diagnostics

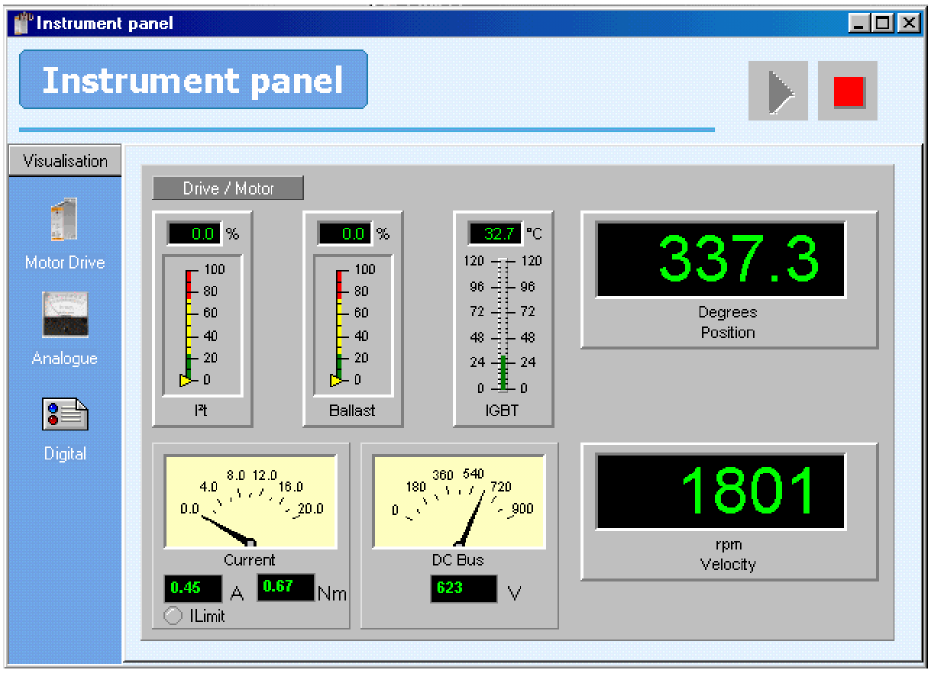


A) Instrument panel :

Icon : 

Action : Allows the monitoring of drive functions

a) Allows the user to see the internal state of the drive and motor.

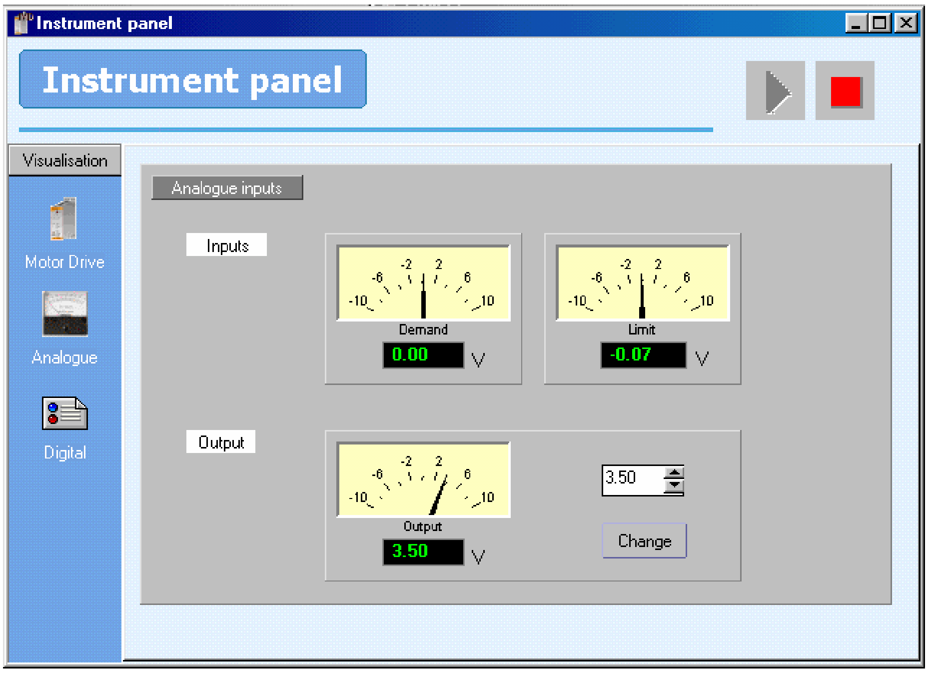


➢ ILimit LED is lit when there is over-current and the following error is growing

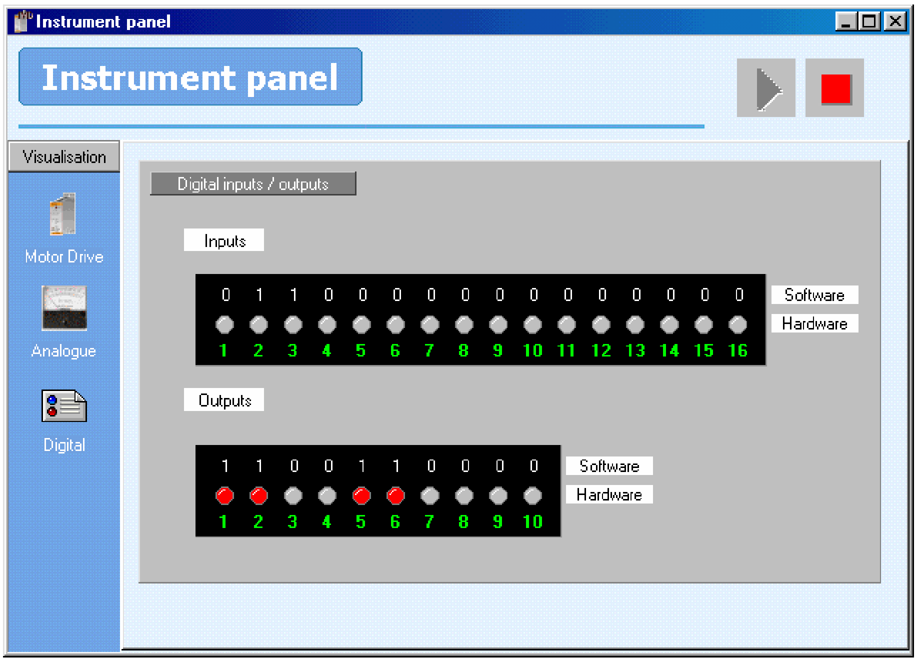
➢ Degrees position : shows the motor position in degrees (0 to 360°)

➢ RPM velocity : shows motor velocity (revs per minute).

b) Allows the user to see the analogue I/O states and to change the output.



c) Allows the user to see the digital I/O states and to change them.



- Click on the switch over output number to change its state.
- Red output are unchangeable such as drive ready, brake.

B) 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 (input E1 or Enable switch in iDPL main screen or Axis off / Axis on iDPL instruction).

C) Auto tuning:

Icon : 

Action : Performs an automatic evaluation of the resolver offset and automatically adjusts all regulation loops.

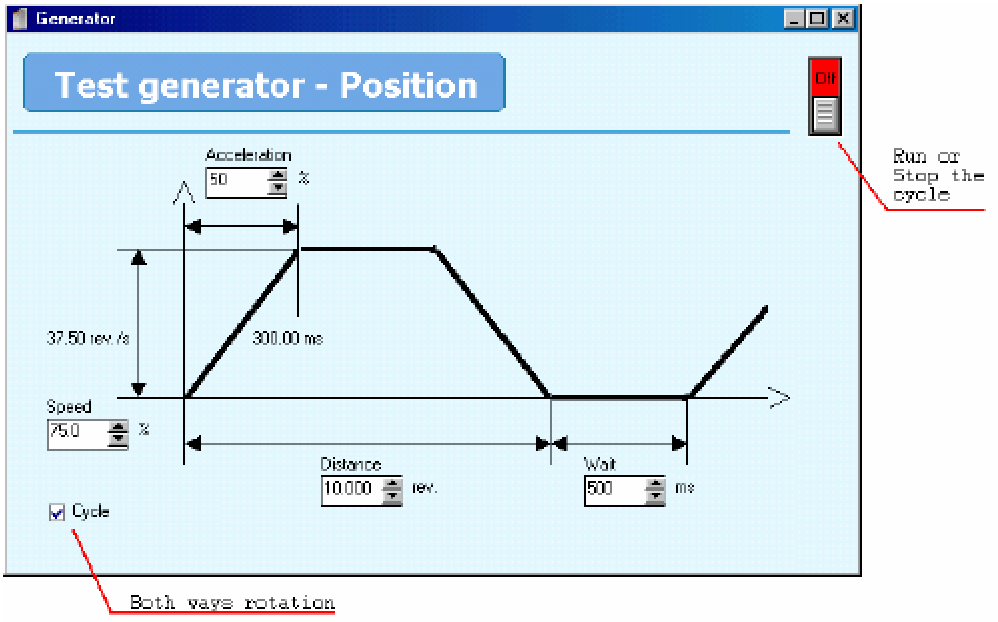
See drive adjustments chapter.

Option only available with advanced parameters selected.

D) Generator :

Icon : 

Action : Generates a range of movements which allow the user to optimize the various control loops in the drive.

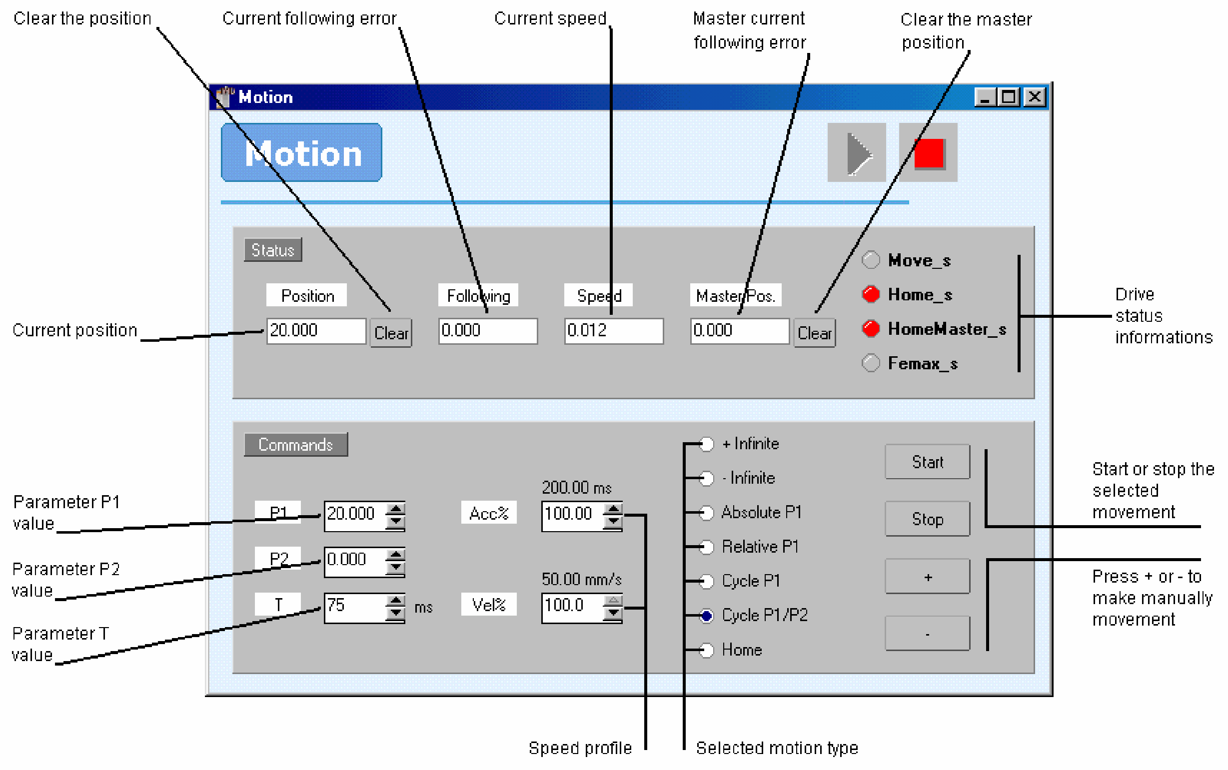


- Set up the generator to carry out the desired movement.
- Activate the drive with the ENABLE button (and / or Input 1).
- Start the movement with the ON/OFF button on the generator.

E) Motion :

Icon:

Action: Allows the testing of the positioning of the axis. It is preferable to start by checking the behaviour 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 switch to controlled mode and adjust the control parameters. When the parameters are correct they should be saved to Flash memory.



F) Oscilloscope :

Icon : 

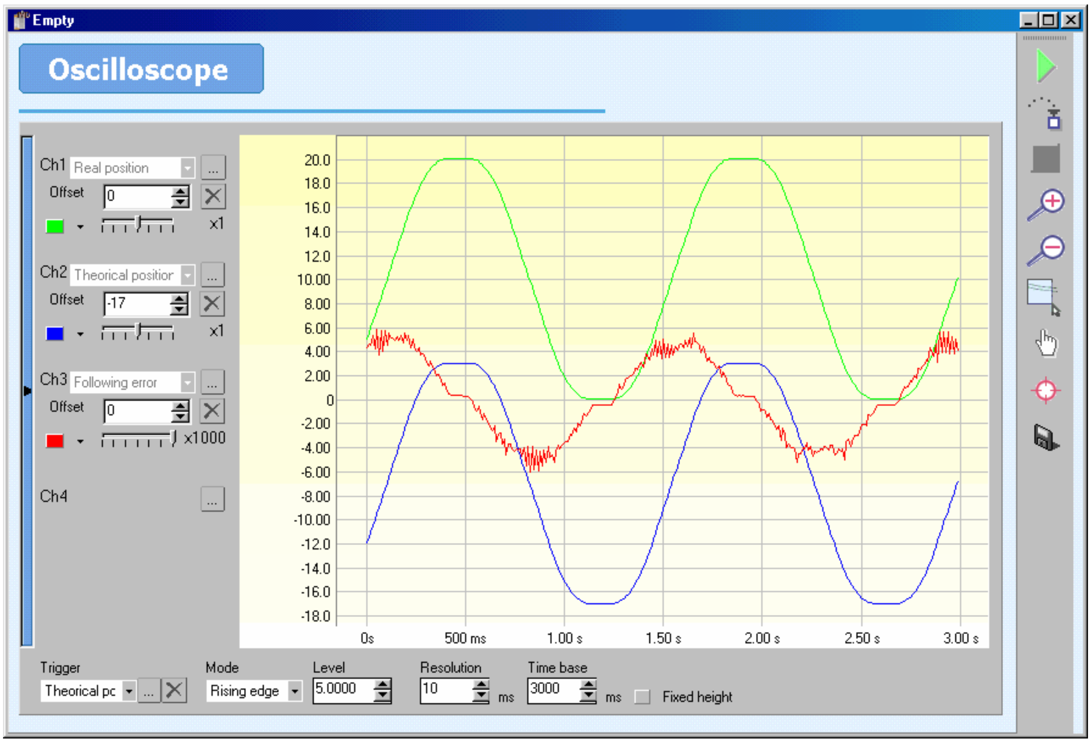
Action : Opens the oscilloscope window. This tool aids commissioning by allowing all of the drive's parameters and states to be observed. Up to 4 channels can be observed simultaneously.

The oscilloscope is divided into three areas :

The display screen

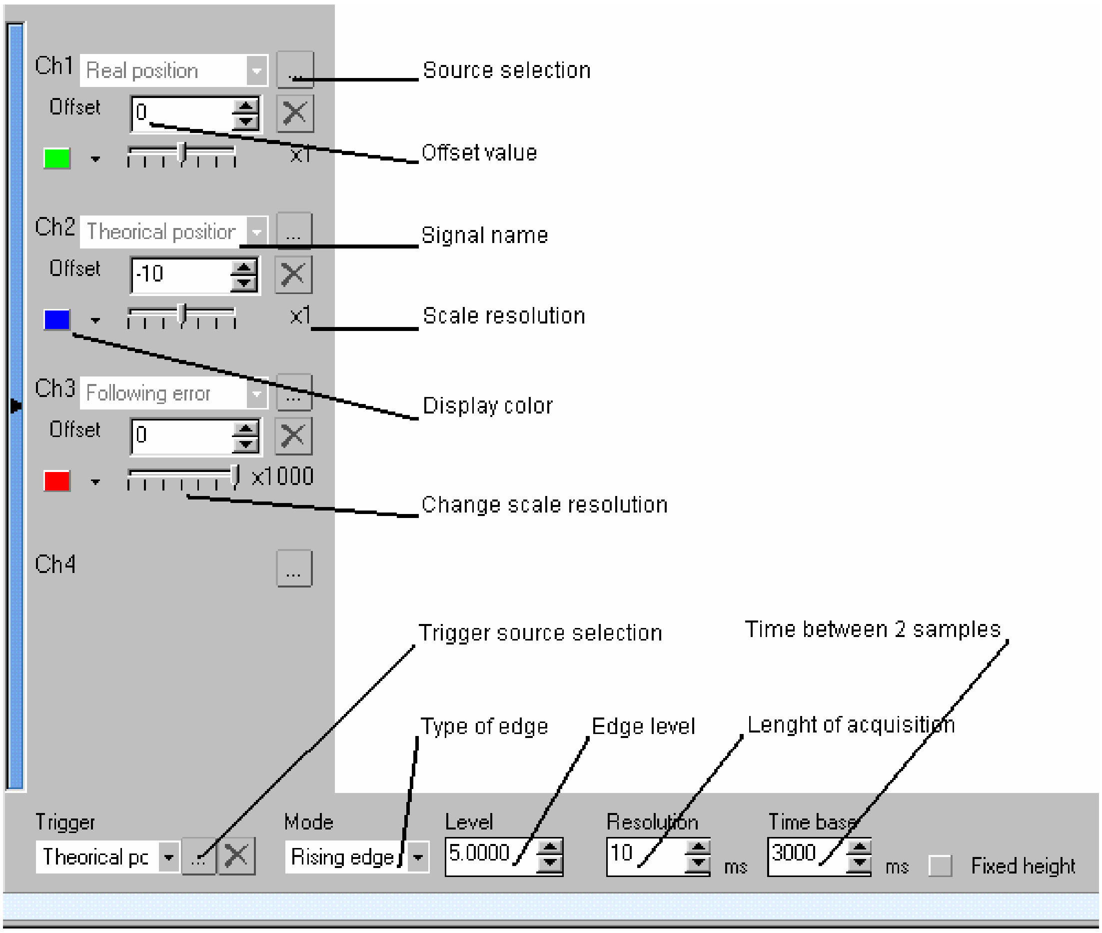
The configuration control area

The display control area



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

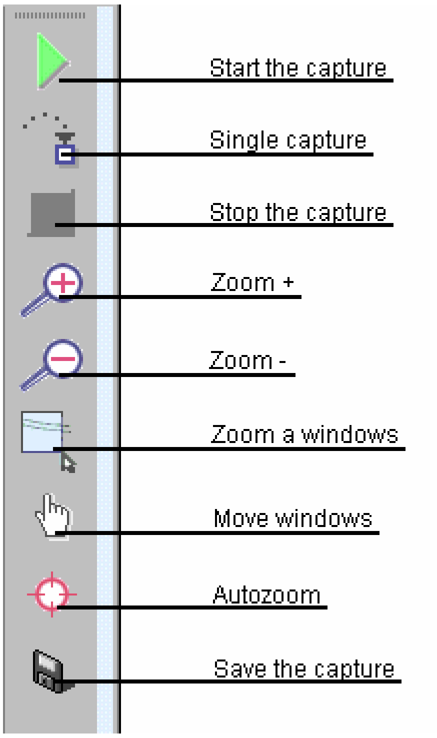
↳ The configuration controls make it possible to choose the signals to be displayed and to set up the mode of acquisition, the number of samples, duration etc.



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

Each channel has a scaling factor to amplify or attenuate the amplitude of the signal.

➯ The display control area is used to start and stop acquisition, and also to modify the plotting on the display screen.



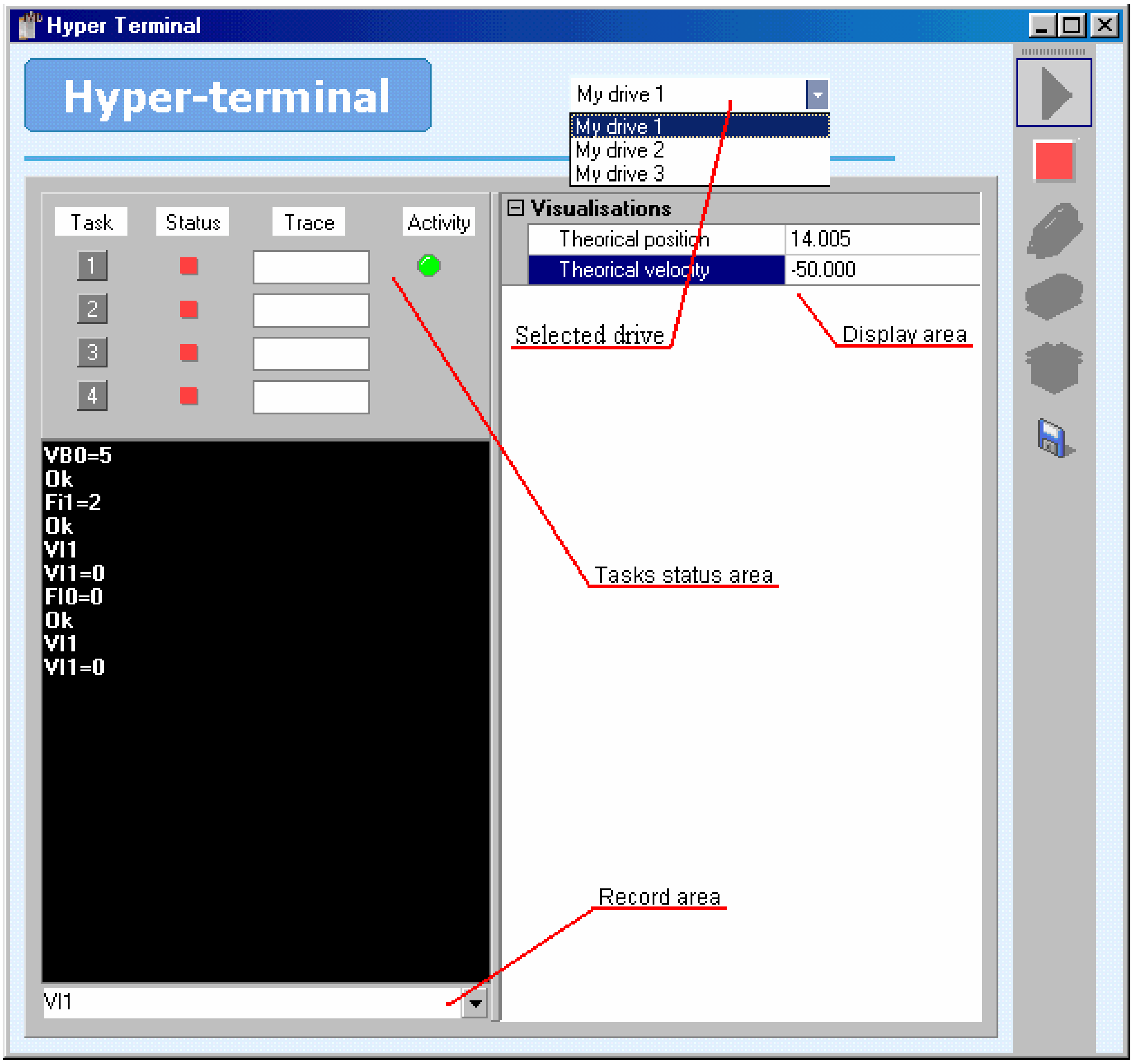
- Zoom window : Click on the switch zoom window. With the switch active, trace out a rectangle on the display screen by keeping the left button of the mouse pressed. Releasing the button completes the zoom
- Save capture : save the current capture as a HTML and JPG file

G) Hyper terminal :

Icon : 

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

In multi-drive mode, select the drive that you want to communicate with.

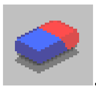


The hyper-terminal window is divided into three areas:

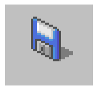
Tasks status area: shows the status and the current line number line of the tasks and communication activity.

Display area: displays a variable, a parameter, an input or an output.

To add a variable or a parameter, click on icon  and double click on a variable or one of the parameters, the name will be displayed in the display area.

To delete a variable or a parameter, select it in the display area and click on icon .

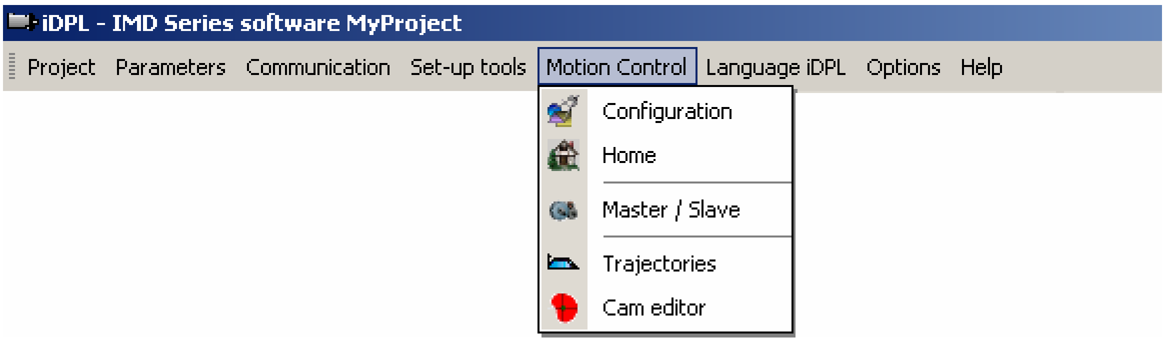
You can display 16 variables or parameters maximum.

It is possible to save or load a HyperTerminal configuration with icon:  , 
.

Record area: used to modify a variable (VF, VB, VI, VL, VR), FRAM variable (FI to integer, FL to long integer and FR to real, long integer and real use 2 consecutive address) or a parameter

3-3-5- Motion control

Menu only available in position mode

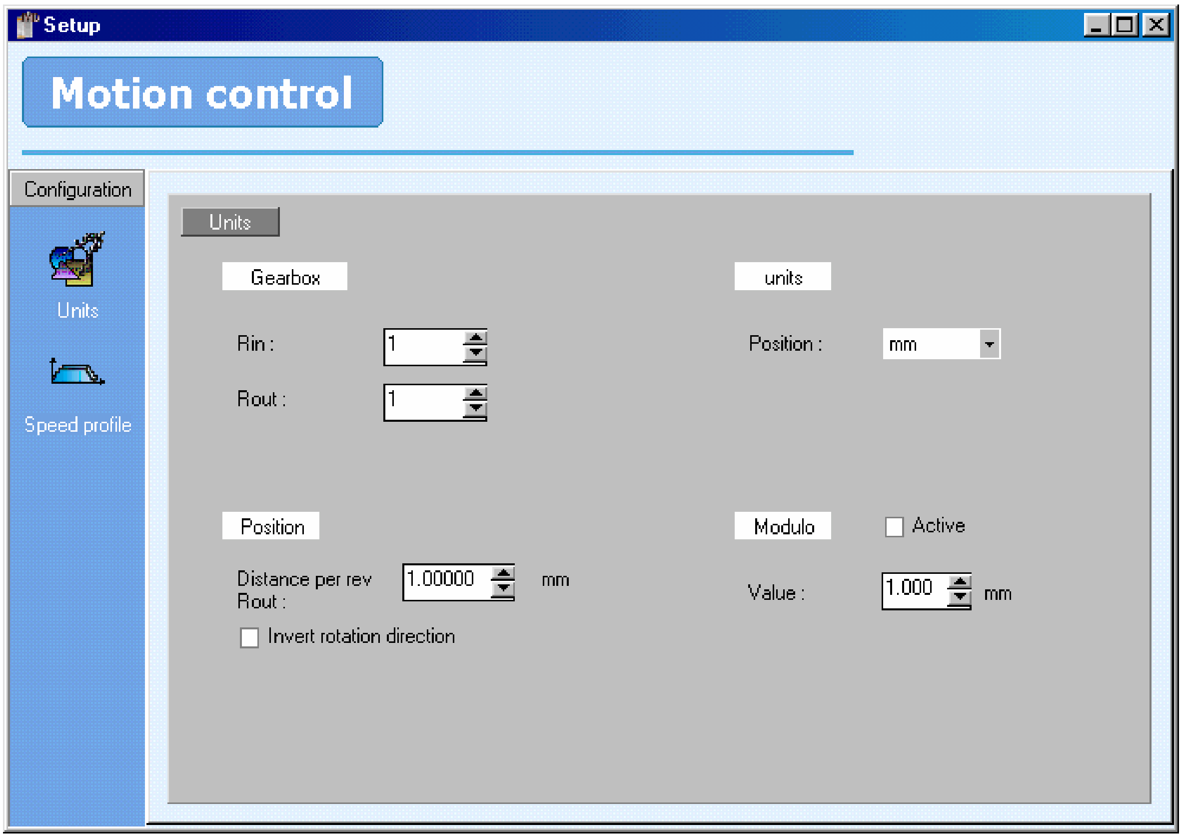


A) Configuration :

Icon : 

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

- Units :



Example 1 : Linear axis

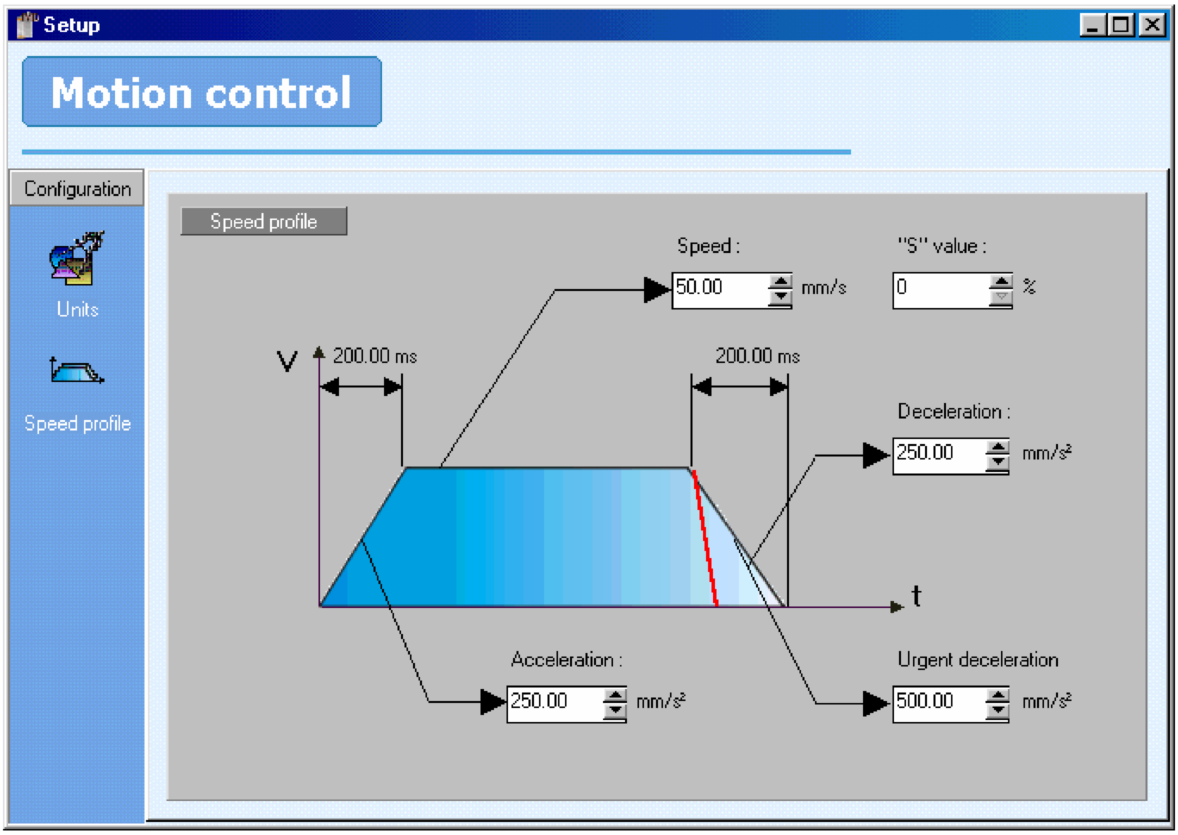
Motor connected to leadscrew with 5mm pitch.
Units = mm, Rin = 1, Rout = 1, Distance per rev = 5.000, Modulo not active.

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 iDPL*

- Speed profile :



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

The urgent deceleration is used to stop axis when limit sensors are active.

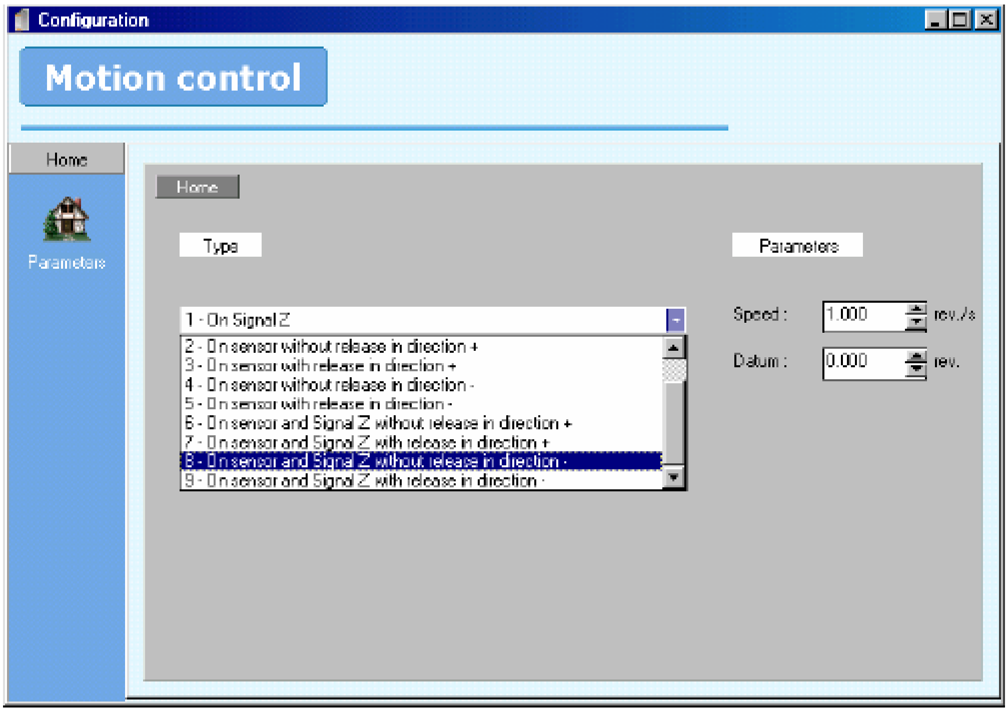
S coefficient parameter allows having acceleration and deceleration with a S form that softens the start and end of a movement. Acceleration with S coefficient is between 0 and 200% of acceleration parameter.

Warning: These parameters are relative to the mechanic of the system, not the motor.

B) Home :

Icon : 

Action : Configure the homing mode.

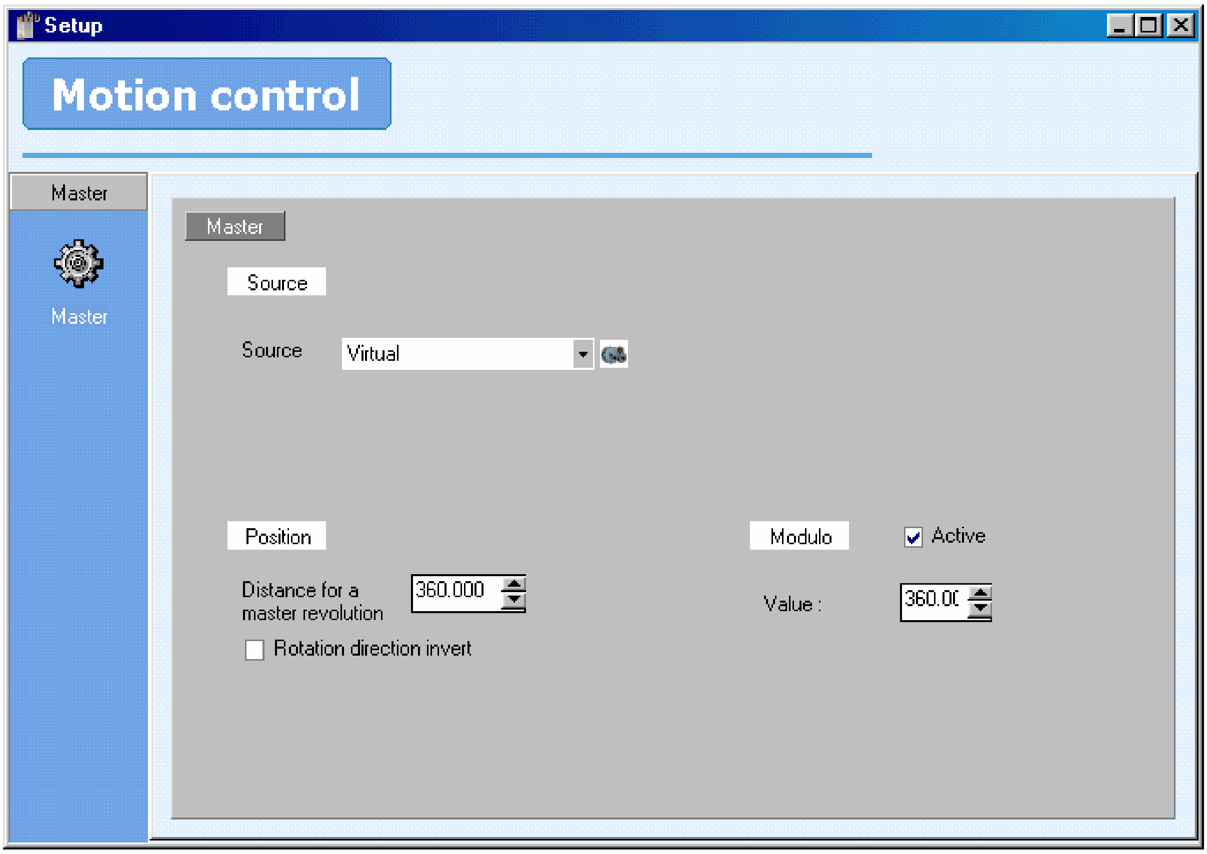


- Homing method.
- Homing speed.
- Home position (0 by default)

C) Master encoder :

Icon : 

Action : Configure the master encoder.



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

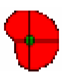
D) Trajectories :

Icon : 

Action : Launches trajectories selected by the digital inputs.

See section on trajectory definition.

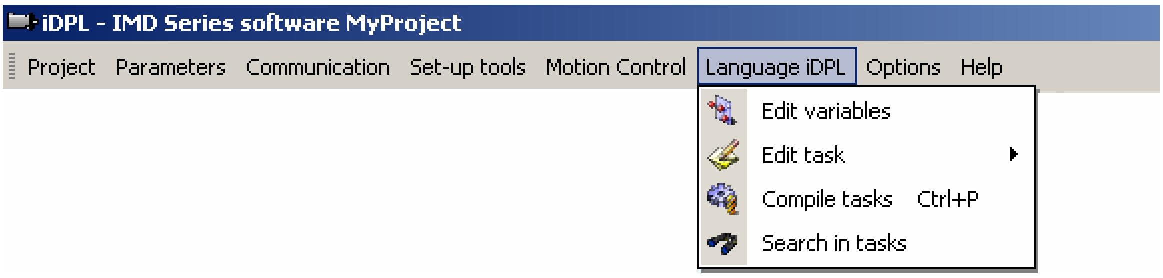
E) Cam editor :

Icon : 

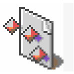
Action : Edit a Cam profile.

See section on cam

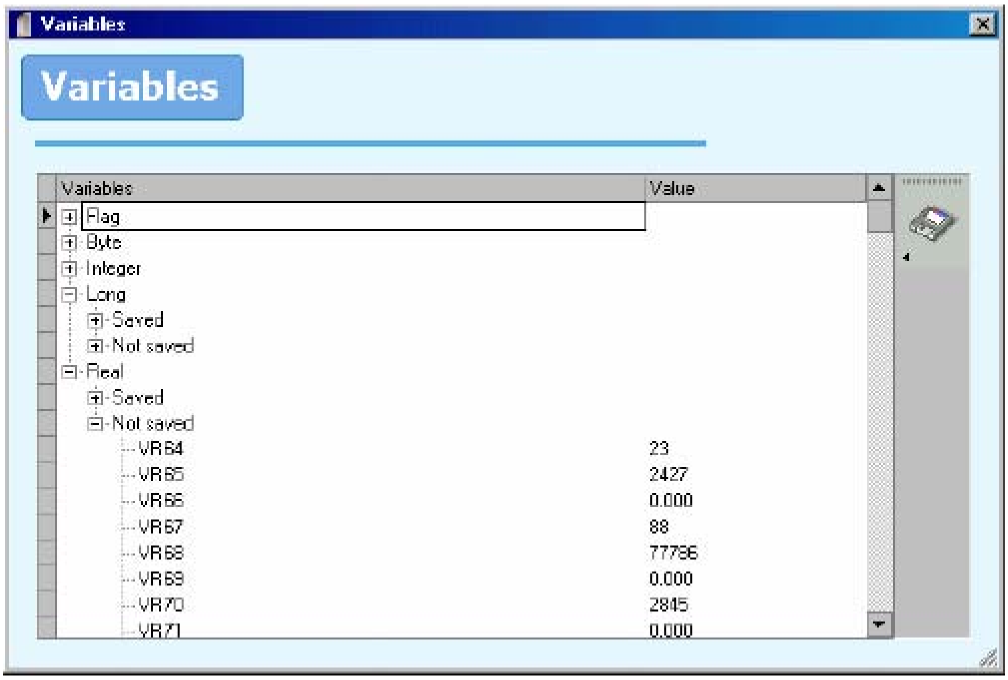
3-3-6- iDPL language



A) Edit variables :

Icon : 

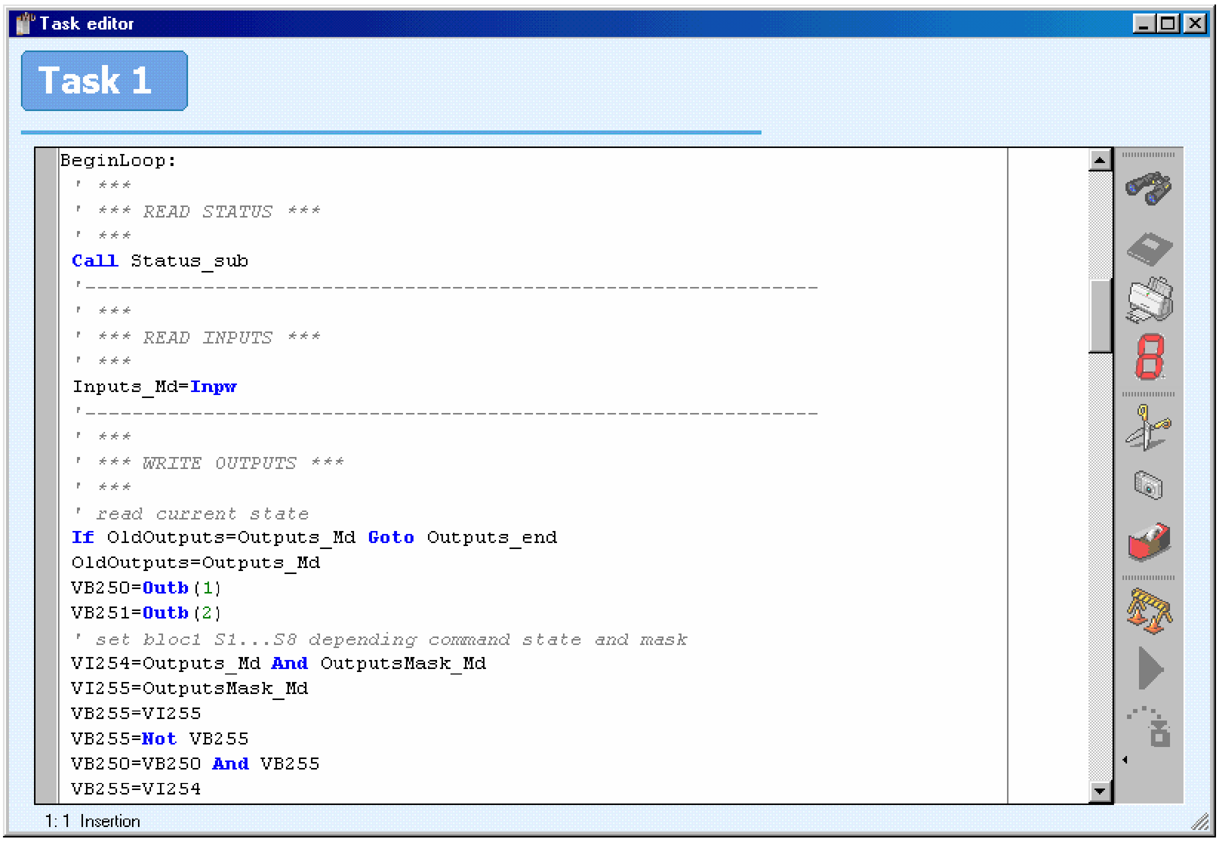
Action : Examine and modify variables (contained in the drive file dpv) and send these to the drive using the command *Communication / Variables iDPL / Send variables.*



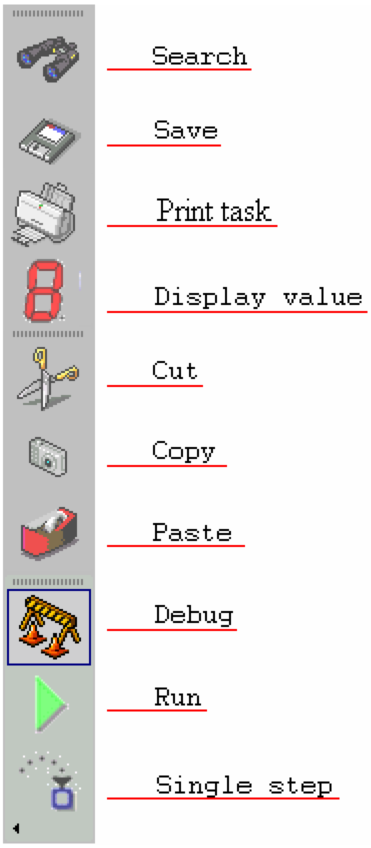
B) Edit a task :

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 :



C) Compile tasks :

Icon : 

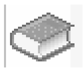
Action : Compile the tasks

D) Search tasks :

Icon : 

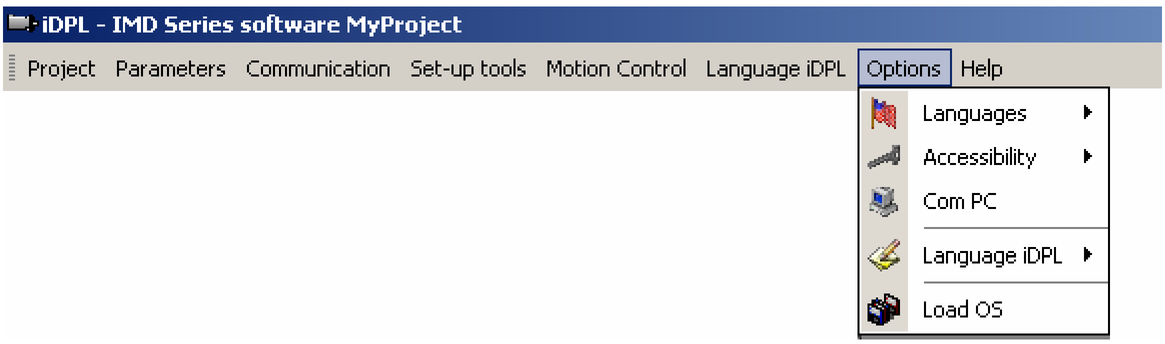
Action : Allows the user to search for a string of characters in the tasks.

E) Information :

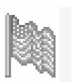
Icon : 

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

3-3-7- Options



A) Languages :

Icon : 

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

B) Accessibility :

Icon : 

Action : Selects the access level to the various parameters. :

- Standard parameters
- Advanced parameters
- Restricted parameters

Select or de-select the iDPL 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.

C) Com PC :

Icon : 

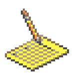
Action : Select the PC communication port : COM1, COM2, COM3 or COM4.

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.

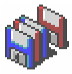
D) iDPL language :

Icon : 

Action : Access the iDPL programming options.

- Precision : defines the number of decimal places used for real numbers. Variables (VR0 to VR63), position (POS_S in iDPL) 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.

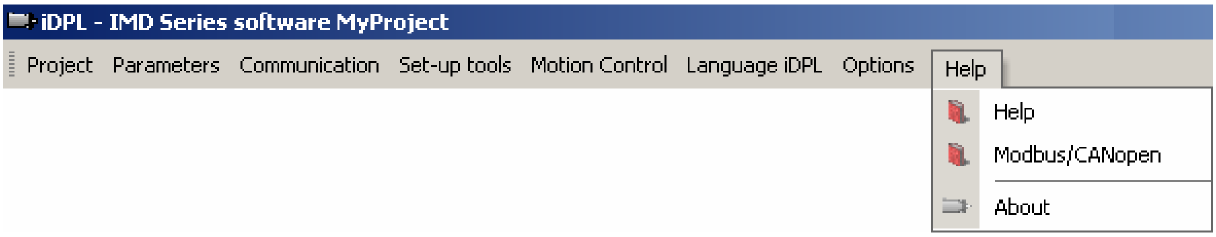
E) Operating system :

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.**

3-3-8- Help

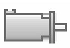


A) Help :

Icon : 

Action : Access the help files.

B) About :

Icon : 

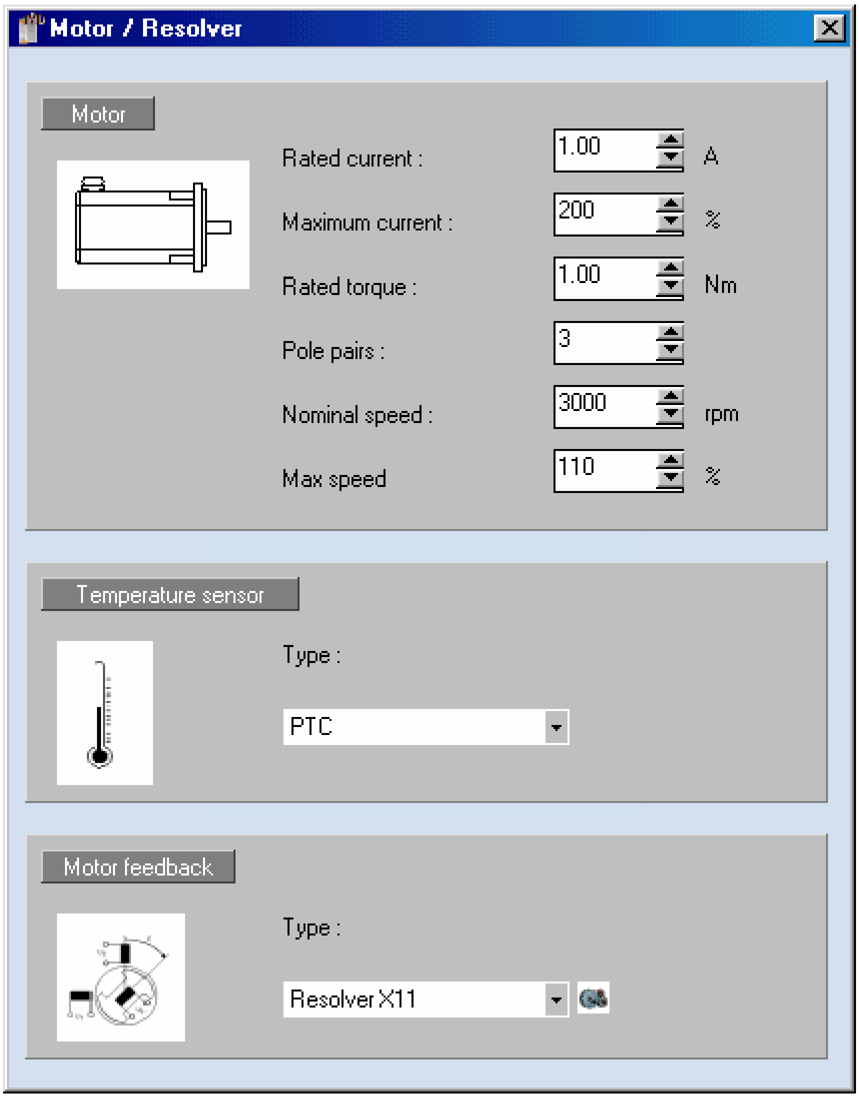
Action : Displays the current version of the software and drive firmware.

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.

- Select drive nominal voltage in the parameter windows. For each value, security parameters change (brake resistor, over voltage ...)
- If not, the parameters can be adjusted by selecting the menu Parameters/motor resolver. The following menu is displayed :



A) 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.

B) Feedback adjustments :

- Select feedback : Resolver or SinCos

a) Resolver :

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 **control panel** in the **diagnostic tools** menu.
3. Check that the position is increasing correctly when you turn the motor.
4. Open the **oscilloscope** in the **diagnostic tools** menu.
5. Select the signals SINE and COSINE in RESOLVER then start the data acquisition.
6. 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 and –0.5), contact our technical department.
7. Execute the feedback auto tuning.

b) SinCos :

1. Enter SinCos resolution and serial link
2. Open the **control panel** in the **diagnostic tools** menu.
3. Check that the position is increasing correctly when you turn the motor.
4. Execute the feedback auto tuning.

c) Feedback 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.**

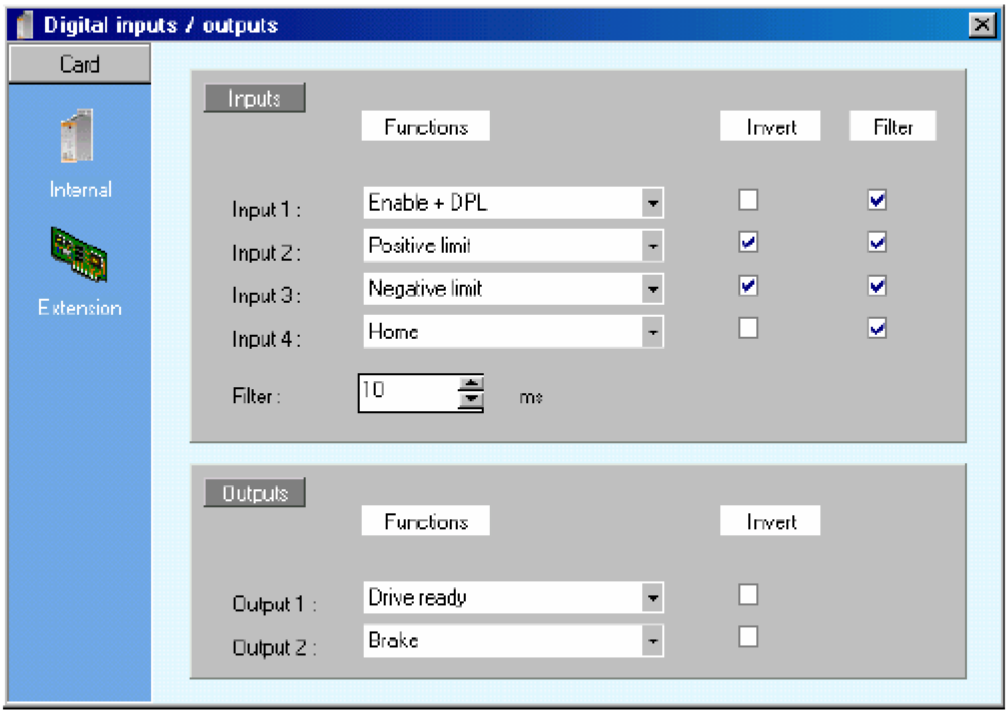
The drive will energise the motor windings and automatically measure the resolver offset. This step lasts only a few seconds.

4. Close the parameter window.
5. Save the parameters.

4-2- 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 field 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 can now be used to enable and disable the drive.

- If the motor had a brake, select brake function to Input 2 (verify that a diode is connect to brake pin to protect electric components)
- Save the parameters

4-3- Operating modes

The iMD 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.

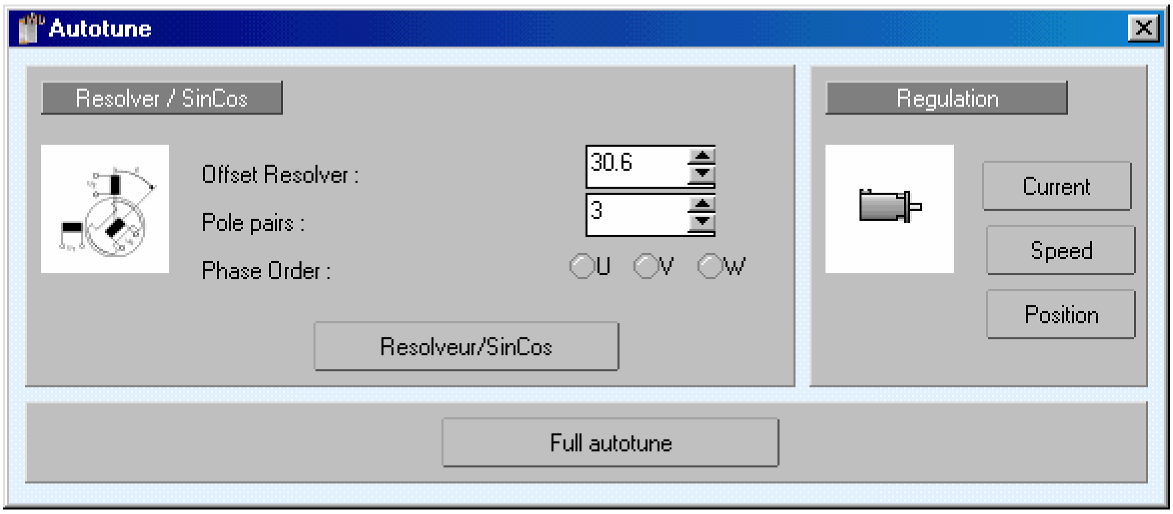
In position mode, the motor follows the demanded trajectory.

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.

4-4- Automatic control loops adjustement

4-4-1- Auto tuning of the control loops



A) Current loop auto tuning :

During this phase, the motor makes small movements and calculates vibration limits then long movement depending on inertia.

Warning : it is possible to make this phase with or without motor load (except if the mechanics are weak).

B) Speed loop auto tuning :

During this phase, the motor turns at average velocity.

Warning : The axis must be a rotary axis because the number of turns is unknown. It is preferable to make this adjustment with the motor load for good stiffness.

C) Position loop auto tuning :

During this phase, the motor turn with a small velocity.

Warning : it is possible to make this phase with or without a motor load.

D) Complete auto tuning :

Executes all tuning routines.

E) Auto tuning precautions :

During auto tuning, all securities are actives (I²t etc ...)

To reduce or cancel overshoot at the beginning and end of a move, set 0 in acceleration compensation in speed loop (following error will grow during acceleration and deceleration phases).

For a better system stiffness, increase the proportional gain of the speed loop.

For a better system time response, increase the integral gain of the speed loop.

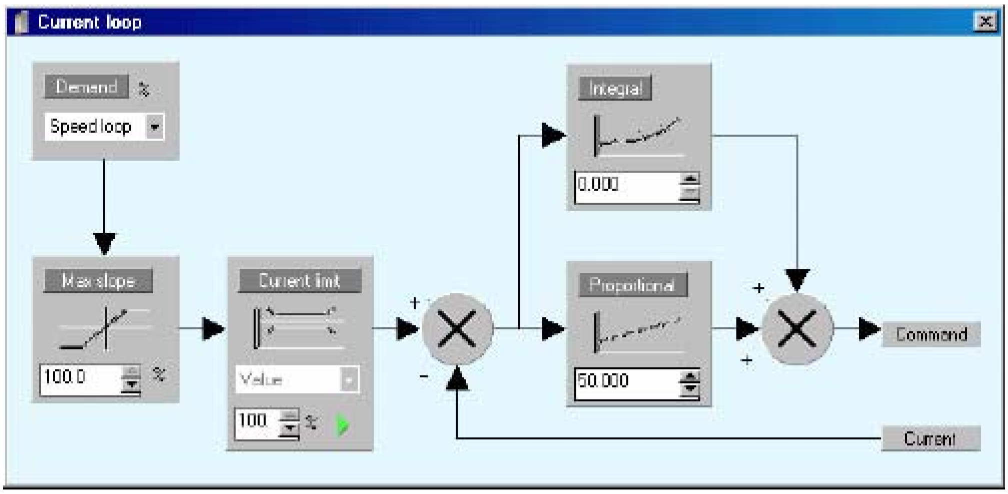
If system becomes unstable, reduce or cancel the integral gain of the speed loop.

4-5- Manual control loop adjustements

4-5-1- 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.

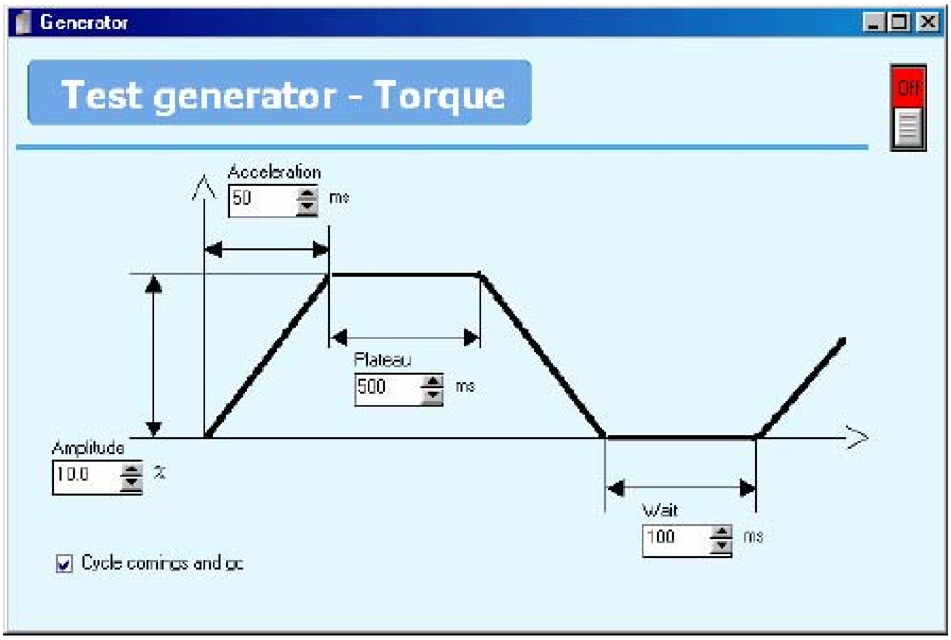
- Disable the drive (*Enable* button OFF in the main window).
- Select torque mode in the main window.
- Select the menu Parameters / Current loop. The following menu appears :



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

The command source must be of type : value.

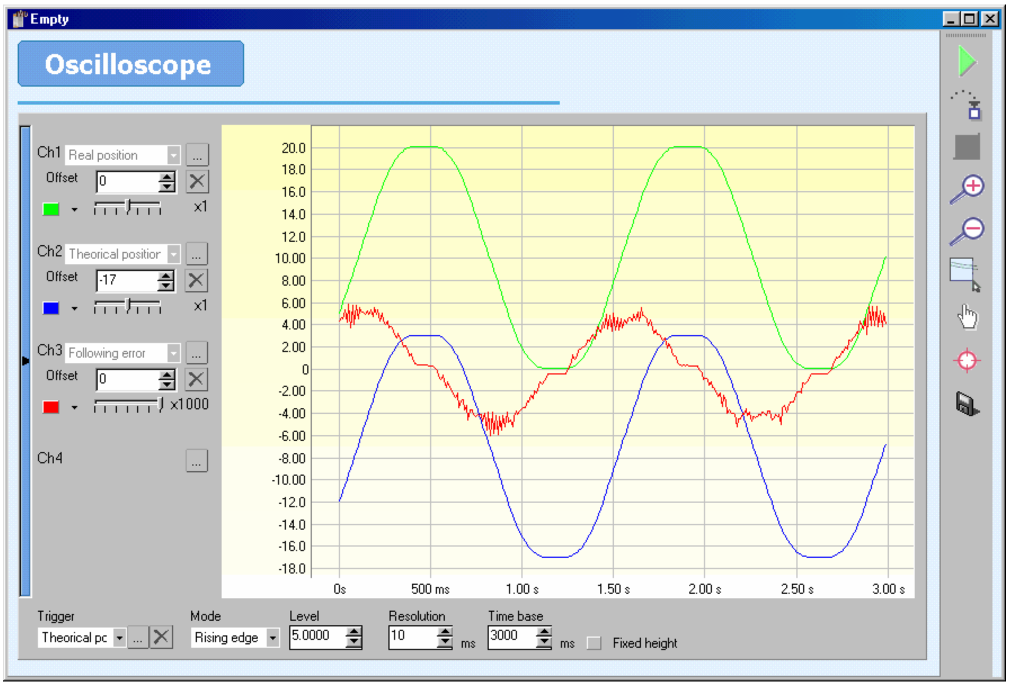
- In **Diagnostic tools / Generator**, start a movement as shown below :



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.

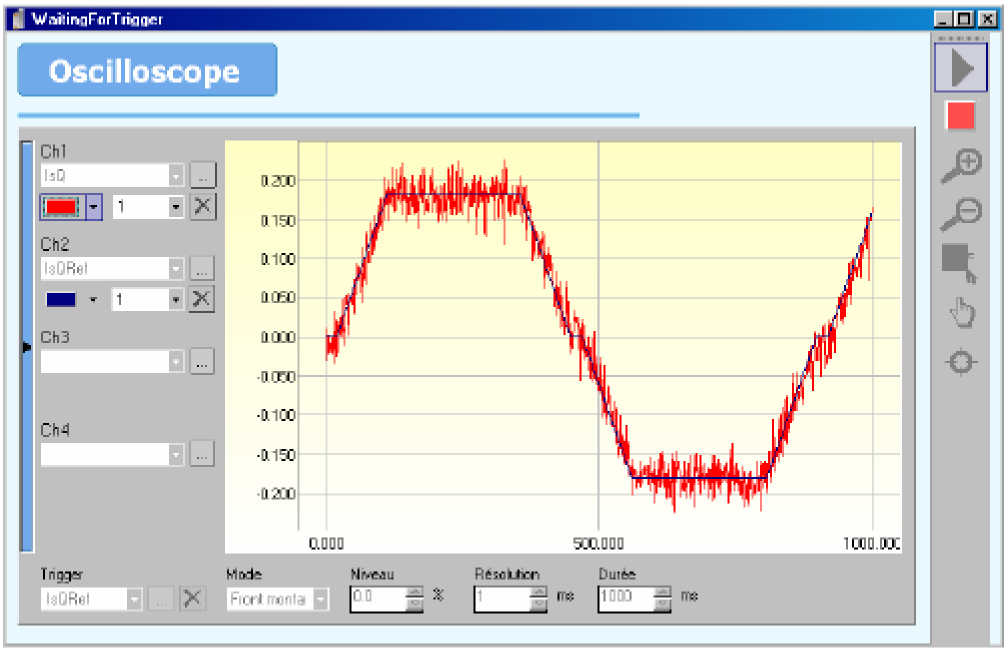
If the signal IsQREF is not trapezoidal, adjust the generator parameters.

- 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.

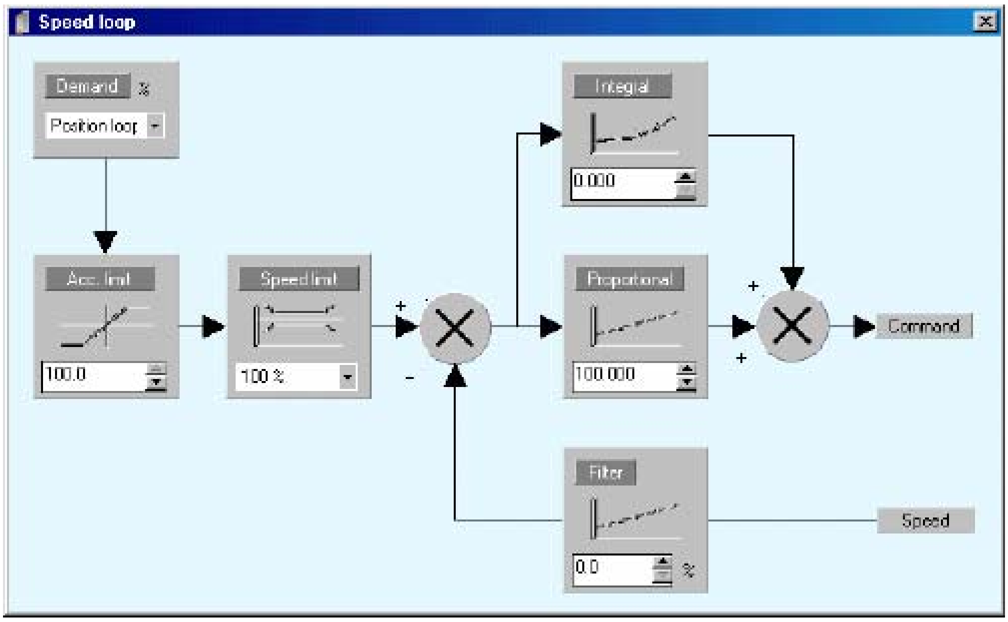
Typical curves for optimised gains.



- Save the adjustments using Parameters/Save parameters.

4-5-2- 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

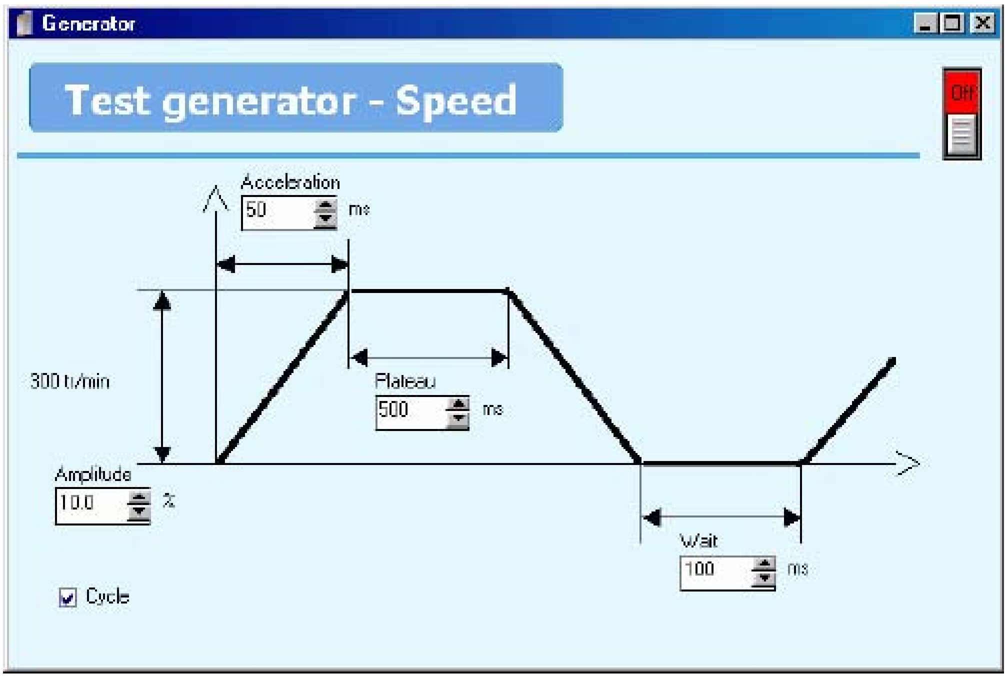


To start the speed loop adjustments use the values shown above.

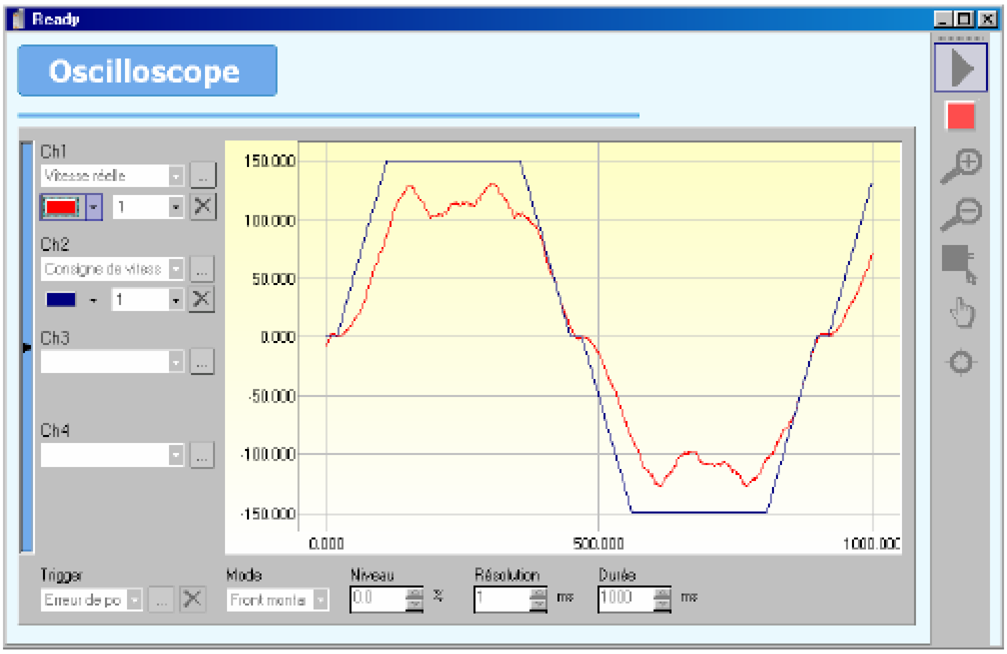
The command source must be of type : value

- 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.



- 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.

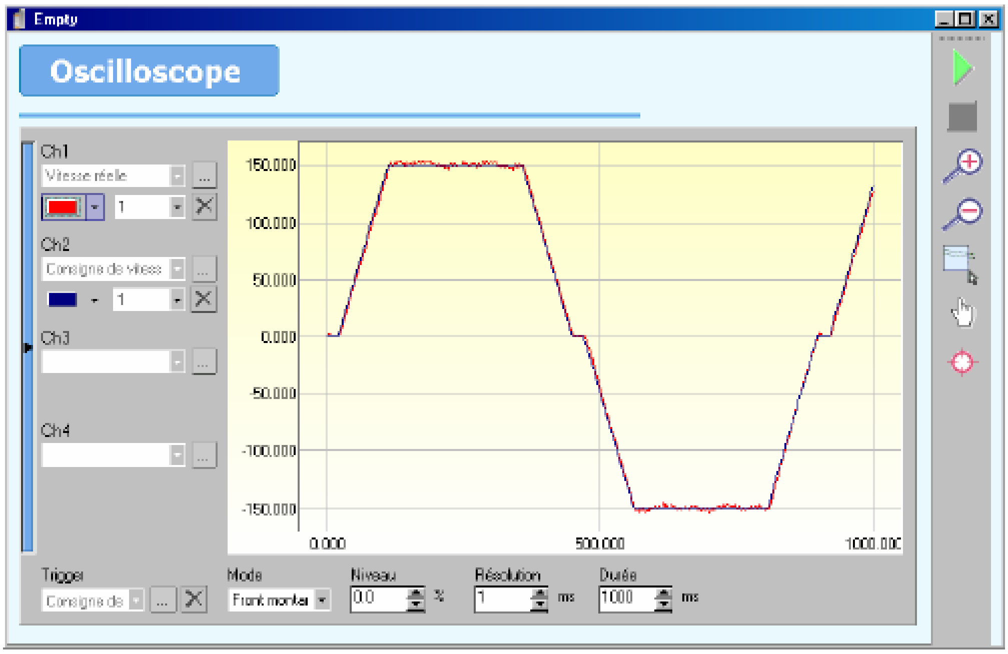
- Increase the proportional gain until the actual speed is as close as possible to the command.

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.

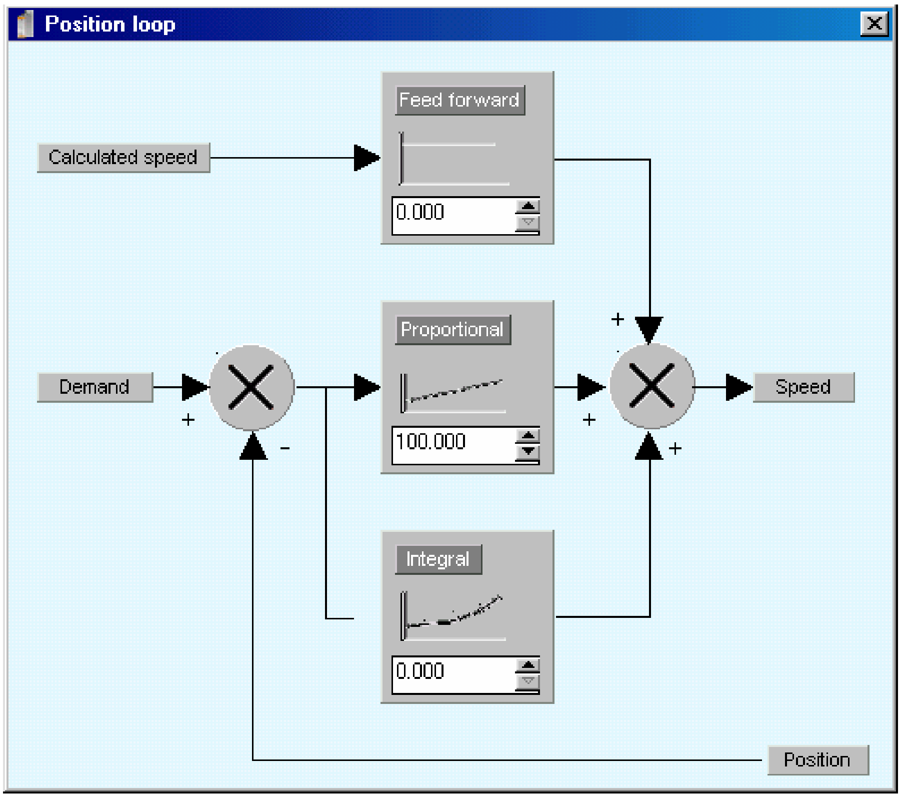
Typical curves for optimised gains.



- Save the adjustments using Parameters/Save parameters.

4-5-3- Position loop adjustment

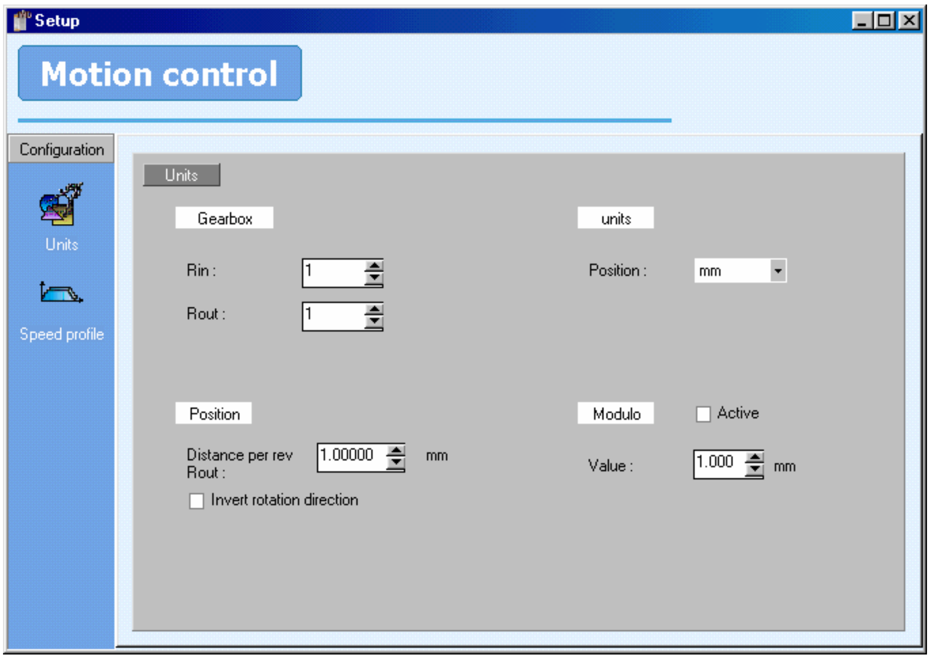
- Disable the drive (***Enable*** button OFF in the main window).
- Select position mode in the main window.
- Select the menu Parameters / Position loop

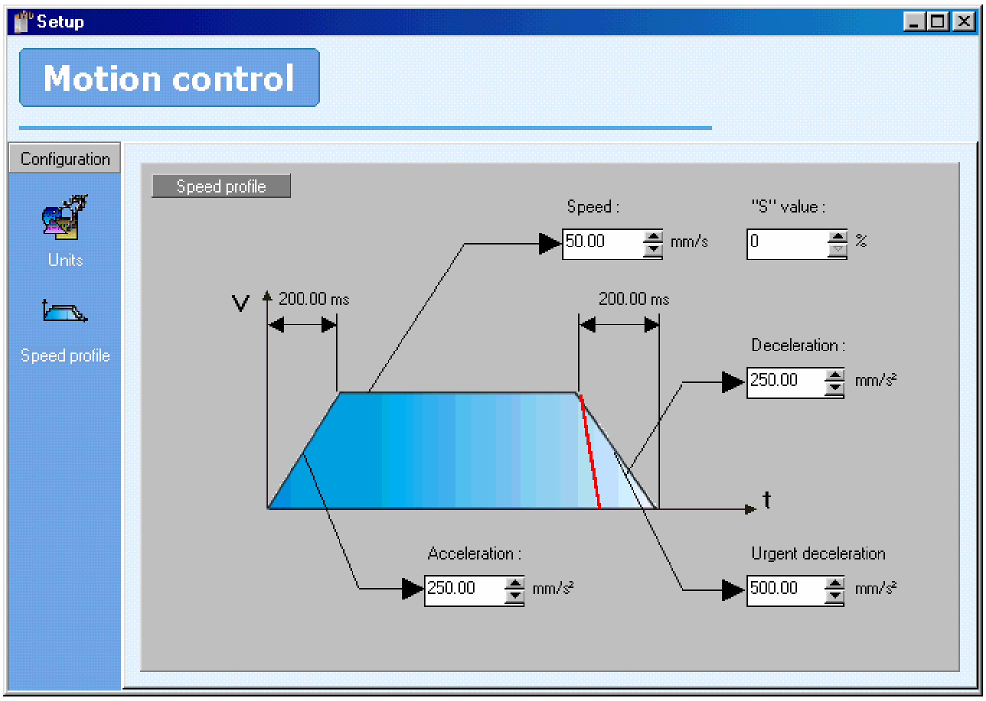


To start the position loop adjustments use the values shown above.

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

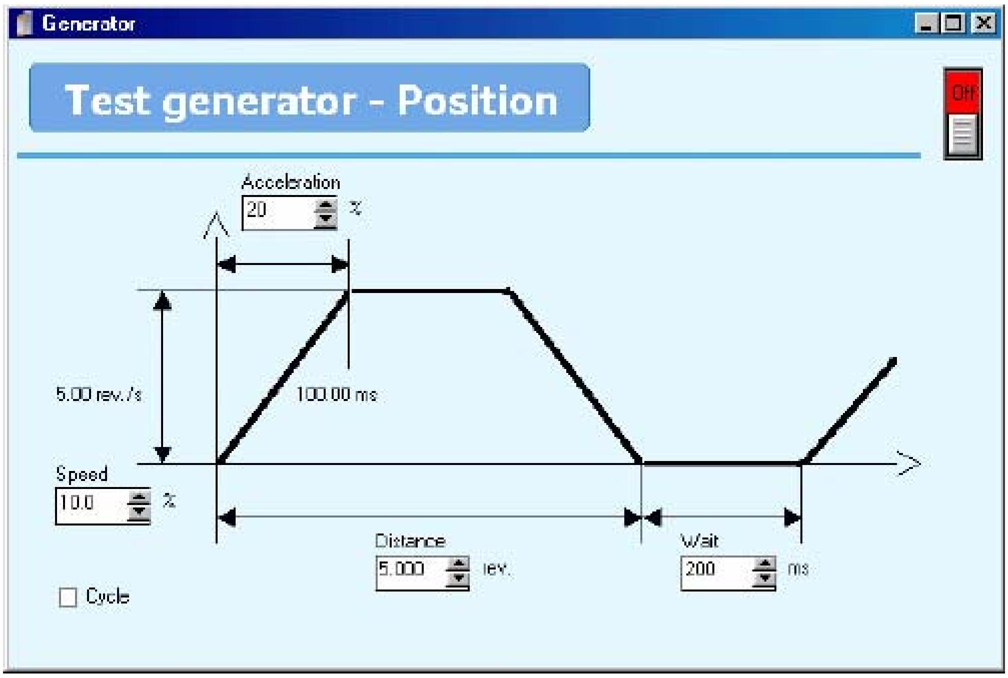
The percentage speed and acceleration used in the generator window are referenced to the values in the menu **Motion control / Configuration / Speed profile.**



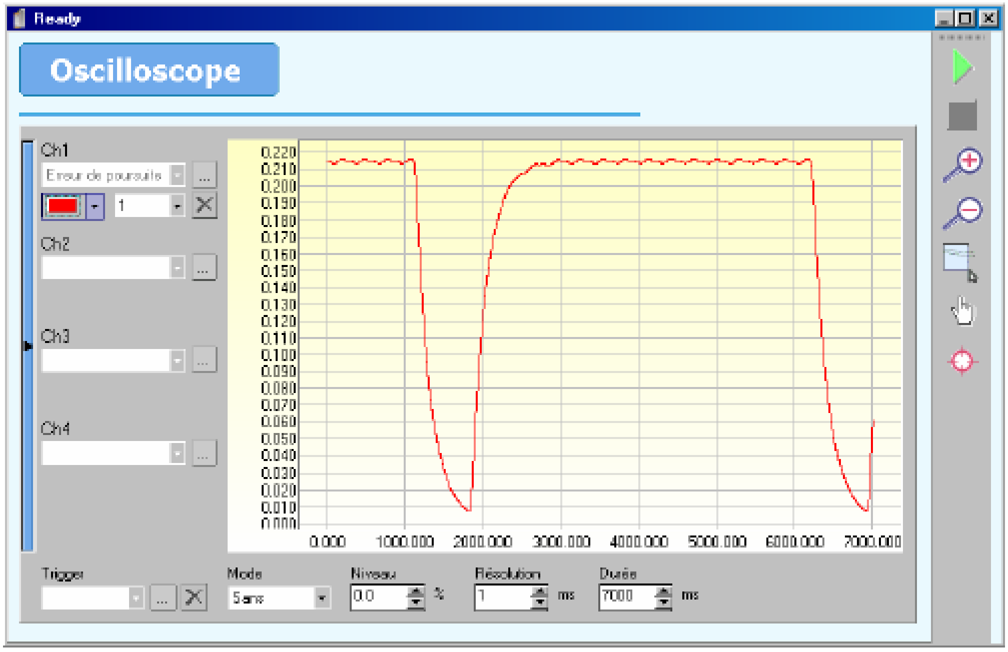


 According to the characteristics of the motor, set the following error in Parameters / Supervision / Position / Following error

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



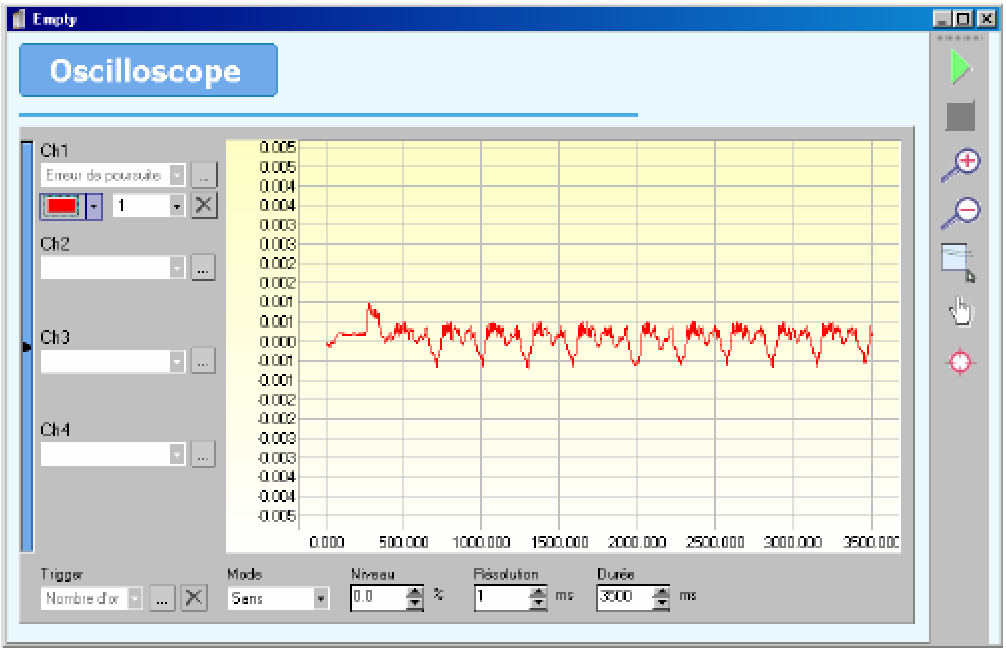
- Use Diagnostic tools / Oscilloscope to observe the following error during the movement :



- 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 reduces 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.

4-6- Other adjustements

4-6-1- Speed loop operation

1. Select speed mode
2. In parameters \ speed loop \ Demand, select analogue input.
3. In parameters \ analogue inputs outputs, verify that analogue scale 1 is 100% (for a ±10V demand)
4. In parameters \ motor parameters, enter nominal motor speed and 110% to the maximum speed.
5. In parameters \ supervision \ DC bus, enable under voltage
6. In parameters \ multifunction encoder output, select bypass mode and enter source and resolution of the feedback.

4-6-2- Double loop operation

1. Select position mode
2. In parameters \ regulation \ loop type, select double and then setup position loop feedback

   Ex : for incremental encoder : select resolver X11 then set the resolution

4-6-3- Stepper input operation

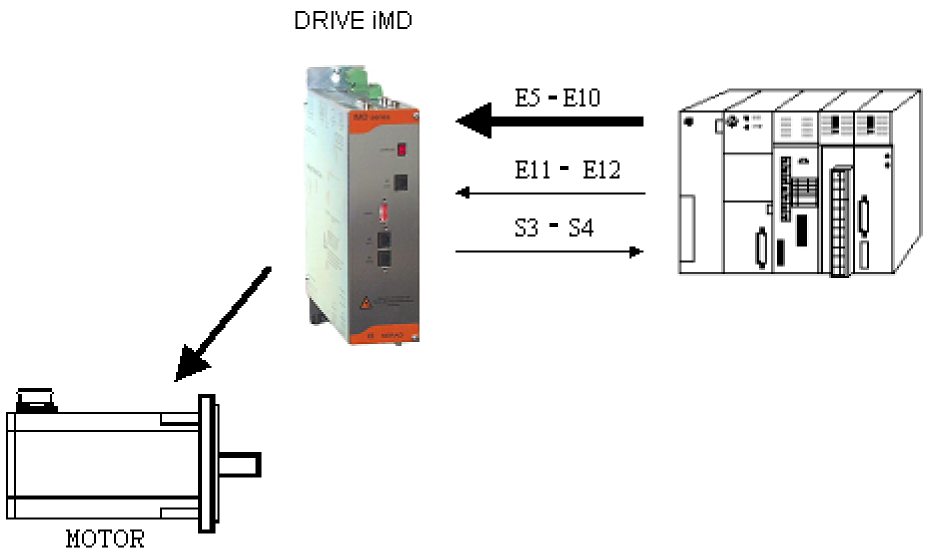
1. Select position mode
2. In motion control \ master-slave, select multifunction encoder input as source and setup the input for stepper mode.
3. Create a new task with gearbox and startgearbox for stepper electronic gearbox.

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.

Trajectories can also be controller by Modbus or CANopen communication.



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 iDPL 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.

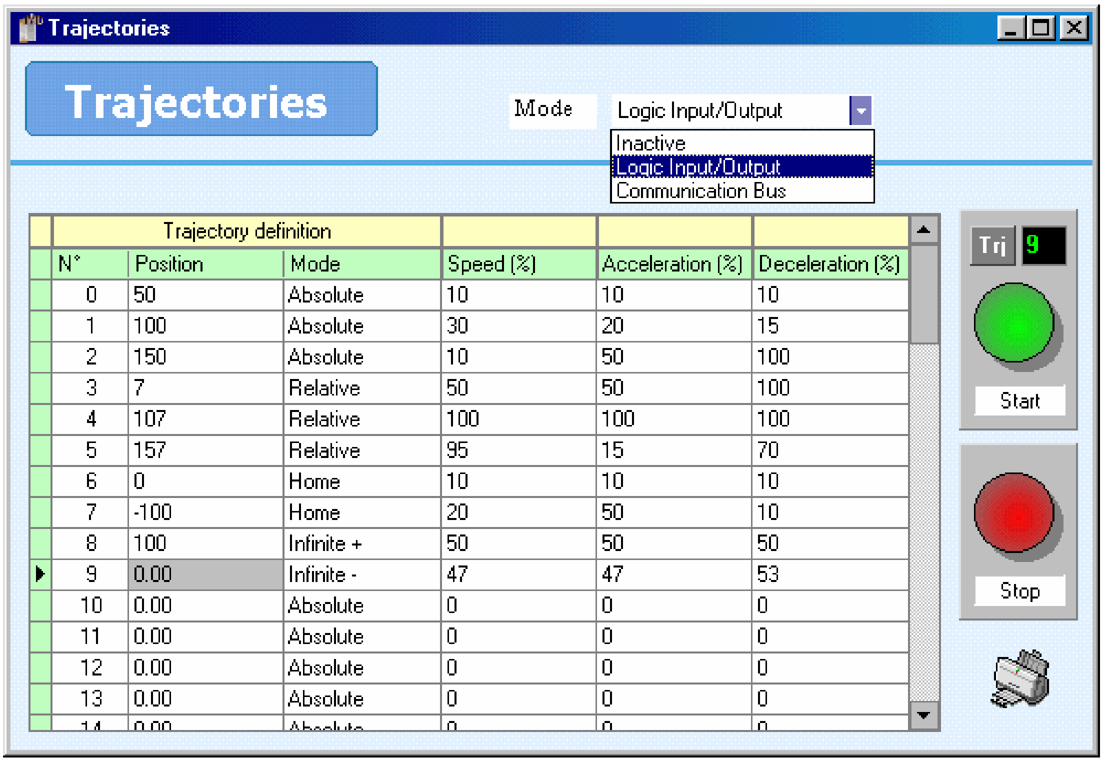
5-2- Trajectories using I/O card

5-2-1- Implementation

a) 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.



- Select mode to use 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.

Make a HOME by trajectories:

1. Declare a trajectory

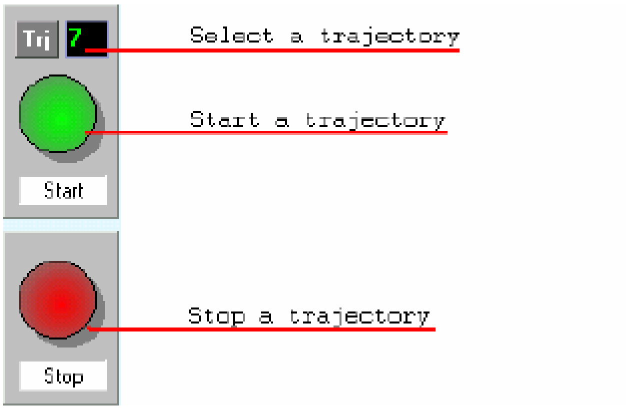
2. Setup home datum in **Motion Control / Home**

3. Setup input 4 as Home function in **Parameter \ Digitials inputs/outputs** (if you use sensor)

Save the trajectories with Communication / Trajectories / Save trajectories.

b) Simulate trajectories :

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



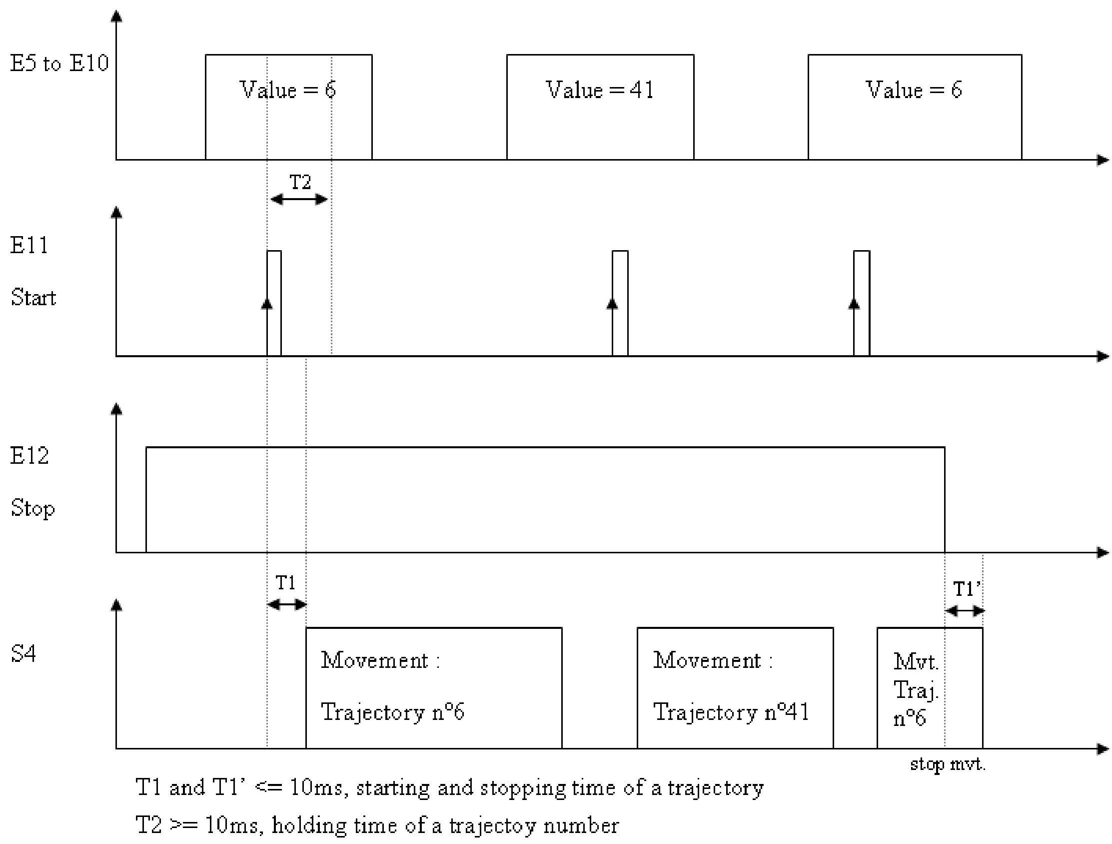
1. Verify that the drive is enabled and that the 'Active' box is selected.
2. Select the number of the trajectory to execute.
3. Press START to launch the trajectory.
4. Press STOP to stop the movement before the end.

c) 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.**

5-2-2- Operation

a) Timing:



b) 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.

c) 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 and VL0

The trajectory TRJ19 is coded using VR19 and VL19

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

1st byte : Mode (MS byte)

- 0 : absolute
- 1 : relative
- 2 : + infinite
- 3 : - infinite
- 4 : home

2nd byte : Speed (in %)

3rd byte : Acceleration (in %)

4th byte : Deceleration (LS byte) (in %)

5-3- Trajectories using communication bus

5-3-1- Implementation

It is possible to launch trajectories by communication bus using drive direct parameters. (see **Help \ Modsbus-CANopen** windows).

a) Control of a trajectory :

- _PARAM_TRAJ_ACTIF : activates the trajectories mode (set to 2)
- _PARAM_TRAJ_SELECTION : selects a trajectory (0 to 63)
- _PARAM_TRAJ_START : starts the selected trajectory
- _PARAM_TRAJ_STOP : stops an executing trajectory

b) 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 and VL0

The trajectory TRJ19 is coded using VR19 and VL19

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

  1st byte : Mode (MS byte)

  - 0 : absolute
  - 1 : relative
  - 2 : + infinite
  - 3 : - infinite
  - 4 : home

  2nd byte : Speed (in %)

  3rd byte : Acceleration (in %)

  4th byte : Deceleration (LS byte) (in %)

5-3-2- Operation

Example of trajectories by CANopen bus:

```
Prog
    'Demo Bitconnect CAN/ModBus/iDPL

    WriteParam(2800h,01h)=2
    WriteParam(6040h,00h)=0 'Disable drive
    wait (readParam(6041h,00h)=0)
    WriteParam(6040h,00h)=1 'Enable drive
    wait (readParam(6041h,00h)=1)

    WriteParam(2800h,04h)=0
```

```
' === HOME ===
VR0=0'WriteParam(3400h,00h)=0 'position 0
VL100=4               'mode : Home
VL100=VL100 << 8
VL100=VL100+0   'speed : 0

VL100=VL100 << 8
VL100=VL100+0   'acceleration : 0
VL100=VL100 << 8
VL100=VL100+0   'deceleration : 0
VL0=VL100          'WriteParam(3300h,00h)=VL100 ' options
WriteParam(2800h,02h)=1

repeat
    VI100=ReadParam(6510h,06h)
    VI100=VI100 and 2
until VI100<>0

VR0=-5      'WriteParam(3400h,00h)=-500 'position -5
VL100=0         'mode : Absolu
VL100=VL100 << 8
VL100=VL100+20   'speed : 20
VL100=VL100 << 8

VL100=VL100+100   'acceleration : 100

VL100=VL100 << 8
VL100=VL100+100   'deceleration : 100
VL0=VL100          'WriteParam(3300h,00h)=VL100 ' options
```

```
WriteParam(2800h,02h)=1

repeat
    VI100=ReadParam(6510h,06h)
    VI100=VI100 and 1
until VI100=0

VR0=-1      'WriteParam(3400h,00h)=-100 'position -1
VL100=1         'mode : Relative

VL100=VL100 << 8
VL100=VL100+10   'speed : 10
VL100=VL100 << 8

VL100=VL100+100   'acceleration : 100
VL100=VL100 << 8
VL100=VL100+100   'deceleration : 100
VL0=VL100          'WriteParam(3300h,00h)=VL100 ' options
WriteParam(2800h,02h)=1
repeat
    VI100=ReadParam(6510h,06h)
    VI100=VI100 and 1
until VI100=0

VR0=2.5    'WriteParam(3400h,00h)=250 'position 2.
VL100=0         'mode : Absolu
VL100=VL100 << 8
VL100=VL100+30   'speed : 30
VL100=VL100 << 8
```

```
VL100=VL100+100   'acceleration : 100

VL100=VL100 << 8
VL100=VL100+100   'deceleration : 100
VL0=VL100'WriteParam(3300h,00h)=VL100 'options

WriteParam(2800h,02h)=1
repeat
    VL100=ReadParam(6064h,00h)
    VR100=VL100
    VR100=VR100/100
until VR100>0
WriteParam(2800h,03h)=1 'stop e movement
repeat
    VI100=ReadParam(6510h,06h)
    VI100=VI100 and 1
until VI100=0

VR0=0'WriteParam(3400h,00h)=0 'position 0
VL100=2         'mode : Infinite +

VL100=VL100 << 8
VL100=VL100+30   'vitesse : 30

VL100=VL100 << 8
VL100=VL100+100   'acceleration : 100
VL100=VL100 << 8
VL100=VL100+100   'deceleration : 100
VL0=VL100         'WriteParam(3300h,00h)=VL100 'options
```

```
    WriteParam(2800h,02h)=1
    delay (1000)
    WriteParam(2800h,03h)=1 'stop movement

    halt 1
EndProg
```

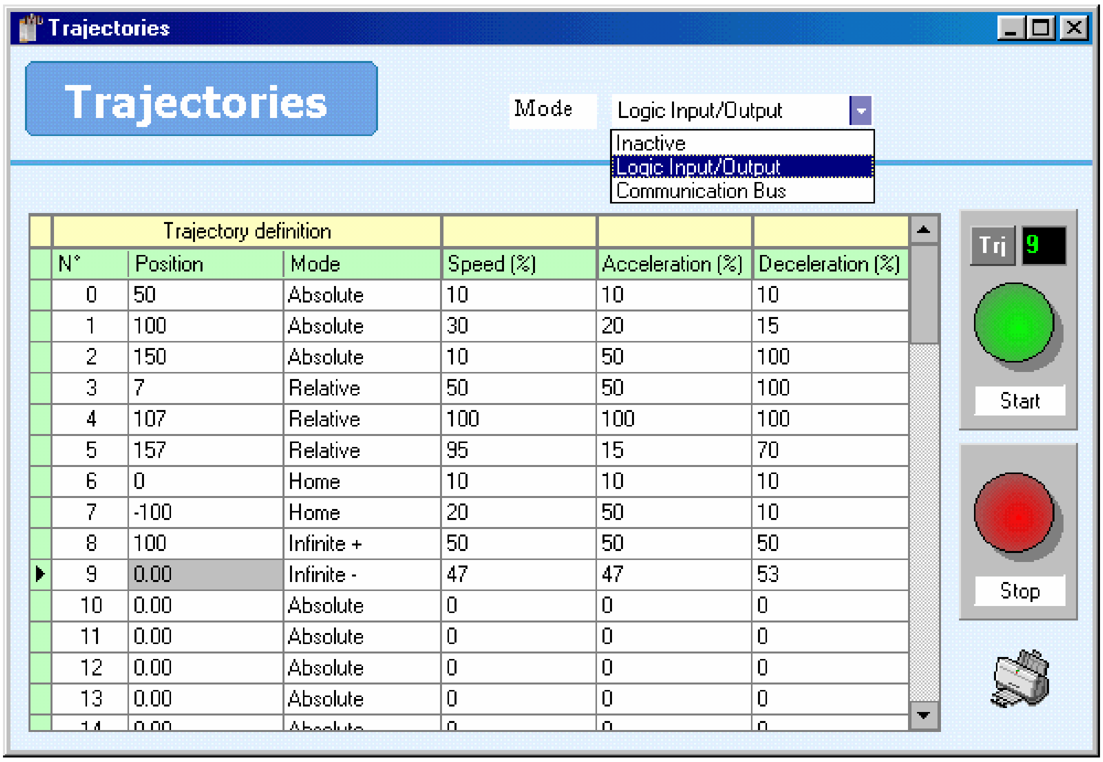
5-4- Advanced trajectories using I/O card

5-4-1- Implementation in advanced mode

a) 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.



- Select mode to use 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.

Make a HOME by trajectories:

1. Declare a trajectory

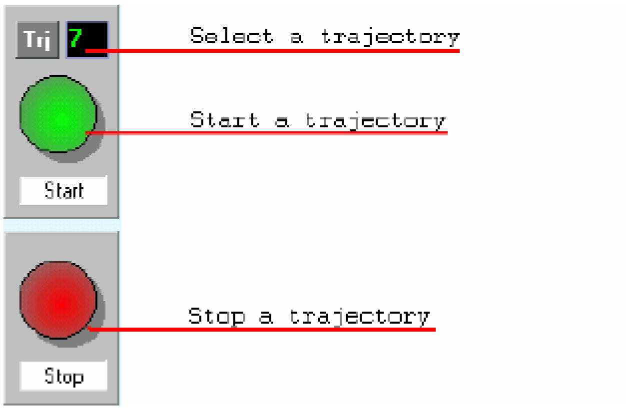
2. Setup home datum in **Motion Control / Home**

3. Setup input 4 as Home function in **Parameter \ Digitials inputs/outputs** (if you use sensor)

Save the trajectories with Communication / Trajectories / Save trajectories.

b) 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.

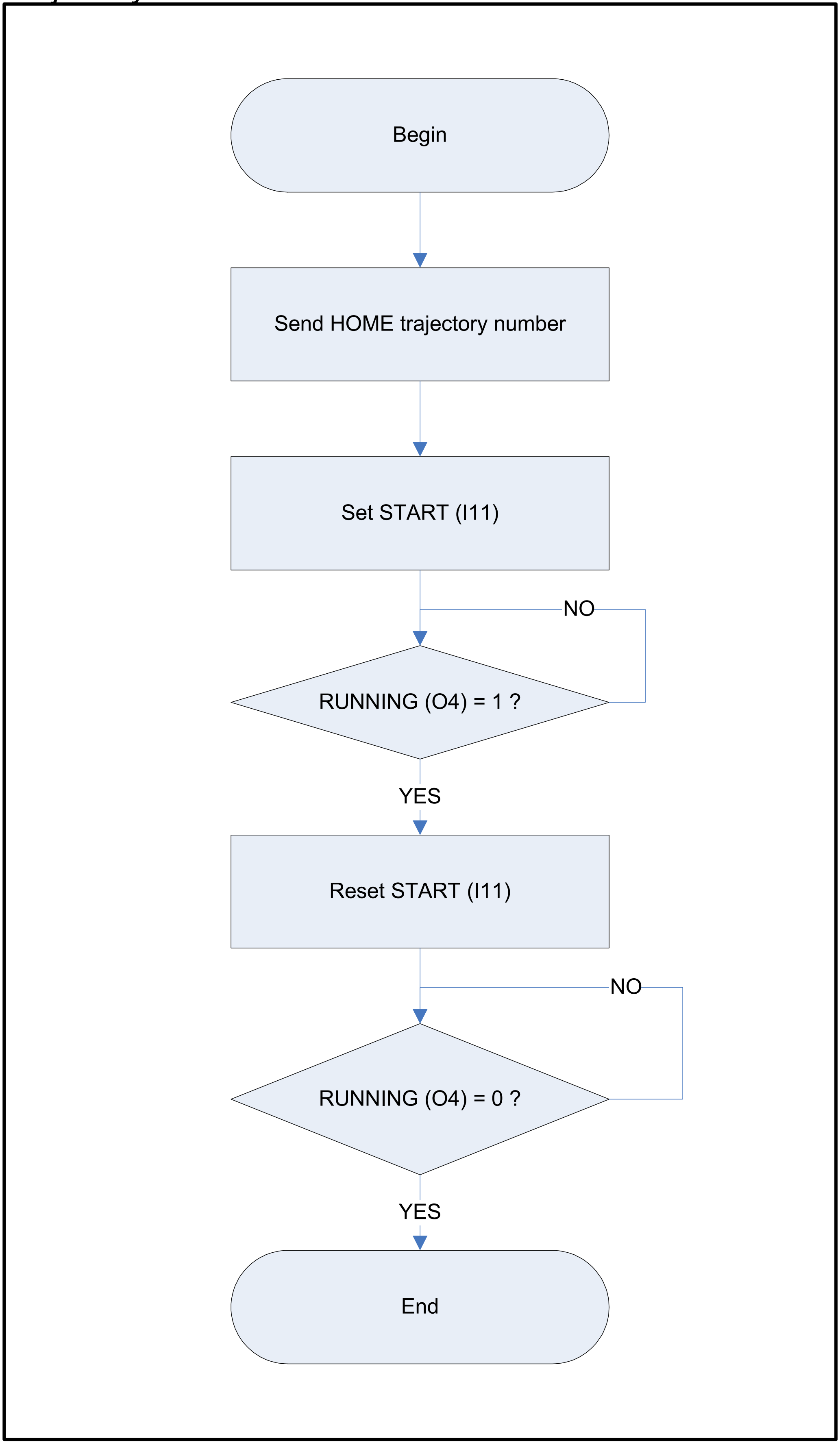
c) TRJ files :

- 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.**

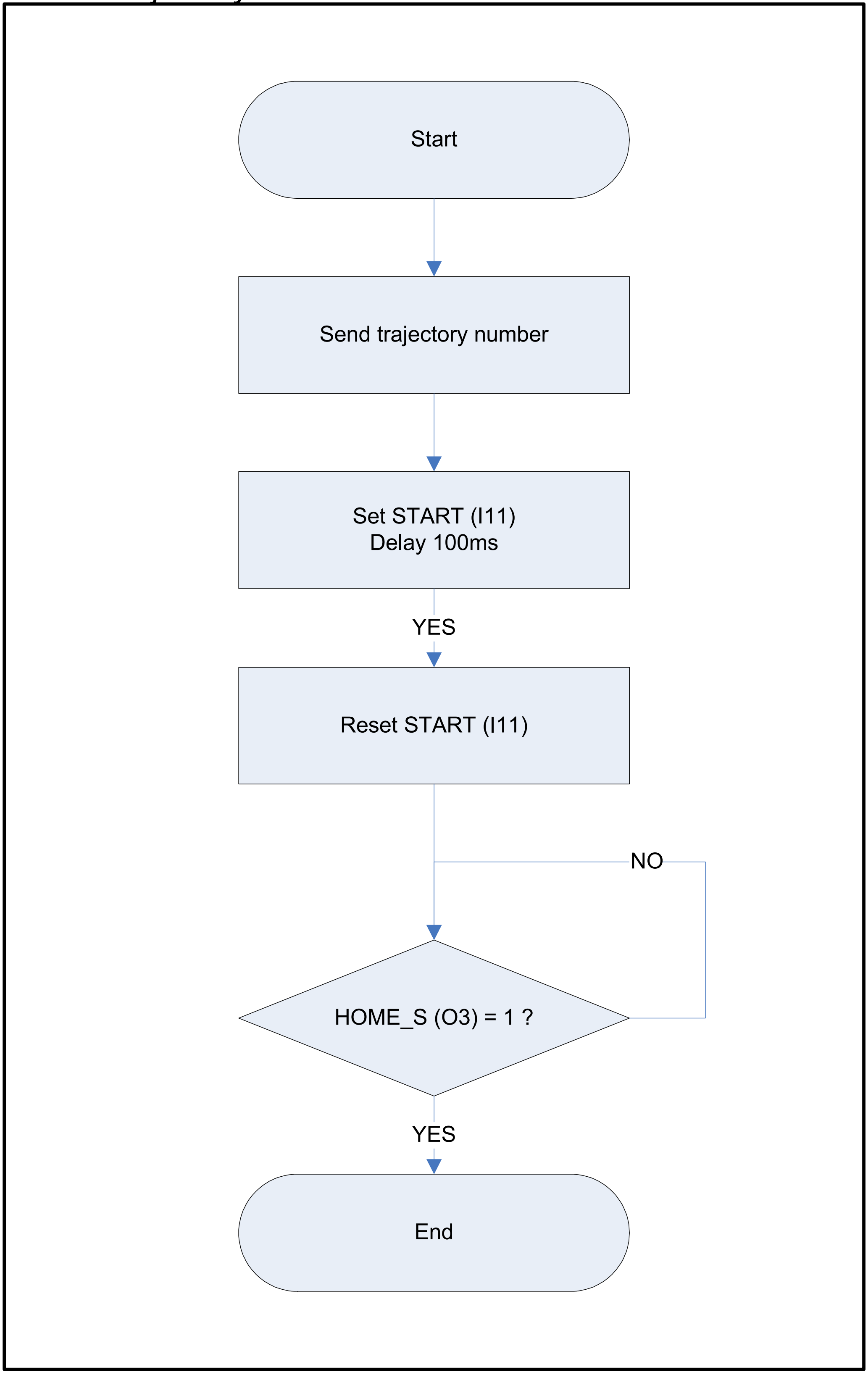
5-4-2- Operation

a) Flow chart:

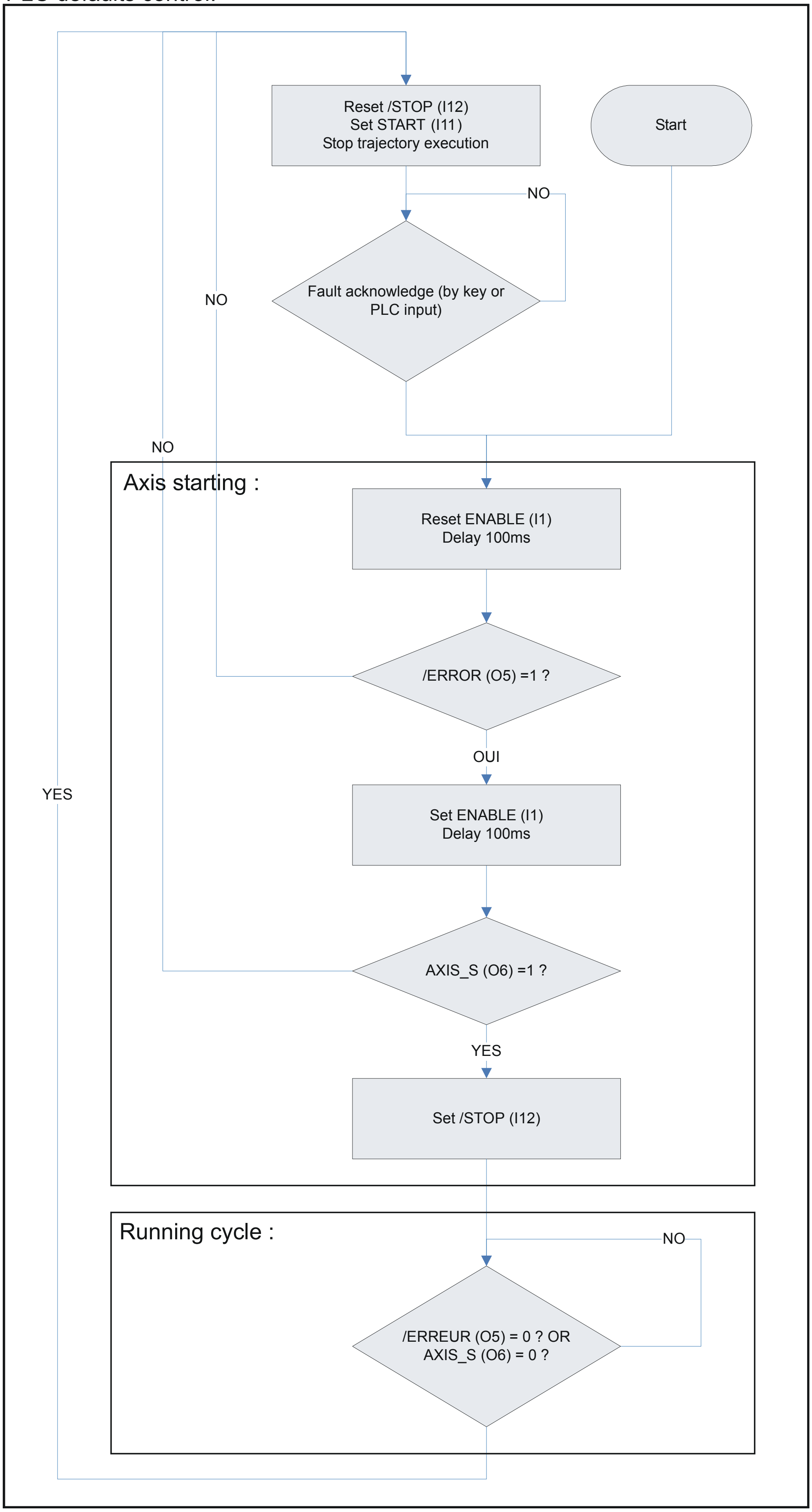
Trajectory execution :



HOME trajectory execution :



PLC defaults control:



b) Digitials I/O card:

Standard :

- Input 1: used to enable drive on positive edge and disable on low state (ENABLE). The input 1 must be setup as ENABLE function in Parameter \ Digital I/O.

Additionnal board

- 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.

c) 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 and VL0

The trajectory TRJ19 is coded using VR19 and VL19

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

  1st byte : Mode (MS byte)

  - 0 : absolute
  - 1 : relative
  - 2 : + infinite
  - 3 : - infinite
  - 4 : home

  2nd byte : Speed (in %)

  3rd byte : Acceleration (in %)

  4th byte : Deceleration (LS byte) (in %)

6- Programming language

6-1- Introduction

6-1-1- Introduction

- The language iDPL (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. iDPL comprises a real-time, multi-tasking kernel using pseudo-basic instructions supplemented by specific instruction for automation and motion control.

- iDPL supports various data variable formats.

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

- IMD drive has 4096 words of FRAM memory that allows the use of saved data or cams.

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 by means of Language iDPL / Declaration.

E.g: 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

All calculation must be of type: <Variable1> = <Variable2> <Expression> <Variable3 or Constant>

With <Variable1> same type as <Variable2> and <Variable3> lower or equal type as <Variable1>

Ex : VR0 = VR1 * 100

VR0 = VR1 * VR2

VL0 = VL0 * VB0

To change the value of variable in the screen Language iDPL / Edit variables:

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

Warning: Variable tables are only used for affectation

Eg 1: VR0 = VR0[VB1]

STTA = VR0

Eg 2: VR0 = VR2[VB2]

VL0 = VL2[VB3]

VR0 = VR0 * VL0

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 iDPL -> Compiler, the project must be recompiled after.

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 binary.

E.g. :	VB0=254	‘ decimal notation
	VB1=0FEh	‘ hexadecimal notation
	VB2=11111110b	‘ binary notation

6-2-4- Saved variables

Some global variables (VR0 to VR63, VL0 to VL63) can be saved and initialized at drive starting or drive restarting.

a) 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.

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

b) 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.

6-3- Saved data

6-3-1- Saved data

4096 words in FRAM:

FRAM memory advantage:

- No limit of writing or reading cycle.
- Data save after power cut.

Thanks to this characteristic, it is possible to use FRAM memory as saved area; it allows the saving of integer, long integer and real variables or cam tables.

N° de mot	Adresse	Fonction
Mot n°1	Adresse 0	Variable 1
Mot n°1	Adresse 1	Variable 2
Mot n°1	Adresse 2	Variable 3
	Adresse 3	Variable 4
Mot n°9	Adresse 8	
Mot n°10	Adresse 9	Came 1 – point 1
Mot n°11	Adresse 10	
Mot n°12	Adresse 11	
Mot n°13	Adresse 12	
Mot n°14	Adresse 13	
Mot n°15	Adresse 14	
Mot n°16	Adresse 15	
Mot n°17	Adresse 16	
Mot n°18	Adresse 17	
Mot n°19	Adresse 18	Came 1 – point 2
Mot n°20	Adresse 19	
Mot n°4095	Adresse 4094	Vide
Mot n°4096	Adresse 4095	Vide

A) Read/write an integer:

Read : WRITEI (<Address >) = <VIn or value >

Write : <VIn> = READI (<Address >)

Limits : < Address >: from 0 to 4095

n from 0 to 255

B) Read/write a long integer:

Read: WRITEL (<Address>) = <VLn or value>

Write: <VLn> = READL (<Address >)

Limits : < Address >: from 0 to 4095

n from 0 to 255

Warning: The reading and writing of a long integer needs 2 consecutive memory addresses (address n and address n+1).

C) Read/write a real:

Read: WRITER (<Address >) = <VRn or value >

Write: <VRn> = READR (<Address >)

Limits : < Address >: from 0 to 4095

n from 0 to 255

Warning: The reading and writing of a real needs 2 consecutive memory addresses (address n and address n+1).

D) Read/write cam table:

See the chapter Motion control programming \ Synchronization \ CAM

Check that the cam profile and saved data don’t use the same addresses otherwise your cam profile can change during moving.

6-4- Parameters

6-4-1- Parameters

It is possible in an iDPL task to change drive parameters (change mode, current limit ...), input functions, adjust regulation...

(see **Help \ Modsbus-CANopen** windows).

A) 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.

B) 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

C) SAVEPARAM - Save drive parameter

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.

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

D) LOADPARAM – Reload the drive parameters

Syntax : LOADPARAM

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

6-5- Tasks

6-5-1- 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 iDPL / 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.

6-5-2- Task priority

In an iDPL project you can have one high priority task that will be executed more often than the other tasks.

The priority task is allocated time slots as shown in the table :

Nb tasks	Execution time High task - Σ normal tasks	Example with task 1: high priority Task execution cycle
1	Nothing	1
2	75% - 25%	1 - 1 - 1 - 2
3	66% - 33%	1 - 1 - 1 - 2 - 1 - 3
4	62,5% - 37,5%	1 - 1 - 1 - 2 - 1 - 3 - 1 - 4

6-5-3- 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 initialization 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.

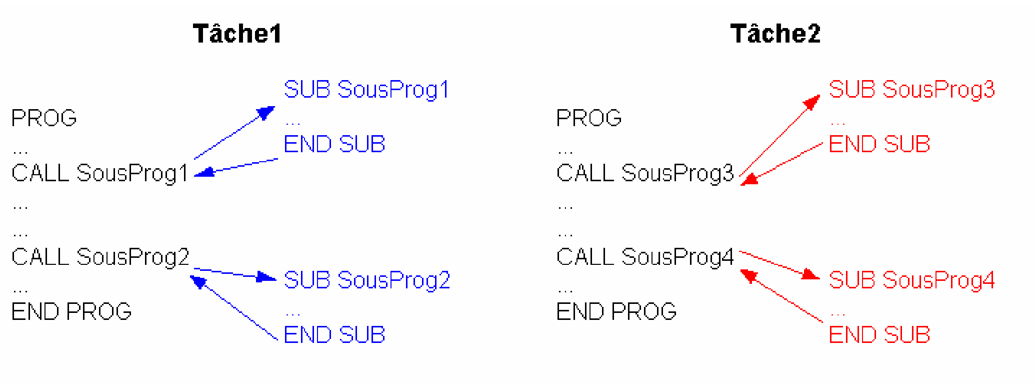
Example :

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	

6-5-4- 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 :



A) Main program

The main program of a task can call all of its subroutines but it can't 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.

B) 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
```

C) 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:
...
```

D) 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

a) 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>

b) 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

c) Unary operators :

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

d) Logic operators :

Operator	Operation	Operand type	Type
NOT	Binary negation	Byte, Integer	Operand type
AND	Binary AND	Byte, Integer	Operand type
OR	Binary OR	Byte, Integer	Operand type
XOR	Exclusive OR	Byte, Integer	Operand type
>>	Right shift	Byte, Integer	Operand type
<<	Left shift	Byte, Integer	Operand type

e) 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

f) Relationship operators :

Operator	Operation	Operand type	Result type
=	Equal	Byte, Integer, Long integer, real	Bit
<>	Different	Byte, Integer, Long integer, real	Bit
<	Lower	Byte, Integer, Long integer, real	Bit
>	Greater	Byte, Integer, Long integer, real	Bit
<=	Lower or equal	Byte, Integer, Long integer, real	Bit
>=	Greater or equal	Byte, Integer, Long integer, real	Bit

E) 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:

…

Or

IF <Expression> THEN

<Instruction1>

...

END IF

Or

IF <Expression> THEN

<Instruction1>

...

ELSE

<Instruction2>

...

END IF

<Expression> must have a bit type value. If <Expression> is true, the jump to <Label> is executed. If <Expression> is false, the 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:
IF 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.
…
```

F) REPEAT instruction

REPEAT instruction allows the repeated execution of one or more instructions in accordance to an expression value.

REPEAT instruction syntax is described below :

```
REPEAT
    <Instructions>
UNTIL <Expression>
```

In this instruction, if <Expression> is right before the REPEAT structure beginning, there is one loop. <Instructions> are executed unit <Expression> is right.

For example :

```
VEL%=100                 ' Fast velocity
STTA=2000                ' Start absolute move to position 2000
```

```
REPEAT
        VR0 = POS_S
        IF VR0 >1000 THEN
                VEL%=50             ' Slow velocity at middle distance
        END IF
UNTIL NOT MOV_S                     ' Loop until motor stop
```

7- Motion control programming

7-1- Introduction

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

The iDPL software contains numerous instructions associated with motion control :

positioning, electronic gearbox, superposition, synchronised movements etc.

The position counter can count up to ± 2 147 483 647 motor revs.

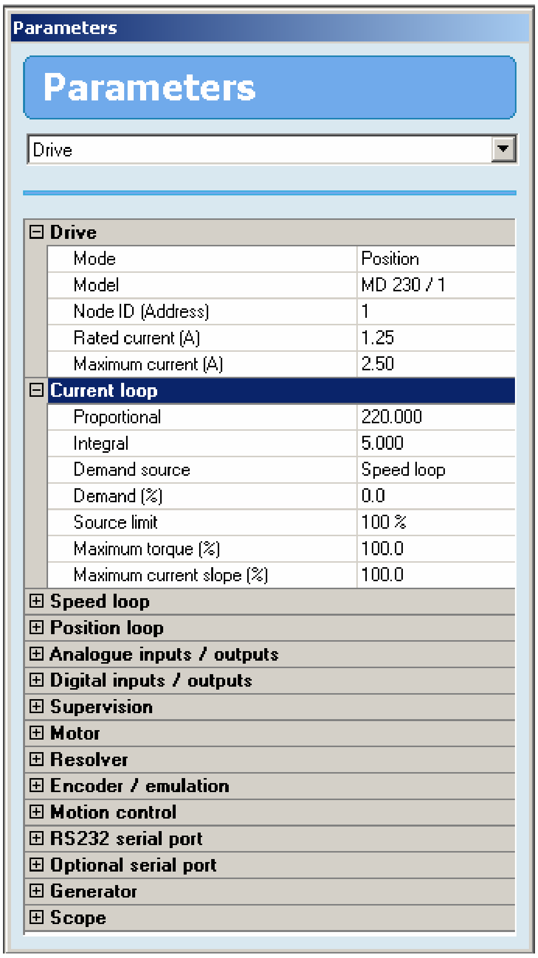
The sense of the position control loop can be inverted in the parameter list : Motion control / Invert motor sense (Caution, this does not reverse the rotor position shown on the instrument panel).

7-2- Configure an axis

7-2-1- Setup an axis

An axis must be set before using it.

The parameters access is from the **Parameter** menu or from a direct acces by the windows parameter.



Regulation

Open your parameters file from the motor librairy and tranfer it to MD drive.

Maximum following error

As soon as an axis leads in controlled mode, it is always controlled, in stop or in motion

If the difference between the calculated theoretic position and the real one given by the encoder feedback is bigger than the maximum following error, all the axis lead in non-controlled mode, and the watchdog contact is getting open (except if you use the instruction SECURITY).

The adjustment of this value is very important: a too small value stops untimely the axis control, a too big one interferes with security of electrical and mechanical devices.

Set in the field 'maximum following error' of the **Parameter \ Security \ Position** windows, the good value.

Position window

When we send an axis to a position, the MCS knows that the motion is over when the theoretic profile is achieved, and the real position is equal to the final one +/- the position window. For example, on a piercing machine for which you need an accuracy of +/- 0.1 mm, we take this value for the position window parameter.

Set in the field 'position window' of the **Parameter \ Security \ Position** windows, the required accuracy.

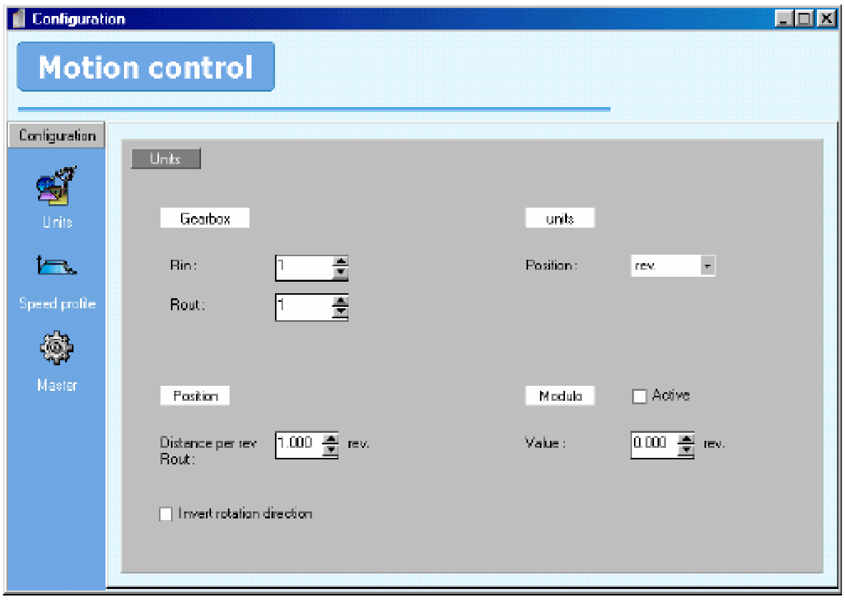
7-2-2- User Miscellaneous

Depending to the application, the mechanical (linear or rotation axis), we may affect to each axis a more easy unit: mm, pulse (encoder pulse * 4), degrees, radian, inch, round, or whatever.

Indeed, this unit is only used on the DPL screen, to be easier to understand and practice.

For example, if the selected unit is “mm”, in the “Units” menu of the DPL, speed’s unit is mm/s, and acceleration and deceleration mm/s².

Open **Motion Control \ Configuration \ Unit** and setup your axis:



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

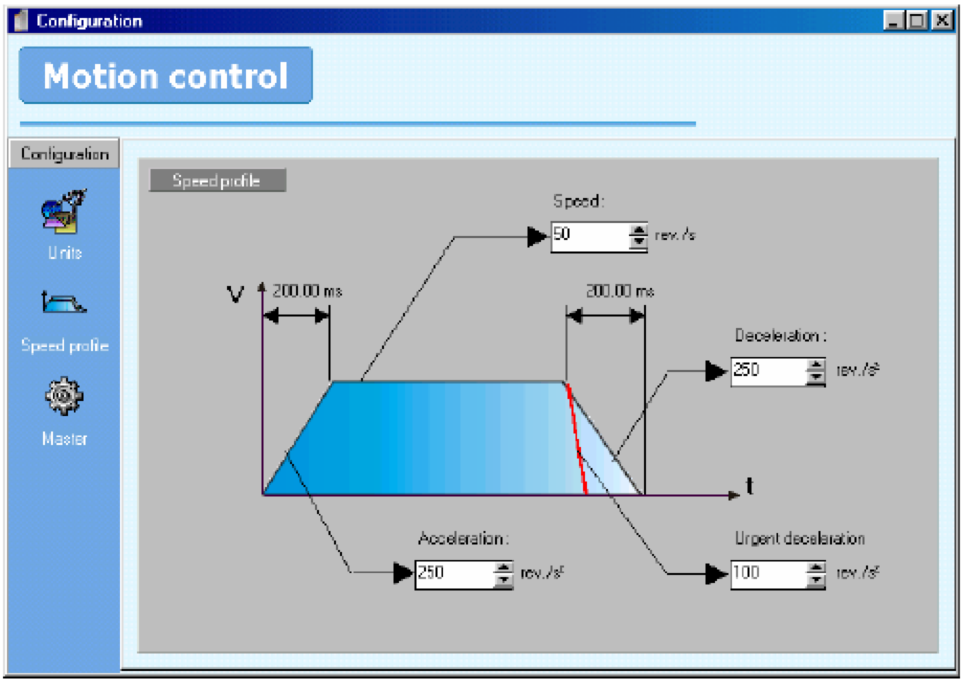
Note : the number of decimal places is a parameter in menu *Options / Language DPL*

7-2-3- Speed profile

A trajectory in positioning is made of the phases of acceleration, constant speed and deceleration.

The fields available from the board configuration in the DPLcan give default values to these different phases. The values are taken in account every time you switch on the MD. They are also used in the debug mode, and with the instructions ACC%, DEC%, VEL% and trajectories mode.

Open **Motion Control \ Configuration \ Speed profile**:



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

7-3- Open loop / Closed loop

7-3-1- 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).

↳ On detecting a fault

↳ 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
```

7-3-2- 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
```

7-4- Homing

7-4-1- Definition :

The homing allow to the system to determine the origin measures of the axis, this one being lost with each power off.

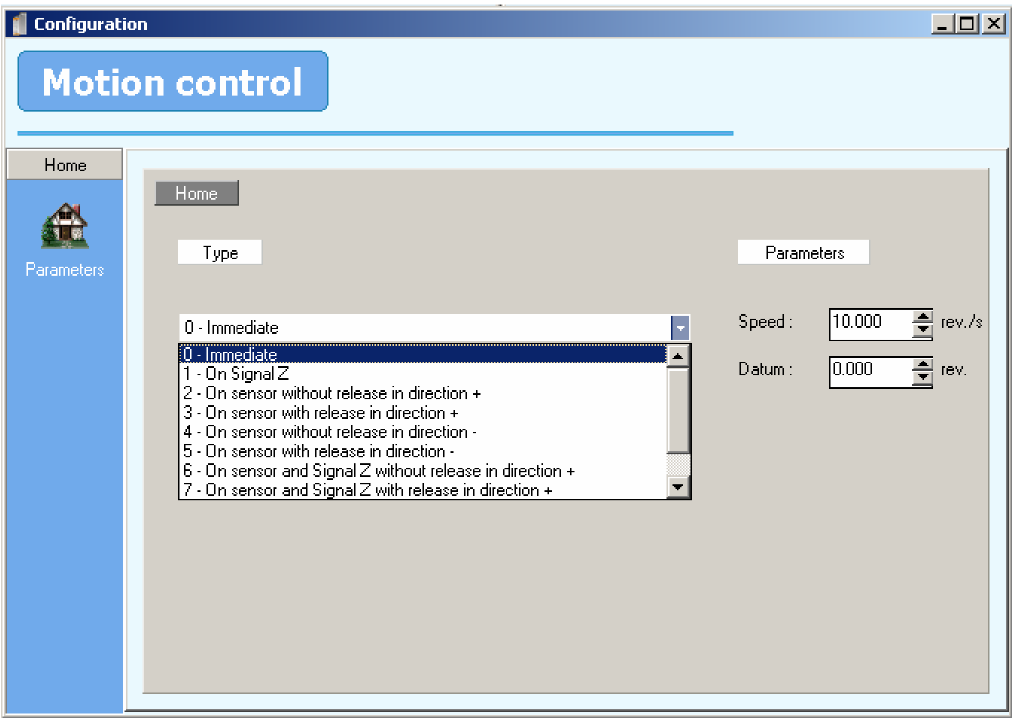
The homing (HOME) allow to refer the motor position to the mechanics position.

Various types of HOME are available: immediate, on sensor, with release.

A cycle of HOME forces the counter position to a value of reference.

7-4-2- Setup the HOME in DPL:

Homing uses the parameters set on the screen **Motion control \ Home**



From this screen, one configures the type of HOME, the speed and the datum to be charged in the counter position.

Information:

- The type chosen in this screen is used only on HOME movement declared starting from Trajectories array when the driver works in mode "trajectories pre-stored"
- If you use the HOME instruction in a BASIC task, the type must be indicated inside the instruction.

  Example: HOME on signal Z - > HOME (1)

- The speed of the axis during the HOME corresponds at the speed seized in this screen. If during the HOME, VEL or VEL% instructions are executed, the speed of the axis is then modified.
- The HOME instruction is blocking for task DPL. If you want to stop the homing during his execution, it is necessary to do in another task: HALT of the task containing the HOME instruction, then a STOP of the axis.

7-4-3- HOME types :

A) Type 0 : immediate :

The counter position is forced with the value of reference in an immediate way.

Example: Datum = 100 in the Home screen

HOME (0) ` position engine = 100

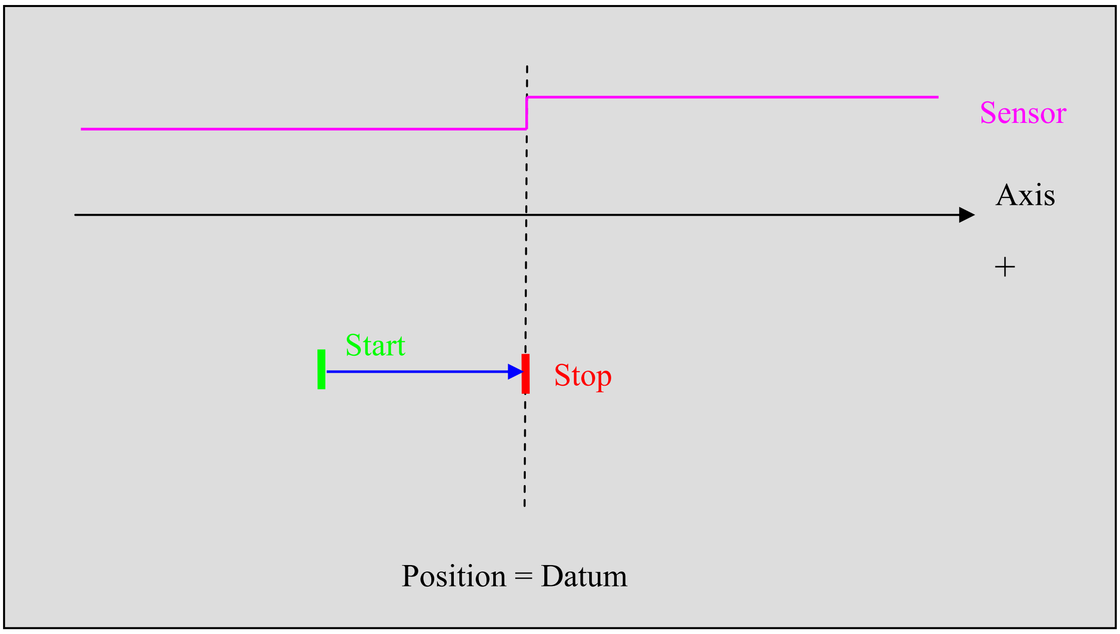
B) Type 1 : On signal Z :

The motor don't make any movement, its position is recomputed compared to driving Signal Z and the value of datum. You obtain a position being located between +/- ½ turn or datum +/- ½ driving turn.

C) Type 2: On sensor, in direction +, without release

The drive launches an infinite movement in positive direction and awaits a growing edge of the entry HOME.

The position is then forced with the value of datum and the motor stops on this position.

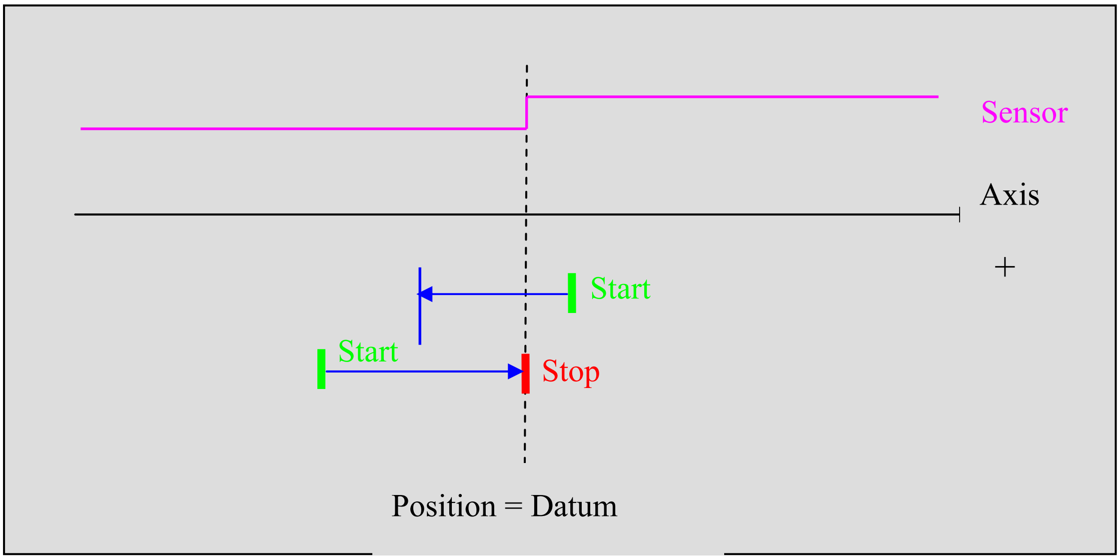


D) Type 3: On sensor, in direction +, with release

If the entry HOME is already to 1 then the drive launches in first an infinite movement in negative direction to emerge from the HOME sensor.

Then the drive launches an infinite movement in positive direction and awaits a growing edge of the entry HOME.

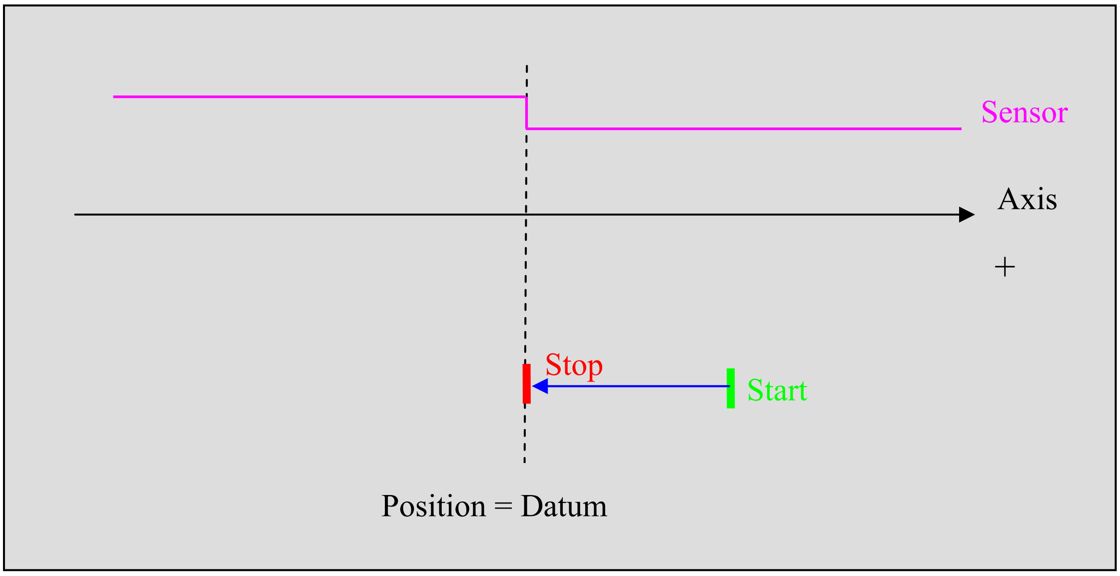
The position is then forced with the value of datum and the motor stops on this position.



E) Type 4: On sensor, in direction -, without release

The drive launches an infinite movement in negative direction and awaits a growing edge of the entry HOME.

The position is then forced with the value of datum and the motor stops on this position.

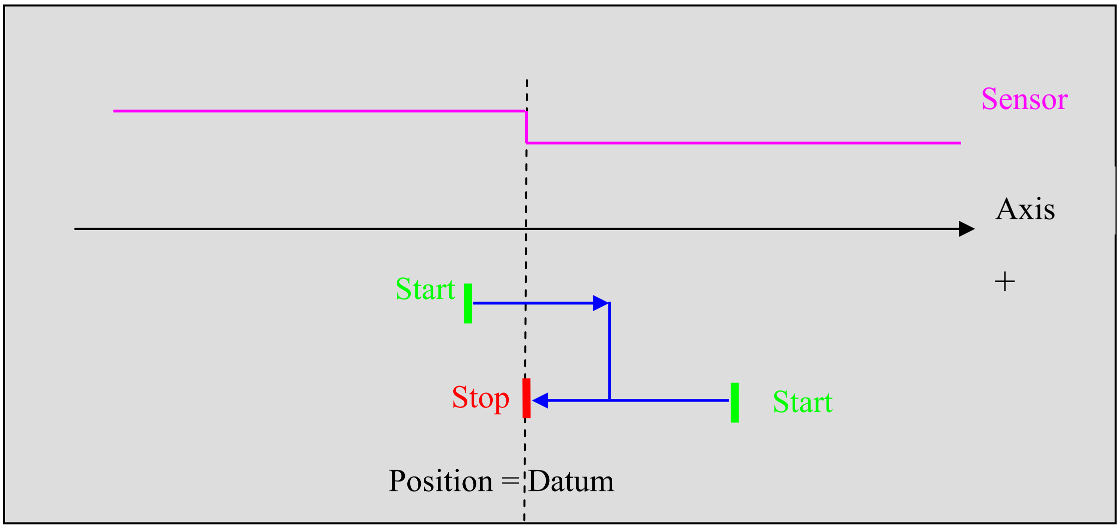


F) Type 5: On sensor, in direction -, with release

If the entry HOME is already to 1 then the drive launches in first an infinite movement in positive direction to emerge from the HOME sensor.

Then the drive launches an infinite movement in negative direction and awaits a growing edge of the entry HOME.

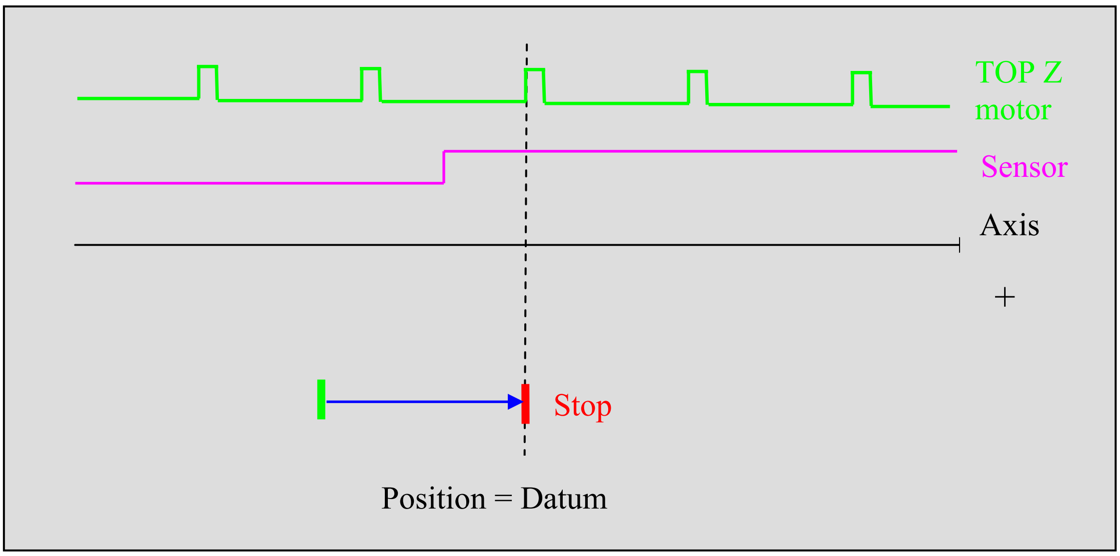
The position is then forced with the value of datum and the motor stops on this position.



G) Type 6: On sensor and signal Z, in direction +, without release

The drive launches an infinite movement in positive direction and awaits a growing edge of the entry HOME then to pass behind the Signal Z.

The position is then forced with the value of datum and the motor stops on this position.

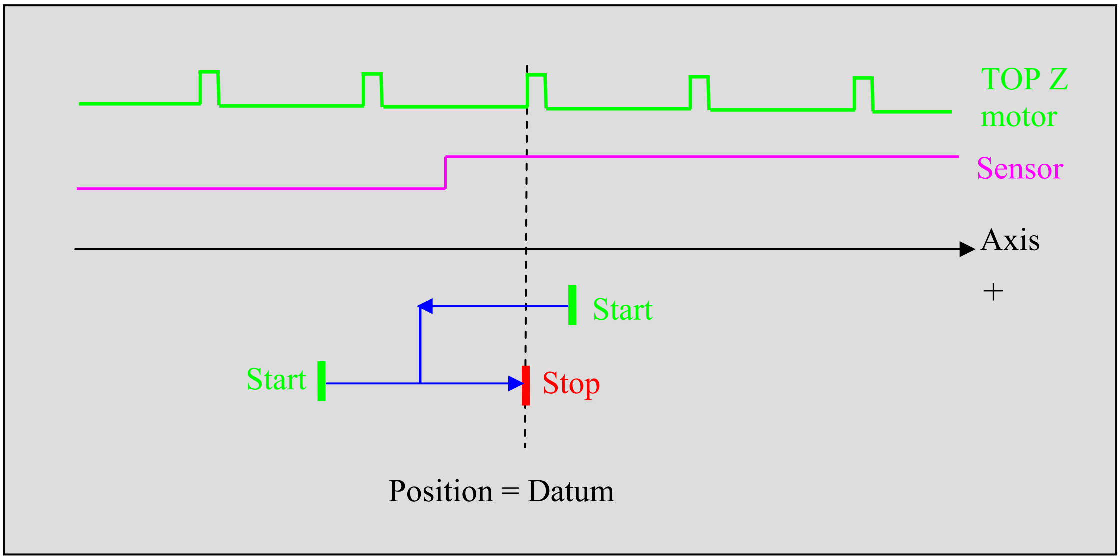


H) Type 7: On sensor and signal Z, in direction +, with release

If the entry HOME is already to 1 then the drive launches in first an infinite movement in negative direction to emerge from the HOME sensor.

Then the drive launches an infinite movement in positive direction and awaits a growing edge of the entry HOME and pass behind the signal Z.

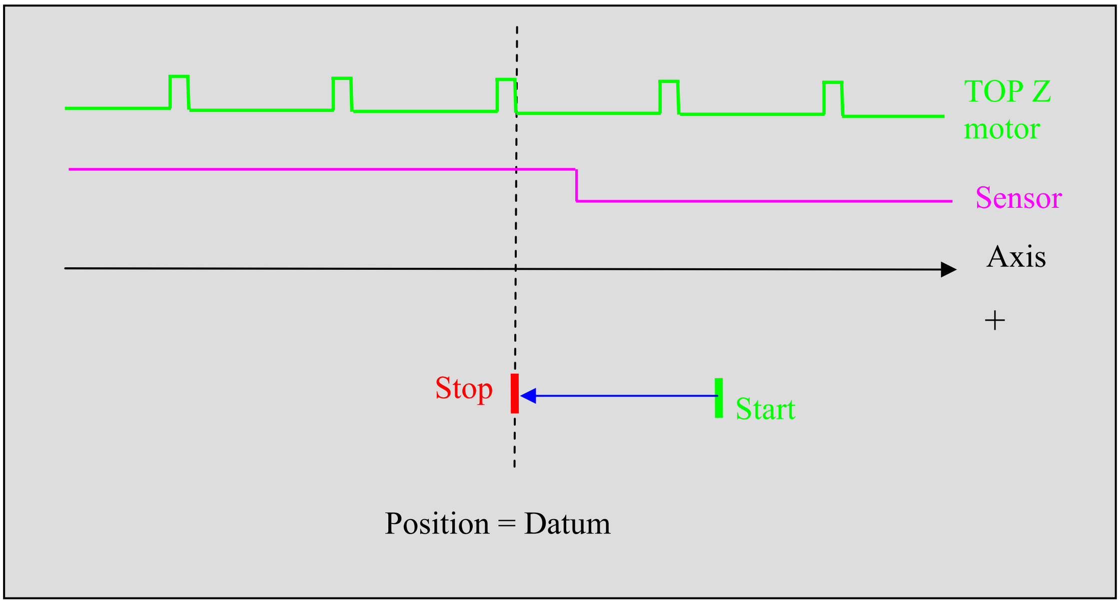
The position is then forced with the value of datum and the motor stops on this position.



I) Type 8: On sensor and signal Z, in direction -, without release

The drive launches an infinite movement in negative direction and awaits a growing edge of the entry HOME then to pass behind the Signal Z.

The position is then forced with the value of datum and the motor stops on this position.

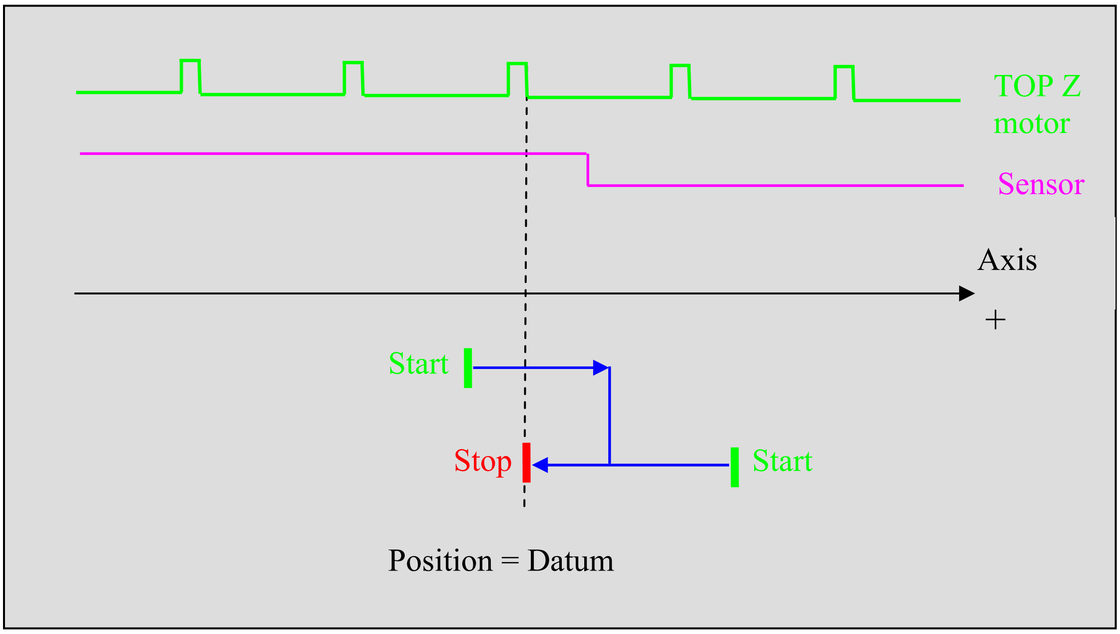


J) Type 9: On sensor and signal Z, in direction -, with release

If the entry HOME is already to 1 then the drive launches in first an infinite movement in positive direction to emerge from the HOME sensor.

Then the drive launches an infinite movement in negative direction and awaits a growing edge of the entry HOME and pass behind the signal Z.

The position is then forced with the value of datum and the motor stops on this position.



7-5- Declaration of an axis in virtual mode

From a basic task, it is possible to lead an axis in virtual mode with the instruction LOOP On. In this mode, the MD drive will simulate the encoder pulses in an intern way, so every command send will be made virtually.

This mode is interesting during the program development phase: we can test the global application without motors and drives connected.

In this mode, don't connect power connector X10.

The LOOP Off instruction cancels the virtual mode.

7-6- Positioning

7-6-1- Absolute movements

A) 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
```

B) Move : 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

C) 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

7-6-2- Relative movements

A) 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.

B) Move : MOVR

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

MOVR=Distance

This instruction sends the axis to a 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

C) Trajectory : TRAJR

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

It allows a movement to be launched towards a 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

7-6-3- 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
```

7-6-4- 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.

7-6-5- Stopping a movement

It is possible to execute movement by communication bus by writing drive parameter (Open Help \ Modbus-CANopen window).

A) Speed profile:

- _MOTION_PROJECT_VEL allows to specify the current speed in units per second.
- _MOTION_PROJECT_ACC allows to modify the current acceleration value.
- _MOTION_PROJECT_DEC allows to modify the current deceleration value.
- _MOTION_PROJECT_VELACCDEC allows to specify speed profil in percent of Motion Control \ Speed profil windows parameters.

B) Positioning :

- _MOTION_PROJECT_HOME allows to execute a HOME (parameter value give the home type)
- _MOTION_PROJECT_STTA allows to start an absolute movement to parameter value.
- _MOTION_PROJECT_STTR allows to start a relative movement to parameter value.
- _MOTION_PROJECT_SSTOP allows to stop movement.

7-7- Synchronization

7-7-1- Electronic gearbox

A) GEARBOX :

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

Syntax :

GEARBOX(<Numerator>, <Denominator>,<Reverse>)

<Numerator> / < Denominator > defines the ratio between one rev of the slave and one motor rev of the encoder, i.e. for <Denominator > increments of the master, the motor will make a move of <Numerator > increments.

<Reverse> is a logical variable that indicates if gearbox is reversible.

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 will return a value of 1 (even if the slave axis is stopped).

 The instruction GEARBOX internally sets the value of GEARBOXRATIO to 1.

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

The Numerator is a real.

The Denominator is a real.

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

B) 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>, with <Numerator> and <Denominator> defined in the instruction GEARBOX.

Syntax : STARTGEARBOX (<Master acceleration dist.>)

< Master acceleration dist.> is real.

With Ratio that corresponding to the value of GEARBOXRATIO.

C) GEARBOXRATIO :

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

Syntax : GEARBOXRATIO(<Ratio>,<Master acc. Distance>)

<Ratio> is real :
The ratio of the gearbox is defined by <Ratio> × <Numerator> / <Denominator>.
<Numerator> and <Denominator> are parameters of the instruction GEARBOX..

<Master acc. Distance>) is the distance where the master will accelerate.

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 encoder.

The instruction GEARBOX internally sets the value of GEARBOXRATIO to 1.

D) STOP :

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

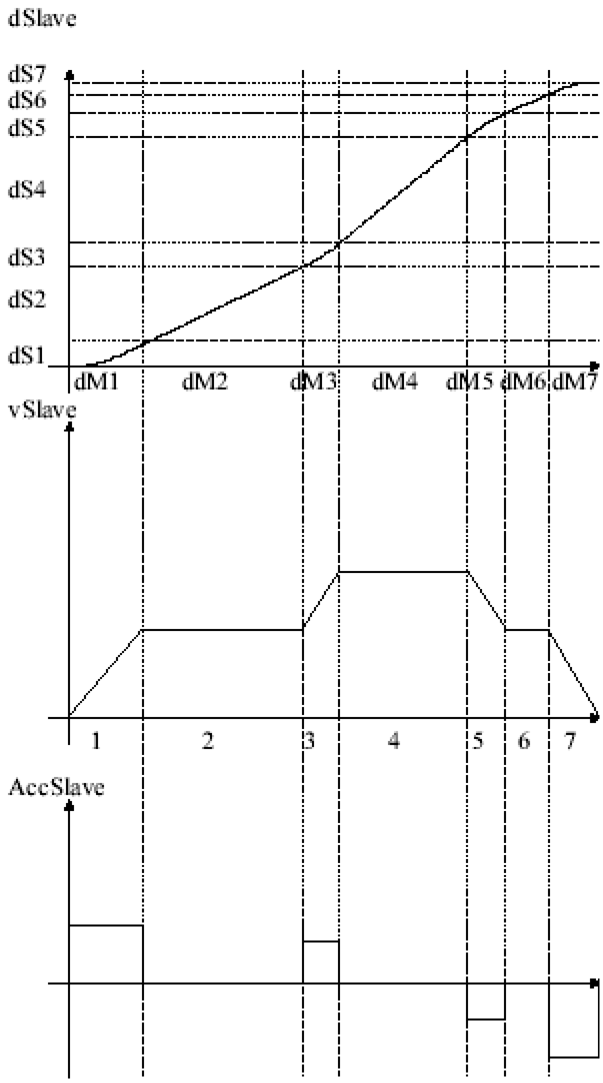
Syntax : STOP

E) Example :

GEARBOX (1, 2,0)	'The motor turns twice as fast as the master encoder
GEARBOXRATIO(1)	
…	
STARTGEARBOX(10)	'Initiate a gearbox with an acceleration phase
…	'of 10 units
GEARBOXRATIO(2)	'Final ratio : 2 * ½ = 1
STOP	'Stop the gearbox with a deceleration phase
WAIT MOVE_S=0	'of 10 units

7-7-2- Synchronised movements

A) General formula :



dSa :	Slave distance during acceleration phase
dMa :	Master distance during acceleration phase
dSd :	Slave distance during deceleration phase
dMd :	Master distance during deceleration phase
dSn :	Slave distance during the next constant phase
dMn :	Master distance during the next constant phase
dSp :	Slave distance during the previous constant phase
dMp :	Master distance during the previous constant phase

For an acceleration from a zero velocity :
dSa/dMa =1/2*dSn/dMn

For an deceleration to a zero velocity :dSd/dMd =1/2*dSp/dMp

For an acceleration or a deceleration phase between two constant phases :

dSa/dMa =1/2*(dSp/dMp+dSn/dMn)

dSd/dMd =1/2*(dSp/dMp+dSn/dMn)

For this example we have :

dS1/dM1 =1/2*dS2/dM2
dS3/dM3 =1/2*(dS2/dM2+dS4/dM4)
dS5/dM5 =1/2*(dS4/dM4+dS6/dM6)
dS7/dM7 =1/2*dS6/dM6

B) Move : MOVS

The instruction MOVS provides a synchronisation between a slave and a master axis.

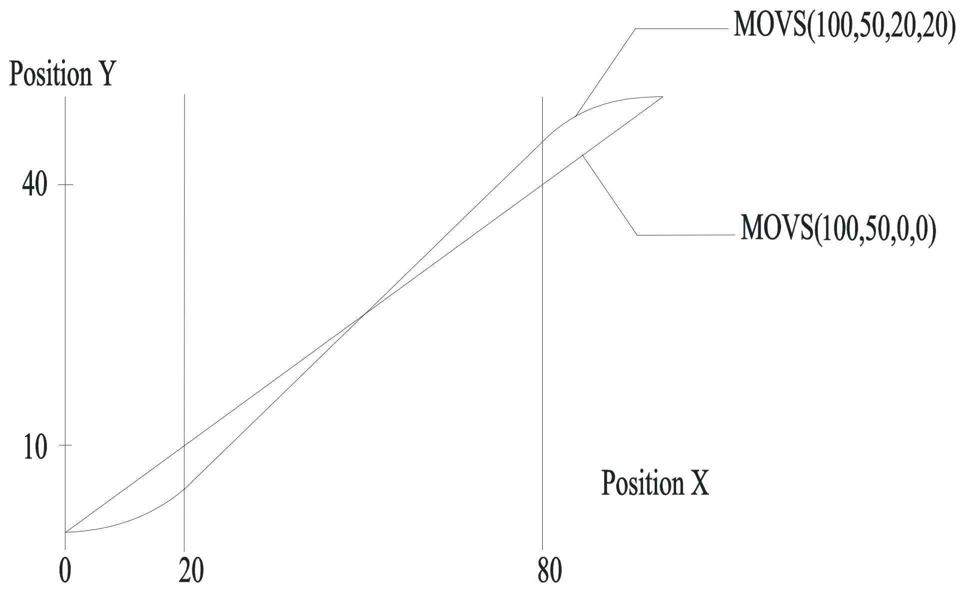
This instruction does not stop the task (except if the movements buffer is full).

Syntax : MOVS(<MasterDist>, <SlaveDist>, <AccelDist>,<DecelDis>)

Example :MOVS(20, 10, 0, 0) ‘for a relative movement of 20 units ‘on master, slave moves of 10

It is used for synchronising the slave and master axis for a precise distance of the master axis, with separately variable phases of acceleration and deceleration on the slave axis. The master axis can be a servo axis or an axis encoder. The slave axis must be a servo axis.

For example :



This example shows 2 synchronised movements with and without the acceleration and deceleration phases. When there is no acceleration and deceleration phase, the master axis and the slave axis must have the same speed to limit the transitory phases. If the speeds are very different, acceleration and deceleration must be adjusted to avoid mechanical problems.

The speeds are not necessarily the same and depend on the acceleration and deceleration phases, because the system has to respect distances.

C) Stop : STOPS

When the master axis arrives at <MasterPos.>, slave axis starts deceleration until <SlavePos.>.

Syntax : STOPS (<MasterPos.>, <SlavePos.>)

<MasterPos.> is a real in the master unit.

<SlavePos.> is a real in the slave unit.

Example : STOPS (20, 105) ‘When the master arrives at position 20,

‘ the slave axis will decelerate until position 105 on

‘ slave axis

Warnings: The call of STOPS instruction reset STOPS_S flag.

D) Status : STOPS_S

This instruction can be use only if STOPS instruction has been call before. This flag indicates if the slave position given by the STOPS has been achieved. This flag is reset after having been read.

Return 1 if :

- If it is not possible to achieve the demanded slave position (e.g.: demanded slave position has already been passed.)
- If slave speed is null (during a constant phase).

Else return 0

Syntax : VF0 = STOPS_S

Example :

```
MOVS (20, 10, 0, 0)
...
STOPS (20, 105)
WAIT MOVE_S=0
IF STOPS_S=1 GOTO ERRSTOPS
```

E) Applications :

MOVS instruction accepts the following combinations :

- Velocity changing phase
- Velocity changing phase + Constant phase
- Constant phase
- Constant phase + Stop phase
- Stop phase
- Velocity changing phase + Constant phase + Stop phase

a) Velocity changing phase

(i) Zero initial velocity :

In the previous example, the phase 1 is a velocity changing phase with zero initial velocity.

MOVS(Slave,Master,dM1,dS1,dM1,0)

The velocity ratio at the end of this phase is equal to 2*dS1/dM1

(ii) Initial velocity greater than zero and lower than final velocity :

Phase 3 represents this kind of velocity changing phase.

The initial velocity ratio is dS2/dM2 and the final velocity ratio is dS4/dM4 so :

dS3 = dM3 * (dS2/dM2+dS4/dM4) /2

MOVS(Slave,Master,dM3,dE3,dM3,0)

The average velocity ratio during this phase is dS3/dM3 and is greater than the initial velocity ratio; so this is an acceleration phase.

(iii) Initial velocity greater than zero and greater than final velocity :

This kind of phase is represented on phase 5.

The initial velocity ratio is dS4/dM4 and the final velocity ratio is dS6/dM6 so :

dS5 = dM5 * (dS4/dM4+dS6/dM6) / 2

MOVS(Slave,Master,dM5,dS5,dM5,0)

The average velocity ratio during this phase is dS5/dM5 and is lower than the initial velocity ratio; so this is a deceleration phase.

b) Velocity changing phase + Constant phase

(i) Zero initial velocity :

In the previous example, the phase 1 is a velocity changing phase with zero initial velocity.

dS10=dS1+dS2=1/2*dM1*dS2/dM2+dS2

dM10=dM1+dM2

MOVS(Slave,Master,dM10,dS10,dM1,0)

(ii) Initial velocity greater than zero and lower than final velocity :

The phases 3 represent this kind of velocity changing phase.

The initial velocity ratio is dS2/dM2 and the final velocity ratio is dS4/dM4 so :

dS30=dS3+dS4=dM3*(dS2/dM2+dS4/dM4)+dS4 / 2

dM30=dM3+dM4

MOVS(Slave,Master,dM30,dS30,dM3,0)

The average velocity ratio during this phase is dS3/dM3 and is greater than the initial velocity ratio; so this is an acceleration phase.

(iii) Initial velocity greater than zero and greater than final velocity :

This kind of phase is represented on phase 5.

The initial velocity ratio is dS4/dM4 and the final velocity ratio is dS6/dM6 so :

dS50=dS5+dS6=dM5*(dS4/dM4+dS6/dM6)/2+dS6

dM50=dM5+dM6

MOVS(Slave,Master,dM50,dS50,dM5,0)

The average velocity ratio during this phase is dS5/dM5 and is lower than the initial velocity ratio; so this is a deceleration phase.

c) Constant phase

Phases 2,4 and 6 are constant.

MOVS(Slave,Master,dM2,dS2,0,0)

MOVS(Slave,Master,dM4,dS4,0,0)

MOVS(Slave,Master,dM6,dS6,0,0)

d) Constant phase + Stop phase

With phase 6 and 7 we have :

dE70=dE6+dE7=dE6+1/2*dM7*dE6/dM6

dM70=dM6+dM7

MOVS(Esclave,MaÓtre,dM70,dE70,0,dM7)

e) Stop phase

Phase 7 is a stop phase

MOVS(Slave,Master,dM7,dS7,0,dM7)

The velocity ratio before this phase was 2*dS7/dM7

f) Velocity changing phase + Constant phase + Stop phase

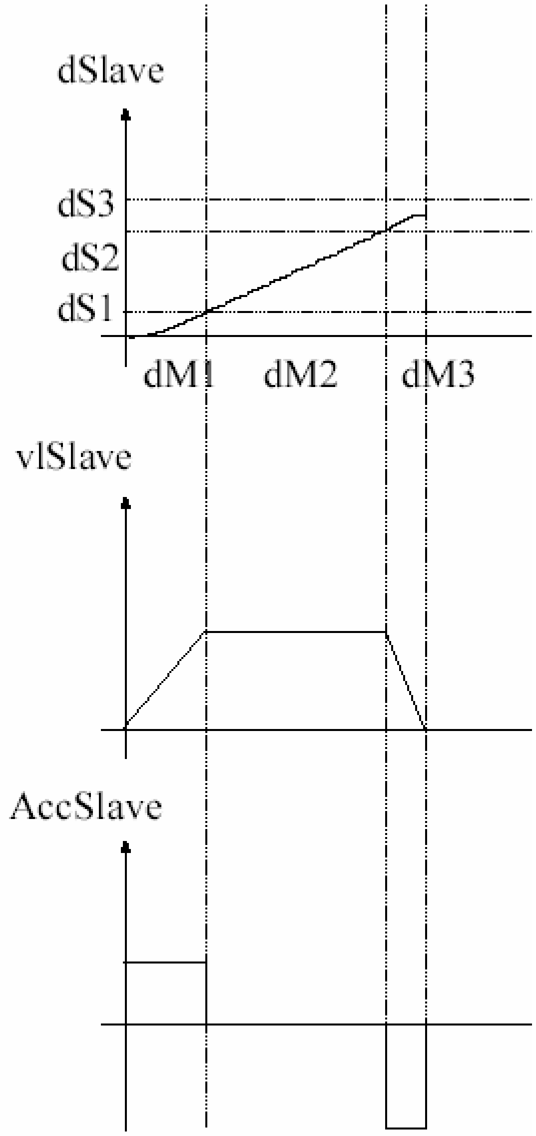
dS1=1/2*dM1*dS2/dM2

dS3=1/2*dM3*dS2/dM2

dS=dS1+dS2+dS3=1/2*(dM1+dM3)*dS2/dM2

dM=dM1+dM2+dM3

MOVS(Slave,Master,dM,dS,dM1,dM3)



7-7-3- Compensation functions

A) ICORRECTION – Correction function

Syntax : ICORRECTION(<Dist.master>,<Dist.slave>,<Dist. accel>)

Units : <Dist.master>, <Dist.slave> : user unit (Ex : mm, degree,…)

<Dist.accel> : user unit /s²

Accepted types :<Dist. master>, <Dist. slave>, <Dist. accel> : real

Description : This function applies a correction movement to the slave axis during the distance of master axis.

Remarks : The slave axis must be linked to the master axis by a gear box function (GEARBOX), a synchronized movement (MOVS) before the execution of the correction instruction. With the synchronized movement of the slave axis, the next movement is superposed: During the distance of the master axis, a movement <Dist. slave> is added with an acceleration and a deceleration on a <Dist. accel>.

Attention : All other corrections are ignored if a correction is being done or if <Dist. master> is null.

B) ICORRECTION_S – Correction status

Syntax : <Variable> = CORRECTION_S

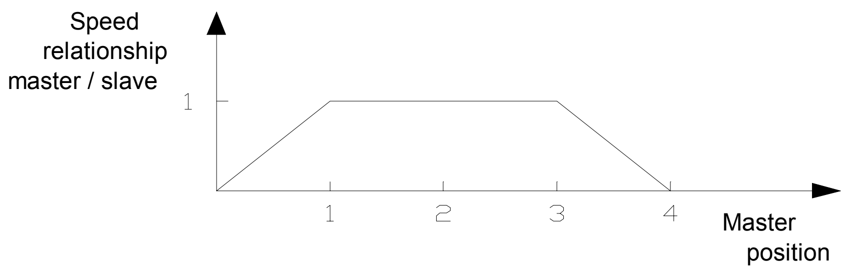
Accepted types:<Variable> : bit

Description : This function is used to ascertain the status of a correction : returns 1 if a correction is taking place else returns 0.

C) EXAMPLE

Synchronised movement:

MOVS (4, 4, 1, 1)



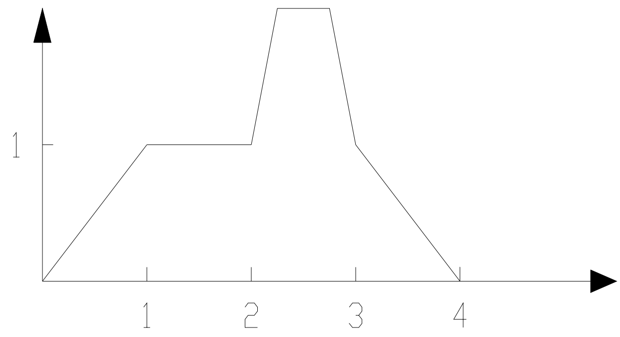
Synchronised movement

Synchronised movement + correction :

MOVS (4, 4, 1, 1)

WAIT (POSMASTER_S > 2)

ICORRECTION (1, 1, 0.2)



Synchronised movement + correction

7-7-4- Cam

A) Graphical editor :

Use the cam editor in Motion Control \ Cam editor to build and edit your cam profiles.

Cam must be declared in windows **Project \ Setup \ DriveName \ Cams**.

The cam function allows the realisation of a cam profile on a slave axis linked to a master axis. This profile is defined with an array of points. An IMD drive can store up to 5 cams and 512 points for the 5 cams.

Each point is defined as a master position and a slave position.

The values given to the master positions inside the array must be increasing.

		Begin		End	
#	Mode	Master	Slave	Master	Slave
0	Manual	0.000	100.000	10.001	100.000
1	Line	10.000	90.000	10.000	1.000
2	Manual	90.000	10.000	99.667	1.136
3	Line	100.000	0.000	100.000	1.000
4	Manual	120.000	0.000	129.000	0.000
5	Line	130.000	5.000	130.000	1.000
6	Manual	152.000	5.000	158.001	5.000
7	Line	158.000	0.000	158.000	1.000
8	Manual	180.000	0.000	188.250	0.000
9	Line	196.500	10.000	196.500	1.000
10	Manual	328.500	90.000	342.750	98.864

A cam point is defined by:

↳a mode

↳a master position

↳a slave position

↳a master tangential

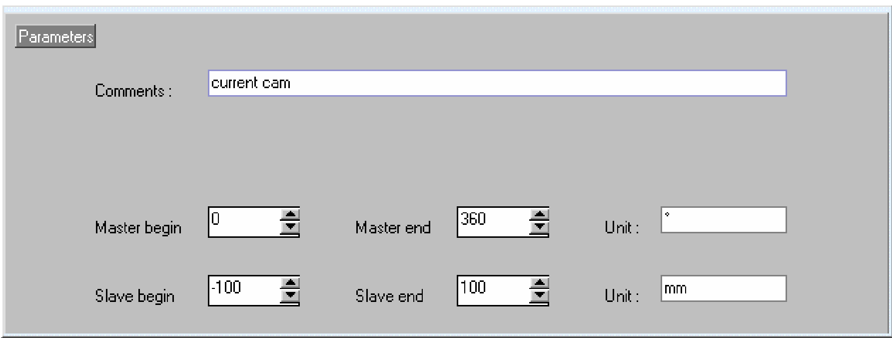
↳a slave tangential

The cam form depends on each point's mode :

↳ Line : calculate a line from the current point to the next point (there is a speed discontinuity at the current speed, speed keeps the same until the next speed).

↳ Auto : calculate a trajectory with a 3rd order polynomial (use the current point, the next point and the previous point).

↳ Manual : calculate a graph according to a tangential at the current point and with slope = master tangential / slave tangential.



In the parameter area of the cam editor, you can set up :

↳ Scale : Begin and end of master (X), begin and end of slave (Y). Units are just used for display.

All cam tables are saved in FRAM memory. To write or read a cam point, use this instruction:

<VRx>=ReadCam(<Index>, <Sub index>)

WriteCam(<Index>, < Sub index>)=<VRx>

<Index> from 0 to 511, cam point number in FRAM

< Sub index> from 0 to 3, cam point parameter:

↳0 for master position

↳1 for slave position

↳2 for master tangential

↳3 for slave tangential

Trajectory mode depends on different parameter values :

↳ If master position <> master tangential then the trajectory is type Manual

↳ If master position = master tangential and slave tangential <> 0 then the trajectory is type Line.

↳ If master position = master tangential and slave tangential = 0 then the trajectory is type Auto.

B) Absolute and relative cams :

The difference between an absolute and a relative cam is the datum, for a relative cam, the datum is the real axis position and for an absolute cam the datum is 0.

Example :

CAM profile	
Master	Slave
0	5
10	7
20	30
30	35
40	30
50	15

If master position is 20 and slave is 30 before starting the cam, the cam will do those movements for an absolute cam:

Absolute CAM	
Master pos.	Slave pos.
20	30
30	35
40	30
50	15

If master position is 20 and slave is 30 before starting the cam, the cam will do those movements for a relative cam:

Relative CAM	
Master pos.	Slave pos.
20	35
30	37
40	60
50	65
60	60
70	45

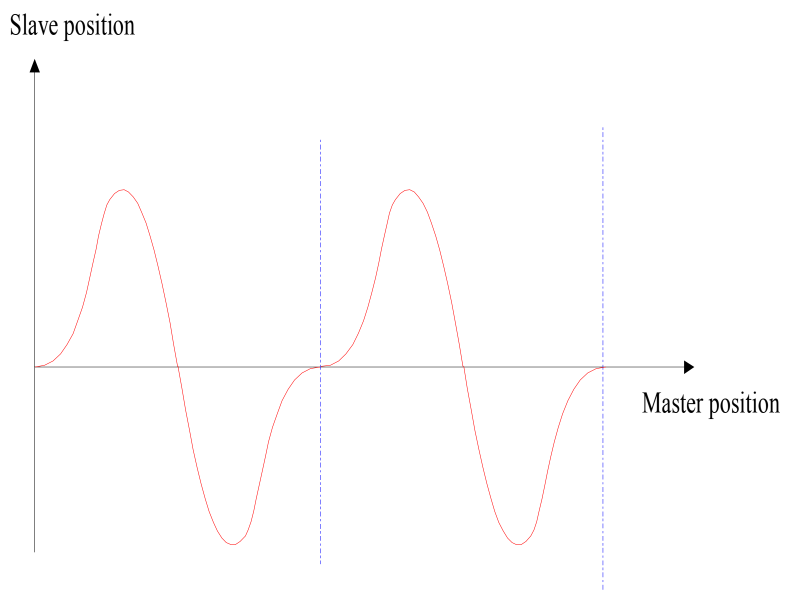
C) Finite and infinite cams :

A mechanical cam corresponds to a finite electronic cam. In the points array, the first and last values of the slave position are the same. The slave movement will be a linear movement with a finite magnitude.

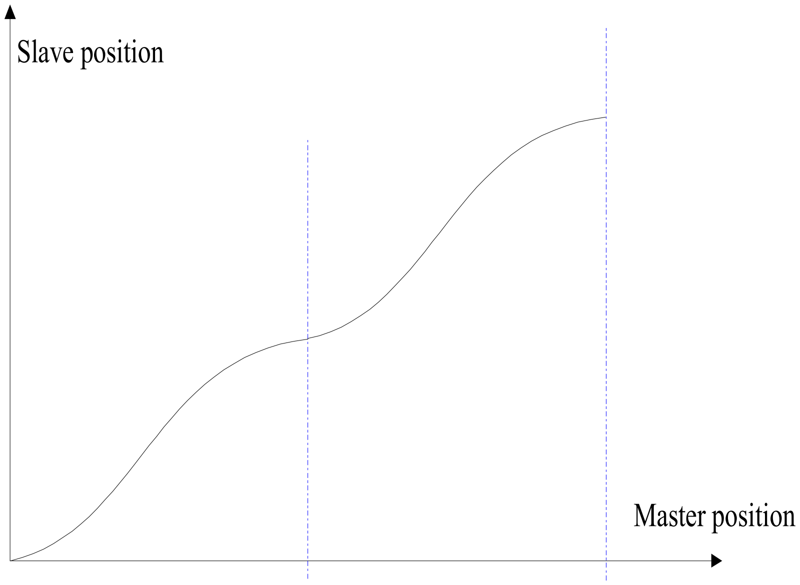
The electronic cam also permits the creation of an infinite slave rotation movement: the absolute slave position increases for each new master cycle.

Warning: If the master axis or the slave axis is infinite, they must be declared as modulo axes from the Motion control tab of the iDPL software.

Finite cam



Infinite cam



D) Loading a cam :

Syntax: LOADCAM (<NumberCam>, <Absolute>, <Table>, <Number>, <SingleShot>, <Reversible>, <Direction>, <MasterGain>, SlaveGain>, <NumberNextCam>, <NumberPreviousCam>)

Description: this instruction loads a cam in the drive.

Limits : <NumberCam>: 1 to 5

<Absolute> : 1 for absolute cam else 0

< Table >: First element of the table to define the cam (0 to 511)

<Number>: Number of elements of the table to define the cam (2 to 512)

<SingleShot>: Define the automatic re-looping of the cam:

↳ 0: Re-looping cam, it will be stopped only when the stop instruction will be executed.

↳ 1: Single-shot cam

<Reversible>: Tell if the <Slave> must follow the master in both directions.

↳ Input 0 for a non-reversible cam: if the master moves in the opposite way as the one defined in <Direction>, the slave stops. It will start off again when the master will go in the right way and pass by the position where the slave stopped.

↳ Input 1 for a reversible cam: The slave follows its cam profile whatever is the master direction.

<Direction>:Input 0 for no direction, 1 for a negative direction, 2 for a positive one.

<MasterGain>: Applied coefficient to cam master position (default value 1).

<SlaveGain>: Applied coefficient to cam slave position (default value 1).

<NumberNextCam>: Input 0 if the cam must not be followed by another one. If it is not the case, input the number of the next cam, from 1 to 5.

<NumberPreviousCam>: Input 0 if the cam will not start at the end of another one. If it is not the case, input the number of the previous cam (from 1 to 5).

E) Launching a cam:

To launch the execution of a cam, use the instruction STARTCAM.

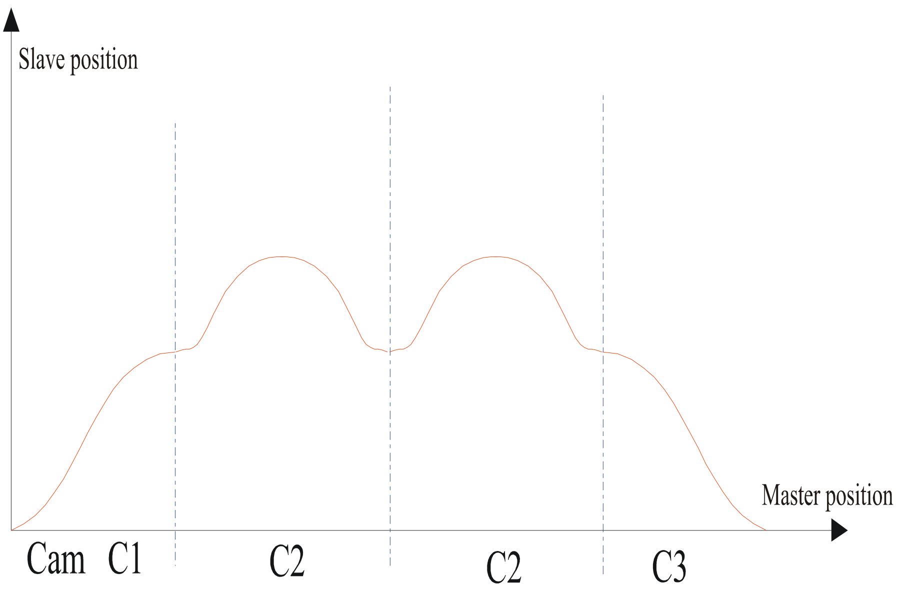
Its syntax is : STARTCAM(<NumberCam>)

<NumberCam> : number of the cam (from 1 to 5).

F) Chaining cams:

Here is a cycle made of three cams: C1 with an input profile single-shot, C2 repetitive, and C3 with an output profile single-shot.

C1 is chained with C2 and C2 to C3.



PROG

.....

‘ Loading cam n °1 : 10 points, single-shot, followed by cam C2

LOADCAM(1,0,0,10,1,1,0,1,1,2,0)

‘ Loading cam n °2 : 36 points, non single-shot, followed by cam C3

```
LOADCAM(2,0,10,76,0,1,0,1,1,3,1)
‘ Loading cam n °3  : 6 points, single-shot
LOADCAM(3,0,86,6,1,1,0,1,1,0,0)
‘ Launching of cam C1 => execution of C1, then C2
STARTCAM(1)
WAIT CAMNUM_S=2             ‘ Wait execution of C2
.....
WAIT INP(StopInfo)          ‘ Wait for stop requirement
ENDCAM(Slave)               ‘ Stop cam 2 at the end of profile
                            ‘ and then cam 3
WAIT NOT CAM_S(Slave)       ‘ Wait for end of cam 3
.....
END PROG
```

G) State of the cam:

Three functions can show the current state of a servo board running a cam.

➬ Instruction MOVE_S : permits to know if a cam is running

Example :

```
IF NOT MOVE _S THEN GOTO FINCAME          ‘Stopped cam
IF MOVE _S THEN GOTO CAME_EN_COURS    ‘Running cam
```

➬ Instruction CAMNUM_S : returns the number of the running cam. The returned value is valid only if MOVE_S is set.

Example :

```
IF CAMNUM_S=1 THEN GOTO ATTENTE_FIN_CAME_1 ‘Cam 1 running
IF CAMNUM_S=2 THEN GOTO ATTENTE_FIN_CAME_2 ‘Cam 2 running
```

➬ Instruction CAMSEG_S : returns the equation number of the cam that is running. The returned value is valid only if MOVE_S is set.

Example :

```
IF CAMSEG_S=1 THEN GOTO ATTENTE_FIN_SEGMENT_1 ‘Cam between point 1 and point 2
```

IF CAMSEG_S=2 THEN GOTO ATTENTE_FIN_SEGMENT_2 ‘Came between point 2 and point 3

H) Stop a cam:

The function ENDCAM stops the slave movement at the end of the cycle, while the function STOP stops it immediately. The syntax of the instruction ENDCAM is : ENDCAM.

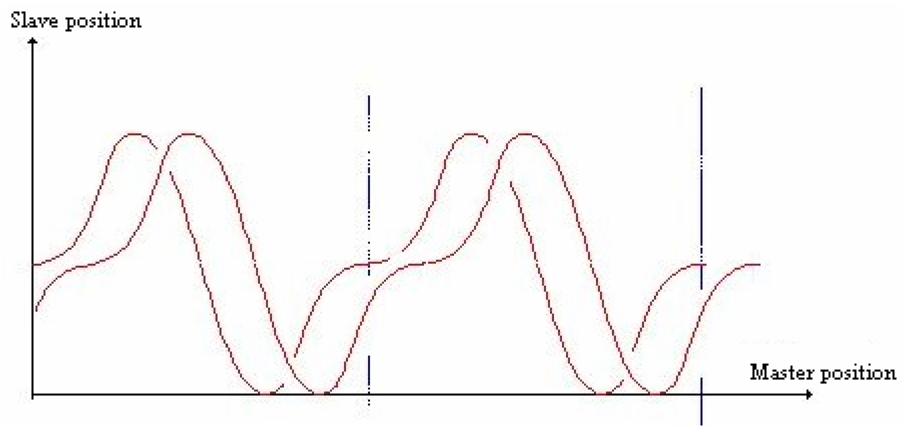
Warning:

If ENDCAM is applied to a cam that has been declared in non-single shot and linked with another one, the cam ends its profile and goes on to the next.

I) Dynamic de-phasing:

a) Master de-phasing

The master de-phasing effects to de-phase the master cycle with regard to the slave. In the case of a re-looped cam, it is necessary to respect this de-phasing for positioning the slave with regard to the master. The master de-phasing can be done progressively by the use of an acceleration parameter. The de-phasing is applied directly if the synchronised movement is not running or if axis in not enabled.



MasterOffset(OffsetMaster,1000)

SlaveOffset(OffsetSlave,1000)

StartCam(1)

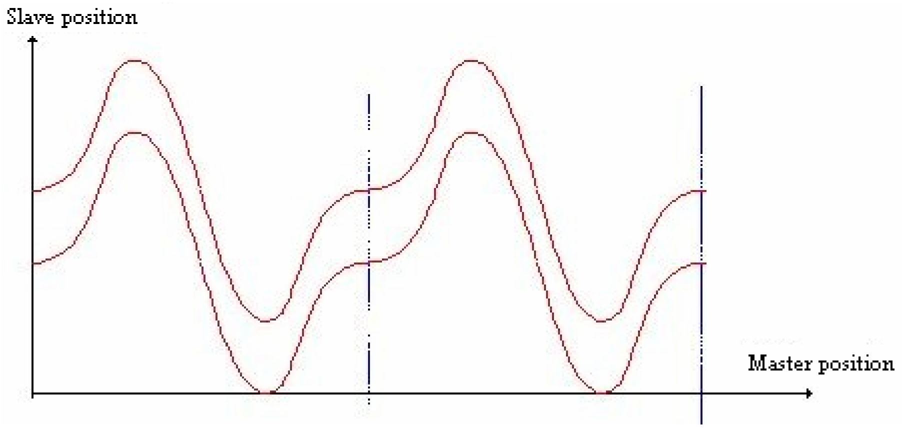
…

…

OffsetMaster= OffsetMaster+10

MasterOffset (OffsetMaster, 0.1)

b) Slave de-phasing



The slave de-phasing effects to de-phase the slave position but keeps the phase with the master cycle. It is necessary in all case to allow for this de-phasing to position the slave according to the master. The slave de-phasing can be done progressively by the use of an acceleration parameter. The de-phasing is applied directly if the synchronised movement is not running or if axis is not enabled.

MasterOffset(OffsetMaster,1000)

SlaveOffset(OffsetSlave,1000)

StartCam(1)

…

…

OffsetSlave= OffsetSlave+10

SlaveOffset (OffsetSlave, 0.1)

J) Modification of a cam point : LOADCAMPOINT

Modify a cam point in FRAM memory.

Syntax : LOADCAMPOINT (<NumCam>, <NumPoint>, <FRAMIndex>)

< NumCam > : Number of the cam loaded previously (from 1 to 5).

< NumPoint > : Number of the cam point to modify (from 1 to NB cam point).

< FRAMIndex > : Address of the point in FRAM(from 0 to 511) to send in the target cam point.

Warning: This instruction blocks the task (LOADCAMPOINT can only be done if the cam is not between previous and next < NumPoint > point). This instruction gives an iDPL error if no cam has been loaded before.

K) Slave position in the cam: CAMREADPOINT

This instruction calculates the slave position <Slave position> in the cam, corresponding to the master position <master position>.

Syntax : <Slave position>=CAMREADPOINT(<Master position>,<NumCam>)

Accepted types :< Master position>: real

<Slave position>: real

<NumCam> Number of the selected cam loaded previously(1 to 5)

Remarks : Return 0 if <Master position> is not in the selected cam.

L) Execution of a triggered cam :

It is possible to make a cam movement launched by the basic instruction TRIGGER.

M) Warning :

↳ Value for master position in the cam table is monotonic.

↳ This difference between 2 points must not be too small (minimum time between 2 points is 300µs).

7-7-5- Multi-axis using CANopen

It is possible to synchronize several drives by position exchange on the CANopen bus:

A) Source drive task :

```
Prog
    StartCANSendPosition(1,1,210h,10)
Bcl:
    Goto Bcl
EndProg
```

B) Slave drive task :

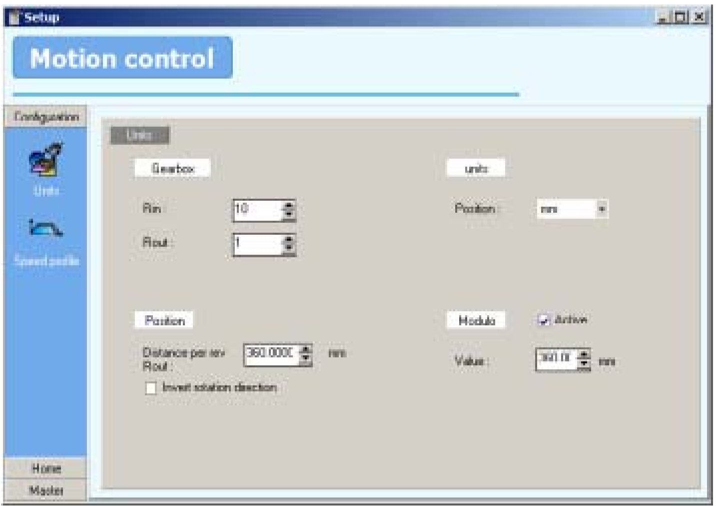
```
Prog
        StartCANReceivePosition(1,210h,0,20)
        Axis On
        Wait(Axis_S) = On
        Filtermaster 1
        Gearbox(1,1,1)
        Startgearbox(1)
Blc:
        vi0=canposstatus
        If (vi0=2) then
                vi1=vi1+1
                canpostimeoutraz
        Endif
        Goto test
EndProg
```

C) Warning :

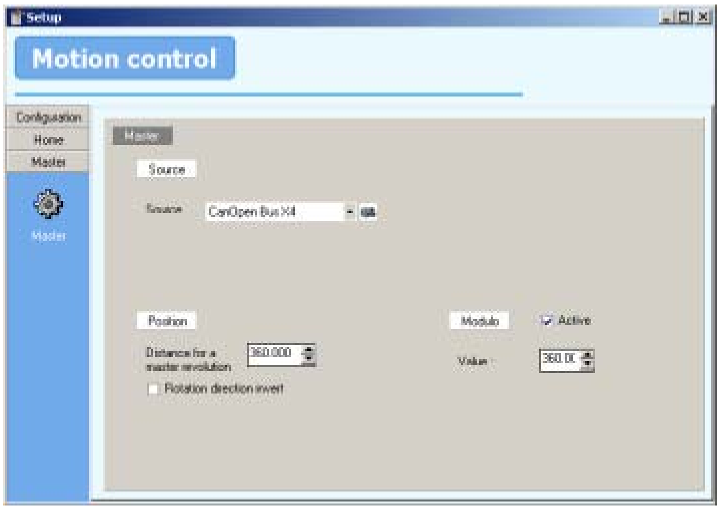
In Motion control \ Master slave, the master source must be configured as CANopen X4

If master use modulo, it is obligatory to same units between Motion Control \ Units windows(master drive) and Motion Control \ Master window (slave drive) :

Master



Slave



Master and slave modulos must be equal and distance per master rev = distance per Rout rev * Rout / Rin.

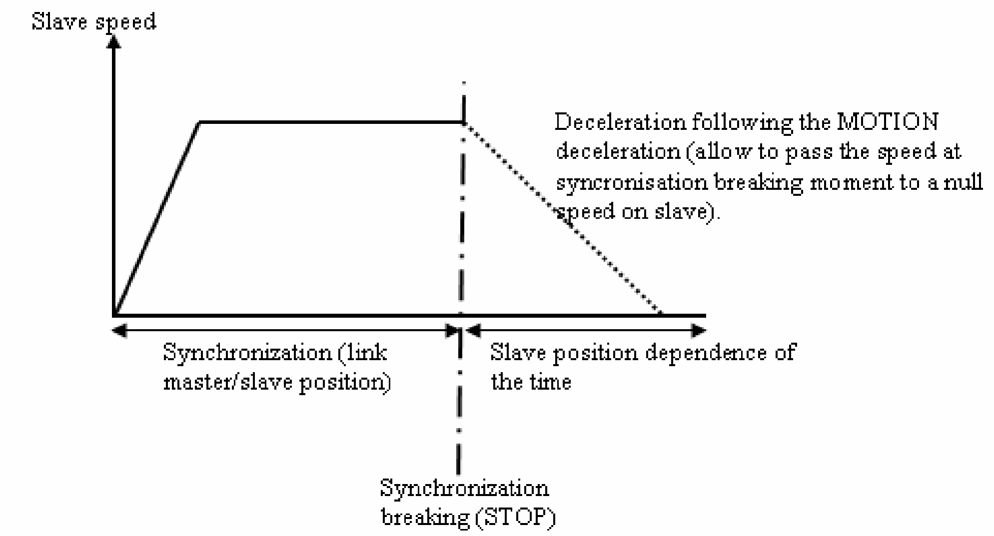
The CAN instructions for synchronization are described in **Appendix \ CANopen \ instructions list**.

7-7-6- Stopping a master / slave link

To stop a synchronise movement, you can use this instructions:

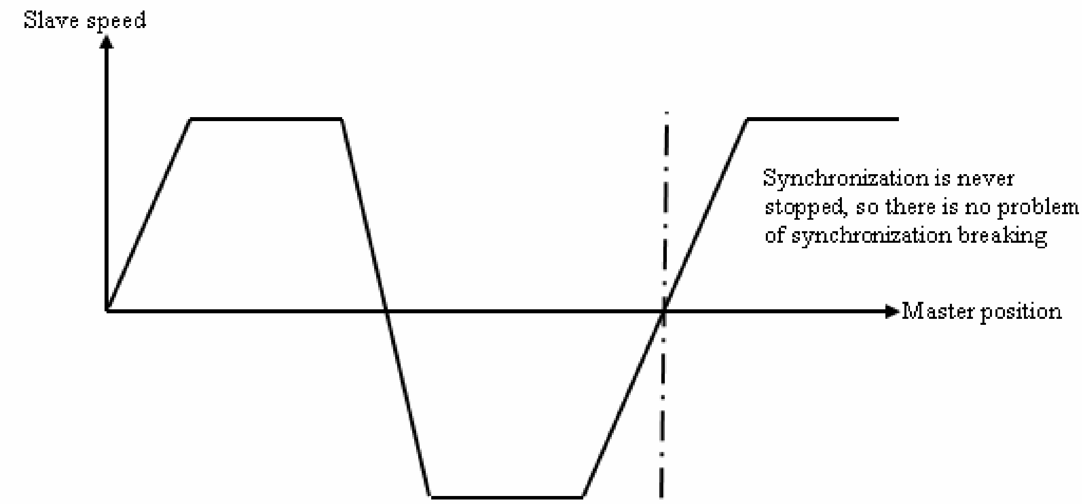
- STOP : finish the movement immediately
- STOPS : finish the movement on master/slave condition
- ENDCAM : stop a cam
- Or an end of movement (eg: came end)

When the synchronised stopping is asked, an iDPL internal deceleration is use to stop slave axis:

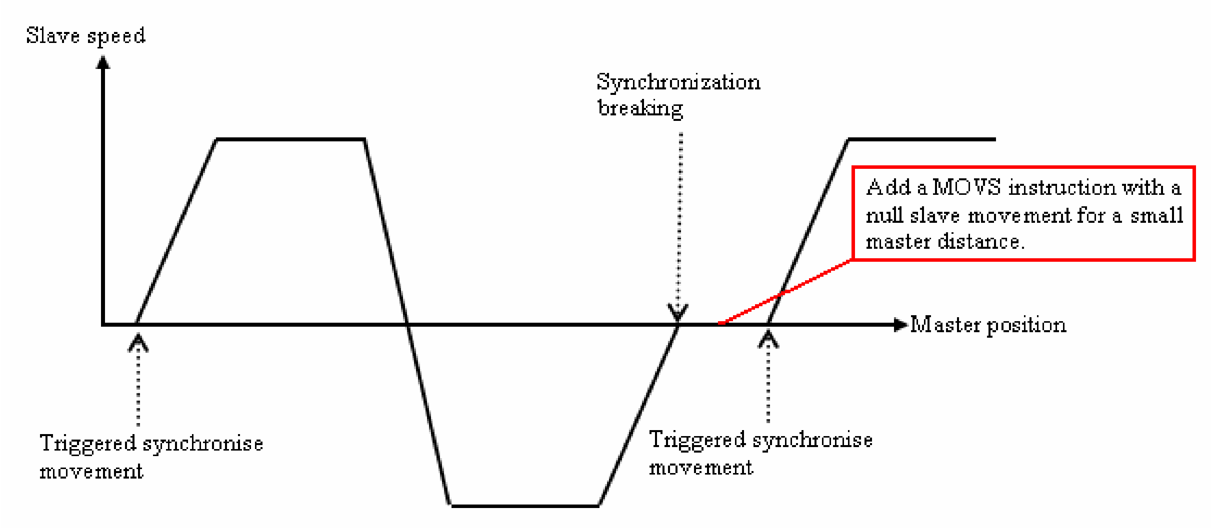


In practice, there is always a deceleration phasing (very low)

- Continuous cycle:



- Triggered synchronise movement:



7-8- Capture

7-8-1- Capture :

Capture allows for the registration of the current axis position on the rising edge of an input signal to the drive.

Capture time:

	Standard input	Quick input
Filter	Filter	600 µs
No filter	150 µs	1 µs

A) CAPTURE1 and CAPTURE2 :

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

Syntax : CAPTURE1 (<Source>, <InputNo>, <Edge>, < 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).

<Edge> 1 for positive edge or 0 negative edge.

< 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>.

Warning : CAPTURE must be re-launched for each new capture. It is forbidden to use the same input and edge with different functions (capture, counter, trigger ...) at the same time.

B) REG1_S and 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.

C) REGPOS1_S and 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.

D) Example :

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

WAITING:
IF REG1_S = ON THEN          'Wait for a capture
  VR1 = REGPOS1_S            'VR1 = value of the captured position
  GOTO STARTCAPTURE
ENDIF
...
GOTO WAITING
```

7-8-2- Automatic axis re-alignment

A) ENABLERECALE – Automatic axis re-alignment

Syntax : ENABLERECALE (<Register Number>, <Initial Position>, <Acceleration>)

Limits : <Initial Position> : between 0 & axis modulo

Accepted types :<Initial Position> : Real

<Acceleration> : Real

Description : This instruction automatically re-aligns the axis position to a sensor.

<Initial Position> indicates the position to be put into the position counter when the sensor is detected.

<Acceleration> as the function MASTEROFFSET, allows an acceleration to be used to apply the offset

Remarks : ENABLERECALE uses parameters of the CAPTURE function that was launched prior this instruction:

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

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

<Edge> 1 for positive edge or 0 negative edge.

< 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>.

ENABLERECALE cancel CAPTURE function.

Example : …

CAPTURE1 (0, 2, 1, 0, 0, 0, 0) ‘Capture on positive edge on input 2

ENABLERECALE (1, 0, 1000) ‘Use parameter of CAPTURE1, set position to 0 and acceleration to 1000

…

DISABLERECALE (0)

B) DISABLERECALE – Cancel axis re-alignment

Syntax : DISABLERECALE (<Axis>)

Limits : <Axis> : 0 = slave axis or 1 = master axis.

Description : This instruction cancels the axis re-alignment to a sensor.

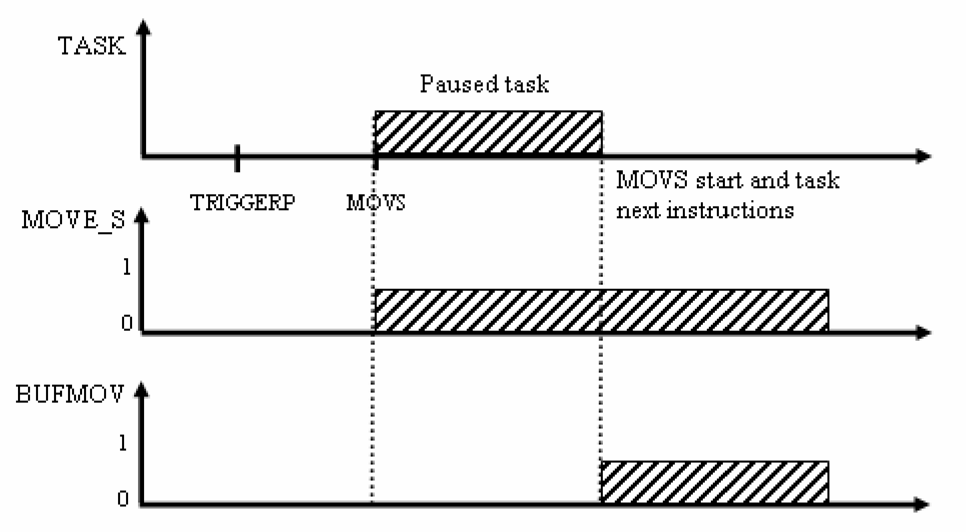
7-9- Triggered movement

7-9-1- Triggered movement

Triggers a movement with an event:

- a master position
- an input
- a capture

On a triggered movement, the task which launches triggered movement is paused until the movement start:



A) TRIGGERP

This instruction indicates that the next movement will be triggered on master position.

Syntax : TRIGGERP (<MasterPos.>, <Edge>)

<MasterPos> real, position in master units.

<Edge> 0 no edge, 1 negative edge, 2 positive edge.

Example :

```
STTA =50
…
TRIGGERP (200,2)
STTA =300     ' Absolute movement to 300
              ' trigger at master position 200
              ' in positive sense
```

B) TRIGGERI

This instruction indicates that the next movement will be triggered on an input edge.

Syntaxe : TRIGGERI (<NumInput>, <Edge>)

< NumInput > from 1 to 16.

< Edge > 0 for negative edge, 1 positive edge.

Exemple : STTA =50

…

TRIGGERI (7,1)

STTA =300 ‘Absolute movement at 300

‘ triggered on positive edge on input 7.

C) TRIGGERC

This instruction indicates that the next movement will be triggered on capture.

Syntaxe : TRIGGERC (<NumCapture>)

< NumCapture> 1 or 2.

Exemple : STTA =50

…

CAPTURE1(0,4,On,10,20,On)

TRIGGERC (1)

STTA =300 ‘Absolute movement at 300

‘ triggered on capture 1.

Warning : TRIGGERC cancels a CAPTURE function, so it is possible to start another. TRIGGERC with capture on inputs 3, 4, 15 and 16 (fast inputs) working as standard inputs.

D) TRIGGERS

This instruction starts the triggered movement without condition.

Needs to be use in another parallel task that had a TRIGGER instruction.

E) TRIGGERR

This instruction cancels the triggered movement without condition.

Needs to be used in another parallel task that had a TRIGGER instruction.

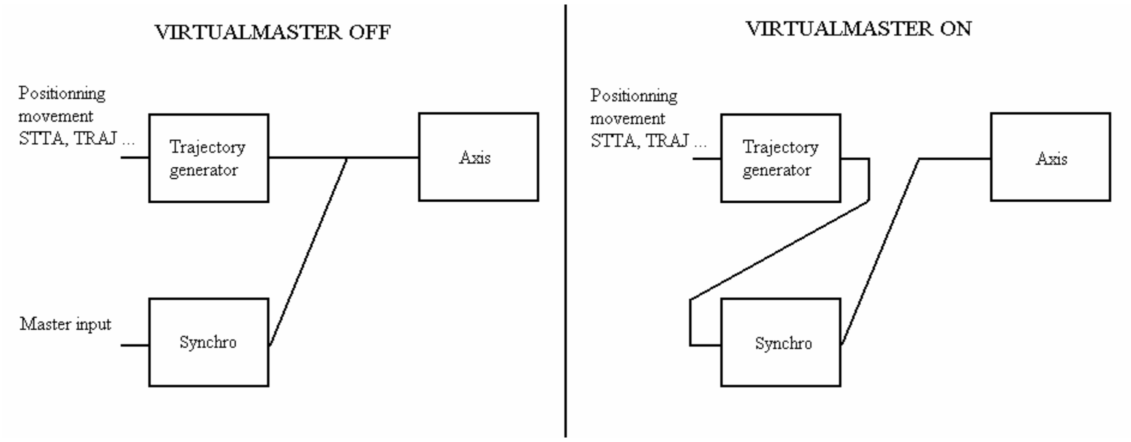
7-10- Virtual master

7-10-1- Virtual master

It is possible to work in master virtual mode to ease development.

A) VIRTUALMASTER – enable/disable virtual master

Syntax: VIRTUALMASTER ON/OFF



Description : This instruction allows the use of a master axis in virtual mode : all positioning instructions (MOVA, MOVR, STTA, SSTR) will "function" for the master axis and the master axis will "move" virtually. It is possible to make synchronised functions between master and slave with using MOVS, GEARBOX ….

Warning : To use the virtual master, select « virtual » source in Motion control \ Master/slave functions.

B) MOVEMASTER_S – Movement status in virtual mode

Syntax : MOVEMASTER_S

Data types : Bit

Description : MOVEMASTER_S is equal to 0 if the 3 following points are true :

- Virtual mode is active.
- The current positioning movement is complete.
- The movement buffer is empty.

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

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

Example: VIRTUALMASTER ON

STTA = VR10

WAIT MOVEMASTER_S = OFF 'Wait until the trajectory on virtual master is finished

C) STOPMASTER – Stop the virtual axis

Syntax : STOPMASTER

Description : This function stops a movement of the virtual master. This function blocks the task until the axis has stopped.

Remarks : If the axis uses a synchronized movement then the axis stops.

The instruction STOPMASTER empties the movement buffer and stops the axis using the current deceleration. This instruction blocks the task until MOVEMASTER_S is not equal to 0.

Example : VIRTUALMASTER ON

MOVS (1, 1, 0, 0)

STTA = 10

…

STOPMASTER ‘ Master stop, axis don’t move more

‘ But synchronising is always enabled

STTA = 10 ‘ Master move and axis start to turn

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 data returned types 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 task 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 input state

It is possible to wait for 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 input 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) and 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
```

Warning: SAVEPARAM and SAVEVARIABLE functions distort time base.

8-3-2- Active wait

A) TIME :

The internal global variable TIME can be used to give an active wait. TIME is a long-integer that represents the number of milliseconds 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 25 days at a time. This is because at power-on TIME is initialized to 0. After 25 days 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 + 5000
Loop  :
  VL3= TIME
  IF VL3<VL2 GOTO Loop              '5 second delay
```

Note : TIME is a long-integer

Warning : TIME does not work in a test.

B) LOADTIMER and TIMER :

The instruction LOADTIMER can be used to give an active wait. This is a real variable that represents the number of milliseconds 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) = 0 the timer has timed-out.

It is possible to use 256 timers simultaneously.

For example :

```
LOADTIMER(VL129)=3000                       '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.

SAVEPARAM and SAVEVARIABLE functions distort time base.

8-4- Counters

8-4-1- Counters

Caution :

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

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

A) 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 5 kHz otherwise it depends on the filter parameter in Parameters / Digital Inputs Outputs .

B) Writing :

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

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

< CounterNo > : Counter number (1 or 2)

<Value> : Value between 0 and 65535

C) 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.

iDPL 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 others can be used elsewhere.

The outputs of a cam box are updated every 300µs.

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.

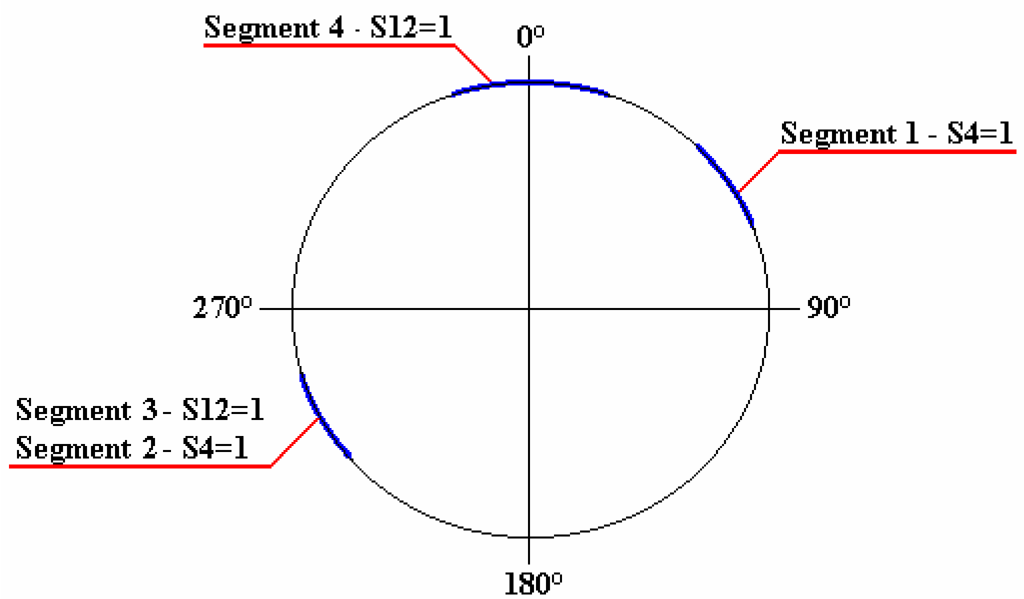
The drive handles up to two cam boxes, each having four segments.

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, the values for the start and the end of the segment are directly tied to scaling and units in the screen Motion control / Configuration / Master.

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



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

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- Operator and instruction list

9-1- Program

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

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

ARCCOS	Inverse Cosine
ARCSIN	Inverse Sine
ARCTAN	Inverse Tangent
COS	Cosine
EXP	Exponential
FRAC	Fractional part
INT	Integer part
LOG	Logarithm
MOD	Modulus

SGN	Sign
SIN	Sine
SQR	Square root
TAN	Tangent

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

A) 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
POSMASTER_S	Actual position of the master axis
VEL	Speed
VEL_S	Actual speed
VEL%	Speed in percent
VELMASTER_S	Return master filter speed

B) 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

C) Synchronization :

CAMNUM_S	Number of the running cam
CAMREADPOINT	Slave position in the cam
CAMSEG_S	Equation number of the running cam
ENDCAM	Stop a cambox
FILTERMASTER	Apply a position filter during a synchronization
ICORRECTION	Correction function
ICORRECTION_S	Status of correction
GEARBOX	Electronic gearbox
GEARBOXRATIO	Modify the ratio of an electronic gearbox
LOADCAM	Load a cam
LOADCAMPOINT	Change a point of a cam
MASTEROFFSET	Shift dynamically the master position
MOVS	Synchronized movement
READCAM	Read a cam point
SLAVEOFFSET	Shift dynamically the slave position
STARTCAM	Launches the execution of a cam
STARTGEARBOX	Start an electronic gearbox
STOPS	Stop synchronization
STOPS_S	Status of the synchronized movement
WRITECAM	Write a cam point

D) Capture

CAPTURE1 and CAPTURE2	Start a position capture
DISABLERECALE	De-activation of re-alignment
ENABLERECALE	Automatic axis re-alignment
REGPOS1_S and REGPOS2_S	Read a captured position

REG1_S and REG2_S	Capture state

E) Triggered move

TRIGGERP	Trigger on master position
TRIGGERI	Trigger on input state
TRIGGERC	Trigger on capture
TRIGGERS	Execute a trigger without condition
TRIGGERR	Cancel a trigger without condition

F) Virtual master

MOVEMASTER_S	Movement state in virtual mode
SSTOPMASTER	Stop movement in virtual mode (without waiting for zero speed)
STOPMASTER	Stop movement in virtual mode
VIRTUALMASTER	Enable or disable virtual master

9-7- PLC

A) 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

B) Analogue I/O

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

C) Timing

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

D) 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

COMCOUNTER	Return the number of exchange frames
DISPLAY	7 segment display
LOADPARAM	Load parameters from Flash
LOADVARIABLE	Load variables from Flash into RAM

READI	Read a FRAM integer
READL	Read a FRAM long integer
READR	Read a FRAM real
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
WRITEI	Write a FRAM integer
WRITEL	Write a FRAM long integer
WRITER	Write a FRAM real

9-10- Alphabetical list

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- ARCCOS – Inverse cosine

Syntax : ARCCOS (<Expression>)

Limits : –1 to +1

Accepted types : Byte, Integer, Long integer, real

Description : This function returns the arccosine of <Expression>.

Remarks : This function returns an angle expressed in radians.

Example : VR1=ARCCOS(0)

See also : SIN, COS and TAN

9-10-19- ARCSIN – Inverse Sine

Syntax : ARCSIN (<Expression>)

Limits : -1 to +1

Accepted types :Byte, Integer, Long integer, real

Description : This function returns the arcsine of <Expression>.

Remarks : This function returns an angle expressed in radians.

Example : VR1=ARCSIN(1)

See also : SIN, COS and TAN

9-10-20- ARCTAN – Inverse tangent

Syntax : ARCTAN (<Expression>)

Accepted types : Byte, Integer, Long integer, real

Description : This function returns the arctangent of <Expression>.

Remarks : The function ARCTAN takes the ratio of two sides of a right triangle and returns the corresponding angle. The ratio is the length of the side opposite the angle divides by the length of the side adjacent to the angle.

Example :

```
VR1=ARCTAN(3)
VR2=ARCTAN(1)
```

See also : SIN, COS and TAN

9-10-21- 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**

9-10-22- AXIS_S – Read the state of the control loop

Syntax : AXIS_S

Description : This instruction is used to read the state of the control loop and returns a value 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**

9-10-23- BUFMOV_S - Number of waiting movements

Syntax : <Variable>=BUFMOV_S

Data types : Byte

Description : This function returns the number of movements waiting in the buffer. The movement 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.

9-10-24- 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**

9-10-25- CAMBOX - Camboxes

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**

9-10-26- 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**

9-10-27- CAMNUM_S – Number of the running cam

Syntax: <Variable>=CAMNUM_S

Accepted types : < Variable>: Integer

Description: this instruction returns the number of the running cam.

Remarks: The returned value is valid only if CAM_S is set.

Example: IF CAMNUM_S=1 THEN GOTO ATTENTE_FIN_CAME_1 ‘ Cam 1 running

IF CAMNUM_S=2 THEN GOTO ATTENTE_FIN_CAME_2 ‘ Cam 2 running

See also: CAM_S, CAMSEG_S

9-10-28- CAMREADPOINT – Slave position in the cam

Syntax : <Slave position>=CAMREADPOINT(<Master position>,<NumCam>)

Description : This intruction allows to calculate the slave positon <Slave position> in the cam, corresponding to the master position <master position>.

Accepted types : < Master position>: real

<Slave position>: real

<NumCam> Number of the selected cam who was loaded before (1 to 5)

Remarks : Return 0 if <Master position> is not in the selected cam.

9-10-29- CAMSEG_S – Equation number of the running cam

Syntax : <Variable>=CAMSEG_S

Accepted types :<Variable> : Integer

Description : this instruction permits to know which equation number of the cam is running.

Remarks : The returned value is valid only if CAM_S is set.

Example : IF CAMSEG_S=1 THEN GOTO ATTENTE_FIN_SEGMENT_1
‘Cam between point 1 and point 2

IF CAMSEG_S=2 THEN GOTO ATTENTE_FIN_SEGMENT_2
‘Came between point 2 and point 3

See also : CAM_S, CAMNUM_S

9-10-30- CAPTURE1 – Position capture

Syntax : CAPTURE1 (<Source>, <InputNo>, <Window>,<Edge>, <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)

<Edge> 1 for positive edge or 0 negative edge.

<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

9-10-31- 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

9-10-32- CLEARMASTER – Set the master encoder position to zero

Syntax : CLEARMASTER

Description : This instruction set to 0 the master encoder position.

Example : CLEARMASTER

9-10-33- COMCOUNTER – Return the number of exchange frames

Syntax : <NB Frames> = COMCOUNTER(X)

Description : This instruction returns the number of exchange frames on the selected bus : 0 for modbus 1 (X1), 1 for modbus 2 (X4), 2 for CANopen and 3 for SDO server (incremented at each SDO request).

Remark : allows the implemetation of a software watchdog and controls lost communication with other device (HMI, drive ...)

Example :

```
TESTCOM :
        LOADTIMER(VL122)=500
        WAIT (TIMER(VL122)=0)
IF OldCounter = COMCOUNTER(1) THEN
                NBErr = NBErr + 1
END IF
        OldCounter = COMCOUNTER(1)
        IF NBErr >3 GOTO ERRCOM
GOTO TESTCOM
```

9-10-34- 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**

9-10-35- COS - Cosine

Syntax : COS(<Expression>)

Accepted types : Expression : real

Description : This instruction returns the cosine of the <Expression>.

Remarks : The result is between -1 and 1.

Example : VR0=COS(3.14159)

See also : SIN, ARCTAN and TAN

9-10-36- COUNTER - Initialize 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

Warning : It is forbidden to use the same input and edge for triggered movement, counter and capture at the same time.

See also : **SETUPCOUNTER**

9-10-37- 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)

9-10-38- 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**)

9-10-39- 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**

9-10-40- 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%**

9-10-41- 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

Warning: SAVEPARAM and SAVEVARIABLE functions distort time base.

9-10-42- DISABLERECALE – Cancel axis re-alignment

Syntax : DISABLERECALE (<Axis>)

Limits : <Axis> : 0 = slave axis or 1 = master axis.

Description : This instruction cancels the re-alignment of an axis to a sensor.

See also : ENABLERECALE

9-10-43- DISPLAY – 7 segment display

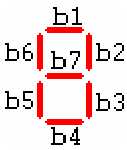
Syntax : DISPLAY <Expression>

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 MSB is not used.

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



9-10-44- ENABLERECALE – Automatic axis re-alignment

Syntax : ENABLERECALE (<Register Number>, <Initial Position>, <Acceleration>)

Limits : <Initial Position> : between 0 & axis modulo

Accepted types : <Initial Position> : Real

<Acceleration> : Real

Description : This instruction automatically re-aligns the axis position to a sensor.

Remarks : ENABLERECALE use the parameters of the CAPTURE function that was launched prior this instruction:

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

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

<Edge> 1 for positive edge or 0 negative edge.

< 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>.

ENABLERECALE cancel CAPTURE function.

Example : …

CAPTURE1 (0, 2, 1, 0, 0, 0, 0) ‘Capture on positive edge on input 2

ENABLERECALE (1, 0, 1000) ‘Use parameters of CAPTURE1, set position to 0 and acceleration to 1000

…

DISABLERECALE (0)

See also : DISABLERECALE

9-10-45- ENDCAM – Stop a cam

Syntax : ENDCAM

Description : The function ENDCAM stops the slave movement at the end of the cycle, while the functions STOP stops it immediately.

Remarks : Warning : If ENDCAM is applied to a cam which has been declared in non-single shot and linked with another one, the cam ends its profile and goes on to the next.

See also : CAM, STOP

9-10-46- EXIT SUB – Exit a subroutine

Syntax : EXIT SUB

Description : This instruction exits a subroutine.

See also : **SUB**

9-10-47- EXP - Exponential

Syntax : EXP (<Expression>)

Accepted types : Expression : real

Description : This function returns ***e*** (natural logarithm base) raised to <Expression> power.

Example : VR0=EXP(2)

See also : LOG

9-10-48- 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 iDPL 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+iDPL***, 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**

9-10-49- FE_S – Following error

Syntax : FE_S

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

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

Example : VR1 = FE_S

See also : **FEMAX_S**

9-10-50- FILTERMASTER – Apply a position filter during a synchronization

Syntax : FILTERMASTER (<Value>)

Description : This function is used to apply a position filter during a synchronization.

Values for <Type> are :

0 : no filter, quick synchronization but risk discontinuous velocity shock if master speed is much less than slave speed.

1 : standard filter (by default)

2 : Thanks to low time constant and advanced filter, the synchronization remains fast and removes a lot of the discontinuous velocity shocks.

3 : Thanks to high time constant and advanced filter, the synchronization remains fast and removes all discontinuous velocity shocks but synchronization loses precision.

4: Interpolation filter for high ratio with **small master speed changes**.

5: Advanced interpolation filter for high ratio with **small master speed changes**, the synchronization remains fast and removes all discontinuous velocity shocks but synchronization loses precision..

9-10-51- 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**

9-10-52- GEARBOX

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

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

Data types : <Numerator> real

<Denominator> real

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

<Reverse> is a Boolean that indicates that the gearbox is reversible.

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**

9-10-53- GEARBOXRATIO

Syntax : GEARBOXRATIO(<Ratio> <Master acc. distance>)

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>.

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

<Master acc. distance > is the distance for master acceleration.

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

Example : GEARBOXRATIO(2)

See also : **GEARBOX**, **STARTGEARBOX**,

9-10-54- 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**

9-10-55- 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 a HALT function, it is recommend to wait for the task to be completely stopped: Wait Status (Task_num) =0

See also : **RUN**, **SUSPEND**, **CONTINUE**

9-10-56- 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 done, the drive calculates the position relative to 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

10 : Position initialization with the absolute position (only in SinCos mode or SSI else initialization to 0)

11 : clear following error

12 : « relative » Home allow to subtract [Reference] to the current position.

[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 adding 16 to <Type> number, the HOME instruction is doing without position modification but the offset is save in HOMEPOS_S variable.

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**

Warning : Input 4 must be declared as HOME function in digital input window for Home on sensor else Home function is cancelled.

9-10-57- 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**

9-10-58- HOMEMASTER- Go to home on master axis

Syntax : HOMEMASTER(<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 : drive is waiting a Top Z on master encoder.

2 : On sensor input : drive is waiting for an input edge on HOME sensor.

3 : On sensor input and Top Z : drive is waiting for an input edge on HOME sensor then a Top Z on master encoder

4 : setup master position to absolute position (with SinCos or SSI else setup master position to 0)

5 : cancel HOMEMASTER without HOMEMASTER_S changing

[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

HOMEMASTER (3,VR0) ‘Go home using mode 3 and a home position of 100

Note : If adding 16 to <Type> number, the HOME instruction is doing without position modification but the offset is save in HOMEPOSMASTER_S variable.

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

HOMEMASTER(2)

See also : **HOME_S**

Warning : Input 4 must be declared as HOME function in digital input windows for Home on sensor else Home function is cancelled.

9-10-59- HOMEMASTER_S - Read master homing status

Syntax : HOMEMASTER_S

Description : This function reads the master homing status

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

Example : IF HOMEMASTER_S = OFF GOTO Next

Next :

See also : HOMEMASTER

9-10-60- ICORRECTION – Correction function

Syntax : ICORRECTION(<Dist.master>,<Dist.slave>, <Dist. accel>)

Units : <Dist.master>, <Dist.slave> : user unit (Ex : mm, degree,…)

<Dist.accel>: user unit /s²

Accepted types : < Dist.master>, <Dist.slave>, <Dist.accel>: real

Description : This function applies a correction movement to the slave axis during the distance of master axis.

Remarks : The slave axis must be linked to the master axis by a synchronized function before the execution of the correction instruction. With the synchronized movement of the slave axis, the next movement is superposed. During the distance of the master axis, a movement <Dist.slave> is added with an acceleration and a deceleration on a <Dist accel>.

Warning : All other ICORRECTION functions are ignored if one correction function is running or if <Dist.master > is null .

9-10-61- ICORRECTION_S – Correction status

Syntax : <Variable> = CORRECTION_S

Accepted types : <Variable> : bit

Description: This function returns the status of the running correction: return 1 if ICORRECTION is running else return 0.

9-10-62- IF

Syntax 1: IF <Condition> GOTO {<Label>}

Syntax 2:

```
IF < Condition > THEN
 <Instructions1>
   ...
END IF
```

Syntax 3:

```
IF < Condition > THEN
<Instructions1>
   ...
ELSE
   <Instructions2>
   ...
END IF
```

Description : The keyword IF begins a control structure. IF...THEN...ELSE...END IF. It must appear before all other part of the structure. <Condition> must be a Boolean expression.

If <Condition> is true then <Instructions1> are executed.

If <Condition> is false then <Instructions2> are executed.

Remarks : <Condition> must be a Boolean expression.

Example :

```
IF VR1=150 GOTO NEXT
IF VR1<150 THEN
 VR1 = VR1 – 1
END IF
```

9-10-63- 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**

9-10-64- 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**

9-10-65- 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**

9-10-66- 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**

9-10-67- LOADCAM – load a cam

Syntax: LOADCAM (<NumberCam>, <Absolute>, <Table>, <Number>, <SingleShot>, <Reversible>, <Direction>, <MasterGain>, <SlaveGain>, <NumberNextCam>, <NumberPreviousCam>)

Description: this instruction loads a cam in the drive.

Limits : <NumberCam>: 1 to 5

<Absolute> : 1 for absolute cam else 0

< Table >: First element of the table to define the cam (0 to 511)

<Number>: Number of elements of the table to define the cam (2 to 512)

<SingleShot>: Define the automatical re-looping of the cam:

↳ 0: Re-looping cam, it will be stopped only when the stop instruction will be executed.

↳ 1: Single-shot cam

<Reversible>: Inducates if the <Slave> must follow the master in both directions.

↳ Input 0 for a non-reversible cam: if the master moves in the opposite direction to that defined in <Direction>, the slave stops. It will start off again when the master goes in the correct sense and passes by the position where the slave stopped.

↳ Input 1 for a reversible cam: The slave follows its cam profile whatever the master direction.

<Direction>: If the cam is not reversible, you must indicate the usual direction of the master. Input 0 for no direction, 1 for a negative direction, 2 for a positive direction.

<MasterGain>: Applied coefficient to cam master position (default value 1).

<SlaveGain>: Applied coefficient to cam slave position (default value 1).

<NumberNextCam>: Input 0 if the cam must not be followed by another one. Otherwise input the number of the next cam, from 1 to 5.

<NumberPreviousCam>: Input 0 if the cam will not start at the end of another one. Otherwise input the number of the previous cam (from 1 to 5).

See also : STARTCAM

9-10-68- LOADCAMPOINT – Change a point of a cam

Modify a cam point in FRAM memory.

Syntax : LOADCAMPOINT (<NumCam>, <NumPoint>, <FRAMIndex>)

< NumCam > : Number of the cam loaded previously (from 1 to 5).

< NumPoint > : Number of the cam point to modify (from 1 to NB cam point).

< FRAMIndex > : Address of the point in FRAM(from 0 to 511) to send in the target cam point.

Warning: This instruction blocks the task (LOADCAMPOINT can only be done if the cam is not between previous and next < NumPoint > point). This instruction gives an iDPL error if no cam has been loaded before.

9-10-69- LOADPARAM – Reload the drive parameters

Syntax : LOADPARAM

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

See also : **SAVEPARAM**

9-10-70- 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**

9-10-71- 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)=3000 'Load a time of 3000ms in variable VL129

See also : **TIMER**

Warning: SAVEPARAM and SAVEVARIABLE functions distort time base.

9-10-72- LOG - Logarithm

Syntax : LOG (<Expression>)

Accepted types : Expression : real

Description : Returns the natural logarithm of <Expression>

Example : VR0=LOG(1.2)

See also : EXP

9-10-73- 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

LOOP ON function allow to ignorate E2, E7 and E8 errors.

9-10-74- MASTEROFFSET – Dynamically shift the master position

Syntax : MASTEROFFSET(<Offset>,<Acceleration>)

Description : This instruction dynamically shifts the master position for an absolute cam.

Limits : <Offset>: Between 0 and the master modulo

Accepted types : <Offset> : Real

<Acceleration> : Real

Remark : <Offset> : Offset value to apply

<Acceleration> Acceleration used to apply the offset (increment/T0²).

The dephasing is directly applicate if the synchronised movement is not running or if axis in not enable.

9-10-75- 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
```

9-10-76- 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
```

9-10-77- 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**

9-10-78- MOVE_S – Movement status

Syntax : MOVE_S

Data types : Bit

Description : This function indicates if the axis is moving (simple or synchronized movement).

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 positioning 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 synchronized 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
```

Warnings: In VIRTUALMASTER mode, MOVE_S is null if this 3 points are true:

The following error is within the positioning window.

The movement buffer is empty.

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

9-10-79- MOVEMASTER_S –Movement status in virtual mode

Syntax : MOVE MASTER_S

Data types : Bit

Description : MOVE MASTER_S is equal to 0 if the 3 following points are true :

Virtual mode is active.

The current positioning movement is complete.

The movement buffer is empty.

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

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

Example: VIRTUALMASTER ON

STTA = VR10

WAIT MOVEMASTER_S = OFF 'Wait until the trajectory on virtual master is finished

9-10-80- 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**

9-10-81- MOVS - Synchronized movement

Syntax 1 : MOVS (<MasterDistance>, <SlaveDistance>, <AccelerationDistance>, <DecelerationDistance>)

Description: This instruction is used to link a slave axis to a master axis during a certain distance of the master axis with acceleration and deceleration phases on the slave axis (if the buffer of movement is not full)

Accepted types: MasterDistance, SlaveDistance, AccelerationDistance, DecelerationDistance : real

Exemple : MOVS (20, 10, 0, 0)

9-10-82- NEXTTASK

Syntax : NEXTTASK

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

9-10-83- 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**

9-10-84- 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 the same type.

Example : VI12=VI12 OR 000FFh

See also : **AND**, **NOT**, **XOR** and **IF**

9-10-85- 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**

9-10-86- 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**

9-10-87- 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**

9-10-88- 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**

9-10-89- 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**

9-10-90- 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**

9-10-91- POSMASTER_S – Actual position of the master axis

Syntax : <Expression> = POS_S

Data types : Real

Description : This function returns the actual position of the master 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=POSMASTER_S
    IF VR1<50 GOTO Loop
STOP            'Stop slave when master is at 50
OUT(5) = 0      'Clear output 5
```

9-10-92- 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
```

9-10-93- READCAM – Read a cam point

Syntax : <VRx>=ReadCam(<Index>, <Sub index>)

Description : Reads a cam point in the FLASH memory

Limits : <Index> from 0 to 511, cam point number in FRAM

< Sub index> from 0 to 3, cam point parameter

↳0 for master position

↳1 for slave position

↳2 for master tangential

↳3 for slave tangential

<VRx> from VR0 to VR255

See also : WRITECAM

9-10-94- READI - Read a FRAM integer

Syntax : <VIn> = READI (<Address>)

Limits : <Adress> : from 0 to 4095

n from 0 to 255

9-10-95- READL - Read a FRAM long integer

Syntax : <VLn> = READL (<Address>)

Limits : <Address> : from 0 to 4094

n from 0 to 255

Warning : Reading or writing a long integer, use 2 consecutively memory (address and adresse+1).

9-10-96- READR - Read a FRAM real

Syntax : <VRn> = READR (<Adresse>)

Limits : <Adrese> : from 0 to 4094

<VRn> : from 0 to 255

Warning : Reading or writing a real, use 2 consecutively memory (address and adresse+1).

9-10-97- 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.

9-10-98- REG1_S – Position capture status

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	

9-10-99- REGPOS1_S – Last Capture1 position

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	

9-10-100- REPEAT ... UNTIL

Syntax : REPEAT

{<Instructions>}

UNTIL <Condition>

Description : This structure allows to the system to execute a list of instructions in a loop as long as the given condition is wrong.

Remarks : In the structure REPEAT ... UNTIL the <Instructions> are executed at least once even if the condition is true. The execution of this instruction launches the execution of the next task.

Example :

```
VEL% = 100   ' High velocity
STTA = 2000  ' move to 2000
REPEAT
VR0 = POS_S
    IF VR0>1000 THEN
    VEL% =50            ' Medium speed at position 1000
    END IF
UNTIL NOT MOVE_S        ' Re-loop until end of movement
```

9-10-101- RESTART – Restart the system

Syntax : RESTART

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

9-10-102- 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

Goto Start

See also : CONTINUE, HALT, SUSPEND

Warning: After a HALT function, it is recommend to wait for the task to be completely stopped: Wait Status (Task_num) =0

9-10-103- 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.

SAVEPARAM and SAVEVARIABLE functions distort time base and cancel CAN position sending.

9-10-104- 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.

SAVEPARAM and SAVEVARIABLE functions distort time base and cancel CAN position sending.

9-10-105- 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	Error. 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).

9-10-106- 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 5 kHz, otherwise it depends on the Filter parameter in Parameters / Digital Inputs Outputs.

9-10-107- SGN - Sign

Syntax : SGN (<Expression>)

Accepted types : Expression : Long integer, real

Description : This function returns a real equal to –1 for the negative numbers, 1 for the positive numbers and 0 for the number zero.

Example : VR0=SGN(10) 'Result : VR0=1

9-10-108- SIN - Sine

Syntax : SIN (<Expression>)

Accepted types : Expression : real

Description : This instruction returns the sine of <Expression>. <Expression> is expressed in radians.

Remarks : The result is between -1 and 1.

See also : COS, ARCTAN, TAN

9-10-109- SLAVEOFFSET – Dynamically shift the slave position

Syntax : SLAVEOFFSET(<Offset>,<Acceleration>)

Limits : <Offset>: Between 0 and the slave modulo

Accepted types : < offset>: Real

<Acceleration>: Real

Description: This instruction dynamically shifts the slave position in an absolute cam.

Remark : <Offset>: Offset value to apply

<Acceleration> Acceleration used to apply the offset.

The dephasing is directly applicate if the synchronised movement is not running or if axis in not enable.

9-10-110- SQR – Square root

Syntax : SQR (<Expression>)

Accepted types : Expression : real

Description : This function returns the square root of <Expression>.

Example : VR0=SQR(2)

9-10-111- 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**,

9-10-112- SSTOPMASTER - Stop movement in virtual mode (without waiting for zero speed)

Syntax : SSTOPMASTER

Description : This function stops a movement of the virtual master. This function does not block the task.

Remarks : If the axis uses a synchronized movement then the axis stops.

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

Example : VIRTUALMASTER ON

MOVS (1, 1, 0, 0)

STTA = 10

…

SSTOPMASTER ‘ Master stop, axis does not move further

WAIT MOVEMASTER_S = 0 ‘ But synchronising is always enabled

STTA = 10 ‘ Master moves and axis starts to turn

9-10-113- 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**

9-10-114- STARTCAM – Launches the execution of a cam

Syntax : STARTCAM(<NumberCam>)

Limits : <NumberCam> : 1 to 5

Accepted types : <NumberCam> : Byte

Description : this instruction launches the execution of a cam.

See also : LOADCAM

9-10-115- STARTGEARBOX – Start electronic gearbox

Syntax : STARTGEARBOX (<Master acceleration dist.>)

Description : 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>, with <Numerator> and <Denominator> defined in the instruction GEARBOX.

Accepted types : < Master acceleration dist.> is real.

With Ratio that corresponding to the value of GEARBOXRATIO.

See also : **GEARBOX**, **GEARBOXRATIO**

9-10-116- 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
```

9-10-117- 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**

Warning : In virtual mode, STOP does not stop positioning movements (STTA, TRAJA …)

9-10-118- 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**

9-10-119- STOPMASTER – stop movement in virtual mode

Syntax : STOPMASTER

Description : This function stops the movement of a virtual master. This function blocks the task until the axis has stopped.

Remarks : If axis uses a synchronized movement then axis stops.

The instruction STOPMASTER empties the movement buffer and stops the axis using the current deceleration. This instruction blocks the task until MOVEMASTER_S is not equal to 0.

Example :

```
VIRTUALMASTER ON
MOVS (1, 1, 0, 0)
STTA = 10
…
STOPMASTER        ‘ Master stop, axis does not move further
                  ‘ But synchronising is always enabled
STTA = 10         ‘ Master move and axis start to turn
```

9-10-120- STOPS_S – status of the synchronised movement

Description : This instruction can be used only if STOPS instruction has been called previously. This flag indicates if the slave position given by the STOPS has been arrived at. This flag is reset after it has been read.

Return 1 if :

- If the demanded slave position is impossible to achieve (e.g. if it has already been passed.)
- If slave speed is null (during a constant phase).

Else return 0

Syntax : VF0 = STOPS_S

Example : MOVS (20, 10, 0, 0)

...

STOPS (20, 105)

WAIT MOVE_S=0

IF STOPS_S=1 GOTO ERRSTOPS

9-10-121- STOPS – stop MOVS instruction

Description : When the master axis arrives at <MasterPos.>, slave axis starts deceleration until <SlavePos.>.

Syntax : STOPS (<MasterPos.>, <SlavePos.>)

<MasterPos.> is a real in the master unit.

<SlavePos.> is a real in the slave unit.

Example : STOPS (20, 105) ‘When the master arrives at position 20,

‘ the slave axis will decelerate until position 105 on the slave axis

Warning : The call of STOPS instruction resets the STOPS_S flag.

9-10-122- STTA – Start absolute movement

Syntax : STTA = <Distance>

Data types : Real

Description : Starts 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**

9-10-123- STTI – Start infinite movement

Syntax : STTI + or -

Description : Starts an infinite movement.

Remarks : The system immediately executes 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**

9-10-124- STTR – Start a relative movement

Syntax : STTR = <Distance>

Data types : Real

Description : Starts 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**

9-10-125- SUB .. END SUB – Subroutine

Syntax : SUB <Name>

Description : This keyword defines the start of a subroutine. Used in conjunction with END, it is 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
```

9-10-126- 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**

9-10-127- TAN - Tangent

Syntax : TAN (<Expression>)

Accepted types : Expression : real

Description : This instruction returns the tangent of <Expression>. <Expression>is an angle expressed in radians.

Example : VR0=TAN(3.14)

See also : SIN, ARCTAN,TAN

9-10-128- 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 millisecond since the last power-on.

Example :

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

Warning : TIME does not work in a test.

SAVEPARAM and SAVEVARIABLE functions distort time base.

9-10-129- 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)=3000 'Load a time of 3s
WAIT (TIMER(VL122)=0)    'Wait until the time has elapsed
```

Warning: SAVEPARAM and SAVEVARIABLE functions distort time base.

9-10-130- 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	

9-10-131- 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	

9-10-132- TRIGGERC - Trigger on capture

Syntax : TRIGGERC (<NumCapture>)

< NumCapture> 1 or 2.

Description : This instruction indicates that the next movement will be triggered on capture.

Exemple : STTA =50

…

CAPTURE1(0,4,On,10,20,On)

TRIGGERC (1)

STTA =300 ‘Absolute movement at 300

‘ triggered on capture 1.

Warning : TRIGGERC cancels a CAPTURE function, so it is possible to start another. TRIGGERC with capture on input 3, 4, 15, 16 (fast inputs) works like standard inputs.

9-10-133- TRIGGERI – Trigger on input state

Syntax : TRIGGERI (<NumInput>, <Edge>)

< NumInput > from 1 to 16.

< Edge > 0 for negative edge, 1 positive edge.

Description : This instruction indicates that the next movement will be triggered on an input edge.

Example :

```
STTA =50
…
TRIGGERI (7,1)
STTA =300     'Absolute movement at 300
              ' triggered on positive edge on input 7.
```

Warning : It is forbidden to use the same edge and input at the same time as counter, capture and trigger functions.

9-10-134- TRIGGERP – Trigger on master position

Syntax : TRIGGERP (<MasterPos.>, <Edge>)

<MasterPos> real, position in master unit.

<Edge> 0 no edge, 1 for negative edge, 2 positive edge.

Description : This instruction indicates that the next movement will be triggered on master position.

Example :

```
STTA =50
…
TRIGGERP (200,2)
STTA =300     ' Absolute movement to 300
              ' trigger at master position 200
              ' in positive sense
```

9-10-135- TRIGGERR – Cancel a trigger without condition

This instruction cancels the triggered movement without condition.

Needs to be used in another parallel task that had the TRIGGER instruction.

9-10-136- TRIGGERS – Execute a trigger without condition

This instruction starts the triggered movement without condition.

Needs to be used in another parallel task that had the TRIGGER instruction.

9-10-137- 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**

9-10-138- VEL_S – Actual speed

Syntax :	VEL_S
Description :	This function returns the current velocity.
Example :	STTA = 100
	IF VEL_S<50 GOTO Stop_1
See also :	POS_S

9-10-139- 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 the speed parameter in screen Motion control / Configuration / Speed profile.
Example :	VB0 = 50
	VEL% = VB0

See also : **ACC%**, **DEC%**

9-10-140- VELMASTER_S – Return master filter speed

Syntax : VELMASTER_S

Description : This function returns the master filter speed.

Example : GEARBOX(1,1)

IF VELMASTER_S<50 GOTO Stop_1

See also : VEL_S

9-10-141- VERSION – OS (Firmware) version

Syntax : <VI _XX>=VERSION

Description : This function returns the version of the operating system.

9-10-142- VIRTUALMASTER – Enable/disable virtual master

Syntax: VIRTUALMASTER ON/OFF

Description : This instruction allows the master axis to be used in virtual mode: all positioning instructions (MOVA, MOVR, STTA, SSTR) will "take place" for the master axis and the master axis will "move" virtually. It is possible to have synchronised functions between master and slave using MOVS, GEARBOX ….

Warning : To use the virtual master, select « virtual » source in Motion control \ Master/slave functions.

9-10-143- WAIT – Wait for a condition

Syntax : WAIT <Condition>

Description : Waits until the condition is true.

Example : WAIT INP(11)=On 'Passive wait

9-10-144- WRITECAM – Write a cam point

Syntax: WriteCam(<Index>, < Sub index>)=<VRx>

Description : Writes a cam point in FLASH memory

Limites : <Index> from 0 to 511, cam point number in FRAM

< Sub index> from 0 to 3, cam point parameter:

↳0 for master position

↳1 for slave position

↳2 for master tangential

↳3 for slave tangential

<VRx> from VR0 to VR255

See alson : READCAM

9-10-145- WRITEI - Write a FRAM integer

Syntax : WRITEI (<Address>) = <VIn or value >

Limits : <Address> : from 0 to 4095

n from 0 to 255

9-10-146- WRITEL - Write a FRAM long integer

Syntax : WRITEL (<Address>) = <VLn or value >

Limits : <Address> : from 0 to 4094

n from 0 to 255

Warning : Reading or writing a long integer use 2 consecutively memory (address and adresse+1).

9-10-147- 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

9-10-148- WRITER - Write a FRAM real

Syntax : WRITER (<Address>) = <VRn or value>

Limits : <Address> : from 0 to 4094

n from 0 to 255

Warning : Reading or writing a real use 2 consecutively memory (address and adresse+1).

9-10-149- 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

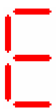
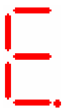
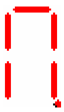
10-1-1- Message descriptions

A) At power-on of the drive:

1. **BOOT initialization phase :**

Before BOOT initialization, the display is:

In the case of an initialization error, you can have these faults:

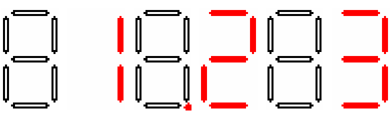
-  : Checksum error of the boot/OS sector.
-  : Error loading OS
-  : Internal error

2. **OS initialization phase :**

The segments light quickly in the following order:

 →

At the end of the OS initialization, version is displayed:



In this example, the version is 1.23

3. **After initialization :**

The output ‘Drive Ready' (S1) is active. If iDPL is in use : the automatic tasks are launched and there should remain only the decimal point that flashes.

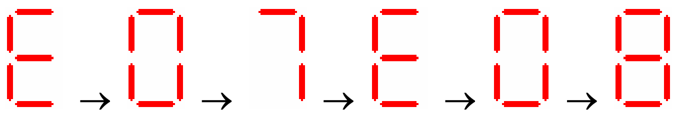
- If iDPL is not in use the segments of the display light in sequence as the motor shaft turns
- If iDPL is in use only the decimal point remains. The segments can be modified using the instruction ‘Display’ in an iDPL task.

B) During drive operation :

1. 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 :

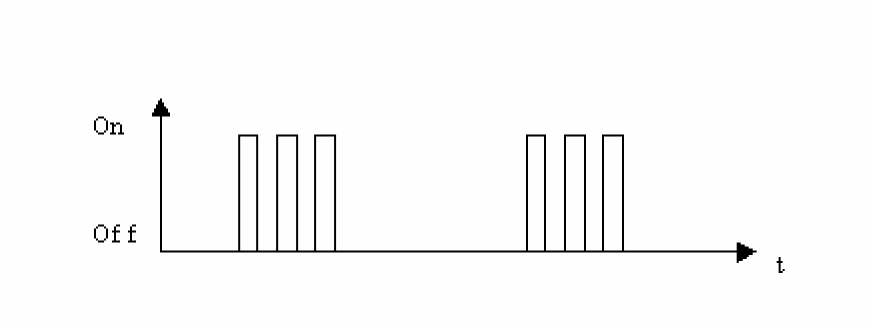


2. On the removal of a fault:

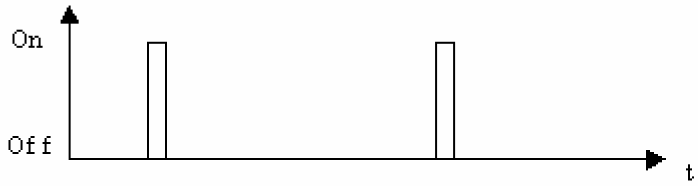
Removal of the error number and return to a normal display (as after the initialization)

 Flashing decimal point :

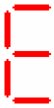
- If system serial connection present (RTS high) :

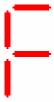


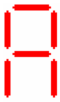
- If no system serial connection:

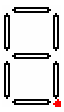


C) During loading of the OS :

 Erase : clear FLASH

 Flash : write flash

 Read : read flash

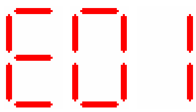
 Reboot

D) During Flash operations :

 is displayed during the Flash operations (SAVEPARAM, SAVEVARIABLE ...).

10-1-2- Error messages

A) 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.

E02	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, DC Bus voltage less than drive's parameter) and also when drive is enabled (DC Bus voltage less than 250V).
E03	I^2t motor : I^2t motor detected.
E04	Over-current : a current greater than the maximum current has been detected.
E05	Short-circuit : a short-circuit between phases or between a motor phase and earth has been detected.
E06	Temperature IGBT : maximum temperature attained in the drive.
E07	Temperature motor : maximum motor temperature attained.
E08	Resolver fault : Resolver feedback or absolute encoder or SinCOS signals defective.
E09	Invalid parameters : checksum error on the drive parameters or parameters not initialized.
E10	Drive type error : the parameter file does not correspond to the drive type or parameters not configured.
E11	iDPL error : an error has been detected during the execution of the iDPL tasks (division by zero, incorrect instruction, CAM or synchro. movement error …).
E12	Following error : the maximum following error has been exceeded. Contact technical support.
E13	FLASH memory error: writing impossible. Contact technical support.

E 14	FPGA error : loading not possible or CAN communication error. Contact technical support.
E 15	Over velocity : motor velocity is higher than nominal speed in torque mode.
E 16	Feedback saturation error. Feedback or SinCos signals are too high.
E 17	Auxiliary supply error. Control 24V auxiliary supply.

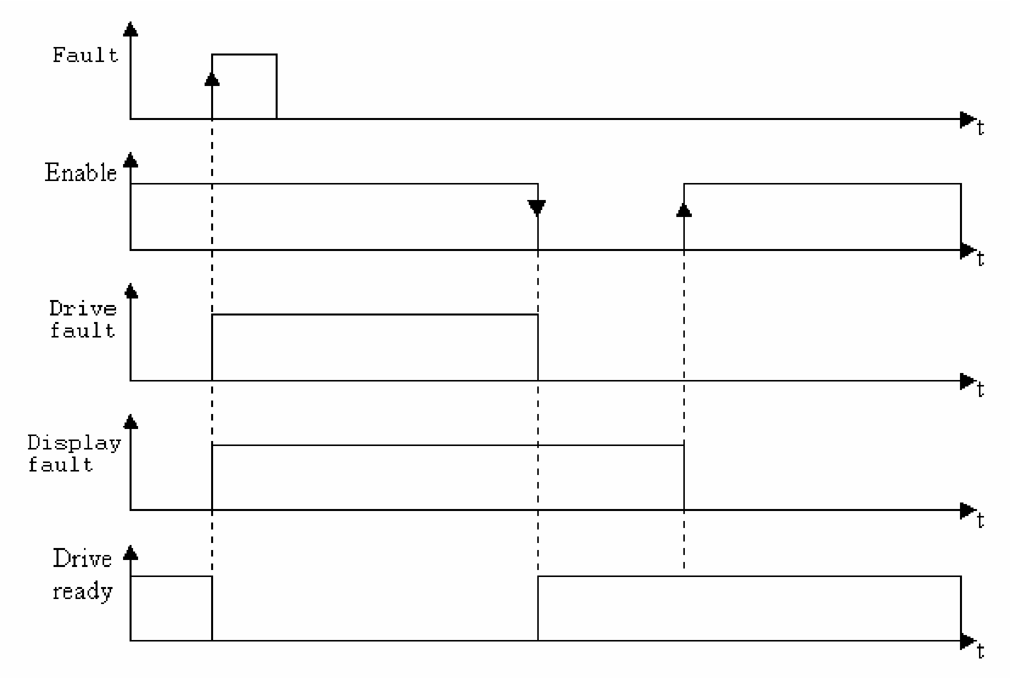
B) List of iDPL errors:

Error 1 :	Instruction illegal. Flash corruption and/or compilation error.
Error 2 and 3 :	Cam allocation error. Internal OS error.
Error 4 :	Impossible to calculate a cam point. Reduce master length.
Error 5 :	Illegal function. Flash corruption and/or compilation error.
Error 6 :	Divide by 0.
Error 7 :	Cam number error in LOADCAM function
Error 8 :	FRAM offset error (value not between 0 and 4095)
Error 9 :	Cam gain error (master or slave distance is negative)
Error 10 :	Invalid task number

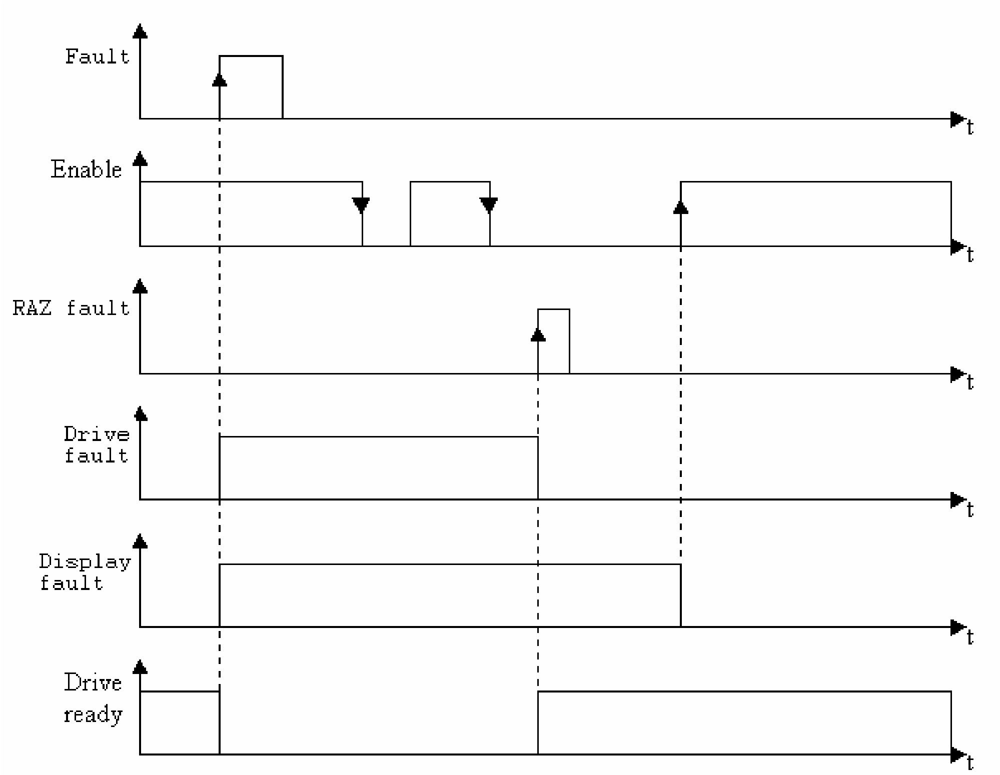
double click on error message to open iDPL task and set cursor at the faulty line.

C) 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 responded 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 initialization. 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.

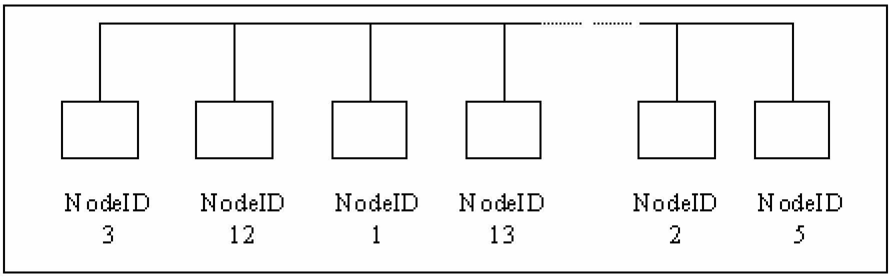
The profile of a device defines the parameters necessary for communication. The contents of this profile are specified by the device manufacturer. Devices with the same profile are directly interchangeable. Most parameters are described by the manufacturer. The profile may also contain empty slots for 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.

The variables mapping used during the PDO type exchanges permits to use in an optimal way the current bandwidth of the bus. CANopen determinates default values of all the parameters.

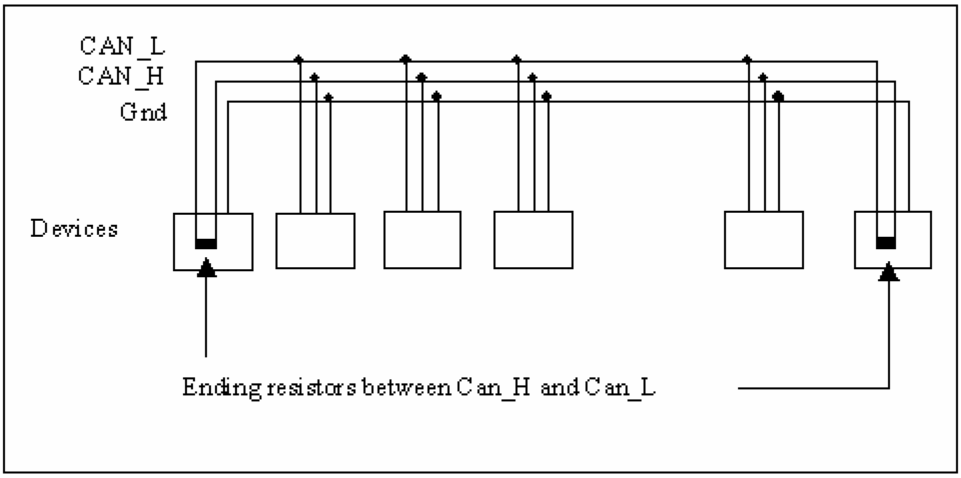
C) Network configuration

The CANopen network is made of several devices, each of them can be master and slave. They are identified in the network by an arbitrary number, called Node-Id. This parameter must be unique: two different devices of the CANopen network can not have the same Node-Id. This Node-Id is very important, it is the real identity card of the peripheral on the CANopen network.



Example of CANopen network configuration

The wiring is as follows:



Wiring of a Can Open network

Warning: Do not forget the termination resistors at each end of the CANopen network.

D) Types of messages

There are two main kinds of messages sent on the CANopen network:

- The SDO are transmitting data
- The PDO are transmitting events

10-2-2- IMDCANI card

A) Presentation - IMDCANI card

The different parameters of the IMD drive and the data tables are stored in a two-dimensional array, called the dictionary.

Each data or parameter is defined by an address index, and a sub-index address.

The IMD drive can communicate with another device of the network by different ways. It can leave data at the disposal of other devices by writing them in its local table: any other peripheral can then read and write to this local table. This is the way used, for example, to communicate with an intelligent operator terminal Dialog 80 or 640.

The IMD drive can also read and write to a local table of another device. This operation is done with the instruction CanRemote.

B) Characteristics

↳ An SDO default server to set the parameters of the remote board by a supervisor.

↳ An SDO client to access the variables and peripheral parameters of devices such as displays, PLC, PC boards.

↳ 8 PDO in emission to drive the outputs of the I/O modules or signal an event to another device.

↳ 8 PDO in reception to receive the inputs of the I/O modules or signal an event from another device.

↳ Direct access functions to the CAN bus to send and receive specific messages such as the functions NMT et DBT.

↳ Node guarding functions.

C) Connections

X2 & X3: Extension: Optional communications port

N°	Module CANopen X2	Module CANopen X3
1		
2		
3		
4		
5	GND	GND
6		
7	CAN_L	CAN_L
8	CAN_H	CAN_H
⚠	SHIELD - Raccorder la tresse blindée sur le corps du SUBD	

- X2 and X3 are identical and have the same connections. They provide for easier network connections.

- Node Address : For RS422, RS485 and CANopen, the NodeID corresponds to the first five dipswitchs + 1

  Ex: dipswitchs: 1 -> ON, 2 -> OFF, 3 -> ON, 4 -> OFF, 5 -> OFF

  Dipswitchs value = 1 + 4 = 5

  NodeID = 5 + 1 = 6

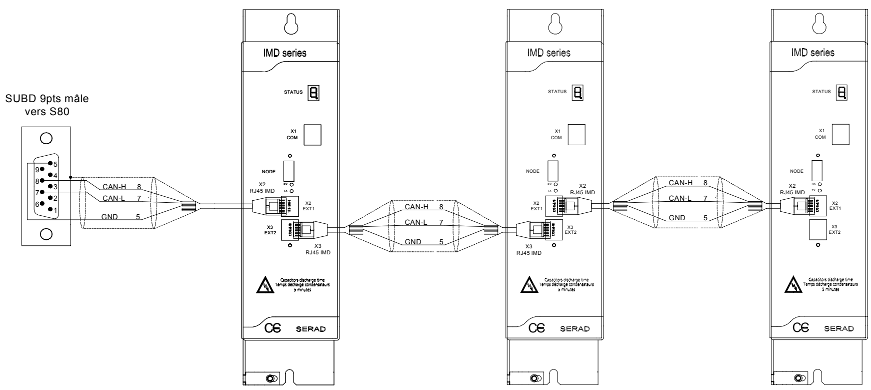
- Put on Dipswitch 6 to activate termination resistor (120Ω).

  RS232 communication allows communication with only 1 device (ex: 1 PLC and 1 IMD drive).

a) Maximum transmission speed regarding the length of the CANopen network

Maximum transmission speed	Network length
10k to 125 kBaud	500 m
250 kBaud	250 m
500 kBaud	100 m
800 kBaud	50 m
1 Mbaud	25 m

b) Example with 3 IMD drive and 1 SUPERVISOR :



D) Diagnostics on the CANopen network

LED CAN Rx/Tx:

Flashing: light is function of the traffic on the CANopen bus (its intensity can be very low or high)

E) CANopen dictionary

The drive can use both the SDO and PDO modes to allow reading from and writing to its parameters and variables or to another CANopen device.

Index	Sub-idx	Nom	Type	Attr.	Défaut	Description
1000	0	Device type	32 bits non signé	ro	403	type d'appareil
1001	0	Error register	32 bits non signé	ro	0	registre d'erreur interne
1002	0	Manufacturer Status Register	32 bits non signé	ro	0	registre d'etat spécifique au constructeur
1003	0	predefined error field	8 bits non signé	ro	1	nombre d'erreurs apparues
	1	actual error	32 bits non signé	ro	0	dernière erreur apparue
1004	0	number of PDO's supported	32 bits non signé	ro	00080008h	Nombre de PDO supporté
	1	Number of synchronous PDO	32 bits non signé	ro	0	Nombre de PDO synchrone supporté
	2	Number of asynchronous PDO	32 bits non signé	ro	00080008h	Nombre de PDO asynchrone supporté
1005	0	COB-ID	32 bits non signé	rw	00000008h	COB-OD SYNC message
100B	0	Node ID	32 bits non signé	ro	aucune	N° de noeud local
100C	0	Guard time	16 bits non signé	rw	aucune	durée en ms
100D	0	Life time factor	8 bits non signé	rw	aucune	Timeout = Guard time x Life time factor
100E	0	Node guarding ID	32 bits non signé	rw	700h + NodeID	COB-ID Nodeguarding
100F	0	Number of SDO's supported	32 bits non signé	ro	00010001h	Nombre de SDO supporté
1200	0	Number of elements	8 bits non signé	ro	2	paramètre du 1er SDO serveur
	1	SDO receive COB-Id	32 bits non signé	ro	600h+node-ID	COB-ID de récéption du 1er SDO serveur
	2	SDO transmit COB-ID	32 bits non signé	ro	580h+node-ID	COB-ID d'envoi du 1er SDO serveur
	3	node ID of the SDO client	8 bits non signé	rw	none	Node ID du SDO client
1280	0	Number of elements	8 bits non signé	ro	2	paramètre du 1er SDO client
	1	SDO receive COB-Id	32 bits non signé	ro	aucune	COB-ID de récéption du 1er SDO client
	2	SDO transmit COB-ID	32 bits non signé	ro	aucune	COB-ID d'envoi du 1er SDO client

	3	node ID of the SDO server	8 bits non signé	rw	none	Node ID du SDO serveur
1400	0	Number of elements	8 bits non signé	ro	2	paramètre de réception du 1er PDO
	1	COB-ID	32 bits non signé	rw	aucune	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de la réception
…						
1407						paramètre de réception du 8ème PDO
1800	0	Number of elements	8 bits non signé	ro	2	paramètre d'émission du 1er PDO
	1	COB-ID	32 bits non signé	rw	aucune	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de l'émission 252->sur synchro 253->remote(RTR) 254->périodique 255->sur modification
	3	Inhinit time	16 bits non signé	rw	254	durée d'inhibition (ms)
…						
1807						paramètre d'émission du 8ème PDO

The dictionary contains the various parameters and variables of the drive.

(see **Help \ Modsbus-CANopen** windows).

10-2-3- Instructions list

A) List of CANopen instructions

a) Exchange instructions between IMD drives

VF	Read or write a remote variable (byte)
VB	Read or write a remote variable (byte)
VI	Read or write a remote variable (word)
VL	Read or write a remote variable (double word)
VR	Read or write a remote variable (real)

b) Dictionary read or write

CANOPENB	Read or write a remote parameter (byte)
CANOPENI	Read or write a remote parameter (word)
CANOPENL	Read or write a remote parameter (double word)

c) SDO Instructions

SDOB	Read or write a remote variable (byte)
SDOI	Read or write a remote variable (word)
SDOL	Read or write a remote variable (double word)
SDOBX	Read or write a remote variable (byte)
SDOI X	Read or write a remote variable (word)
SDOLX	Read or write a remote variable (double word)

d) PDO Instructions

CANSENDNMT	Send a NMT on CAN bus
CANSENDSYNCHRO	Send 1 synchronisation message on CAN bus
CANSETUPSYNCHRO	Set up the CAN synchronization
PDOEVENT	Test a PDO arrival
PDOTX	Send mapping data

e) Generic CAN instructions

CAN	Read and write a message
CANERR	Fault detection
CANERRCOUNT	Controls and erases the communication errors
CANEVENT	Test a message arrival

CANTX	Send a message
SETUPCAN	Configure of a message

f) Multi axis instructions

CANPOSSTATUS	Return CAN position reception status
CANPOSTIMEOUTRAZ	Remove TIMEOUT error of CANPOSSTATUS function
STARTCANRECEIVEPOSITION	Start to receive drive position by CANopen
STARTCANSENDPOSITION	Start to send position on CANopen
STOPCANRECEIVEPOSITION	Stop reception of drive position on CANopen
STOPCANSENDPOSITION	Stop sending position on CANopen

B) CAN - Read and write a message

Syntax 1: CAN (<ByteNumber>) = <Variable>

Syntax 2: <Variable> = CAN (<ByteNumber>)

Accepted types :< Variable>: Characters string

Description: This function reads or sends a message.

Remark: You have to tell the parameters of the reception COBID to receive the message.

C) CANERRCOUNTER – Controls and erases the communication errors

Syntax 1: <Variable> = CANERRORCOUNTER

Syntax 2: CANERRORCOUNTER = 0

Limits : <Variable>: from 0000h to FFFFh

Accepted types :< Variable>: integer

Description: Syntax 1 gives the number of errors that have occurred since the counter was reset. Syntax 2 resets the errors counter.

D) CANERR – Error detection

Syntax: <Variable> = CANERR

Accepted types :< Variable>: Byte

Bit 0 to 1 if bus error

Bit 1 to 1 if SDO timeout

Bit 2 to 1 if Node Guarding error

Description: This function shows if an error has occurred.

E) CANEVENT – Test a message arrival

Syntax : <Variable> = CANEVENT

Accepted types :<Variable> : Boolean

Description : This function shows if a message has been received.

Remark : You have to set the parameters of the reception COBID to receive the message.

F) CANOPENX - Read or write a remote parameter

Syntax 1 : CANOPENB (<Index>, <Sub-Index>) = <byte or variable>

Syntax 2 : <Variable> = CANOPENB (<Index>, <Sub-Index>)

Syntax 3 : CANOPENI (<Index>, <Sub-Index>) = <word or variable>

Syntax 4 : <Variable> = CANOPENI (<Index>, <Sub-Index>)

Syntax 5 : CANOPENL (<Index>, <Sub-Index>) = <double word or variable>

Syntax 6 : <Variable> = CANOPENL (<Index>, <Sub-Index>)

Limits : <Index> : : from 0000h to FFFFh

<Sub-index> : from 00h to FFh

Syntax 1 and 2 : <Variable> : from 00h to FFh

Syntax 3 and 4 : <Variable> : from 0000h to FFFFh

Syntax 5 and 6 : <Variable> +/- 7FFFFFFFh

Description: This function reads or writes a remote parameter in the dictionary of the IMD drive.

G) CANPOSSTATUS - Receive status of the CAN position

Syntax : CANPOSSTATUS

Description : This instruction returns the receive status of the CAN position

- 0 : no reception
- 1 : reception in progress
- 2 : the reception has been interrupted for more than <TimeOut> but is running now.
- 3 : the reception is stopped because a master position error has been detected.

H) CANPOSTIMEOUTRAZ - Remove TIMEOUT error of CANPOSSTATUS function

Syntax : CANPOSTIMEOUTRAZ

Description : This instruction removes <TimeOut> error of the CANPOSSTATUS function.

I) CANSENDNMT - Send an NMT on CAN bus

Syntax : CANSENDNMT (<Node>, <Action>)

Description : This instruction sends an NMT command to <Node> devices for starting PDO.

Acepted values :<Node> 0 to 31

- 0 : send NMT to all devices
- local drive : send to itself
- other : send to <Node> device

<Action>

- 1 : send START
- 2 : send STOP
- 3 : send DTSCONNECT

J) CANSENDSYNCHRO - Send a synchronization message on the CAN bus

Syntax : CANSENDSYNCHRO (<COBID>)

Description : This instruction sends a synchronization message.

Accepted values:<COBID> between 0x80 and 0xFF (0x80 by default)

K) CANSETUPSYNCHRO – Set up CAN synchronization for PDO messages

Syntax : CANSETUPSYNCHRO (<COBID>, <Period>)

Description : This instruction sets up the synchronization of the PDO messages.

Accepted values :<COBID> between 0x80 and 0xFF (0x80 by default)

< Period> number of 150µs intervals between 2 PDO messages.

Warning : If <Period> = 0 then the synchronisation is stopped.

L) CANTX - Send a message

Syntax: CANTX

Description: This function send the CAN message.

M) PDOEVENT – Test a PDO arrival

Syntax: <Variable> = PDOEVENT (<NumPDO>)

Limits : <NumPDO> : from 01h to 08h

Accepted types :< Variable>, <NumPDO> : Byte

Description: This function indicates if the request for a PDO is effective.

Remark : You have to set the transmission parameters of the PDO to receive a PDO.

N) PDOTX - Send mapping data

Syntax : PDOTX

Description: This function sends mapping data.

O) SDOB, SDOI, SDOL - Read or write a remote variable

Syntax 1 : SDOB (<Index>, <Sub-Index>) = <byte or variable>

Syntax 2 : <Variable> = SDOB (<Index>, <Sub-Index>)

Syntax 3 : SDOI (<Index>, <Sub-Index>) = <word or variable>

Syntax 4 : <Variable> = SDOI (<Index>, <Sub-Index>)

Syntax 5 : SDOL (<Index>, <Sub-Index>) = <double word or variable>

Syntax 6 : <Variable> = SDOL (<Index>, <Sub-Index>)

Limits : <Index> : : from 0000h to FFFFh

<Sub-index> : from 00h to FFh

Syntax 1 and 2 : <Variable> : from 00h to FFh

Syntax 3 and 4 : <Variable> : from 0000h to FFFFh

Syntax 5 and 6 : <Variable> +/- 7FFFFFFFh

Description: This function reads or writes a remote variable in the dictionary of the IMD drive.

P) SDOBX, SDOIX, SDOLX - Read or write a remote variable

Syntax 1 : SDOBX (<Index>, <Sub-Index>, <Drive>) = <byte or variable>

Syntax 2 : <Variable> = SDOBX (<Index>, <Sub-Index>, <Drive>)

Syntax 3 : SDOIX (<Index>, <Sub-Index>, <Drive>) = <word or variable>

Syntax 4 : <Variable> = SDOIX (<Index>, <Sub-Index>, <Drive>)

Syntax 5 : SDOLX (<Index>, <Sub-Index>, <Drive>) = <double word or variable>

Syntax 6 : <Variable> = SDOLX (<Index>, <Sub-Index>, <Drive>)

Limits : <Index> : : from 0000h to FFFFh

<Sub-index> : from 00h to FFh

Syntax 1 and 2 : <Variable> : from 00h to FFh

Syntax 3 and 4 : <Variable> : from 0000h to FFFFh

Syntax 5 and 6 : <Variable> +/- 7FFFFFFFh

Description: This function reads or writes a remote variable in the dictionary of the IMD drive.

Q) SETUPCAN – Configure a message

Syntax : SETUPCAN (<TX COBID>, <RX COBID>)

Accepted types :<TX COBID>, <RX COBID> : Long integer

Description : This function configures the reception and transmission COBID before sending a message.

R) STARTCANRECEIVEPOSITION - Start to receive drive positions by CANopen bus

Syntax : STARTCANRECEIVEPOSITION (<PDO>, <COBID>, <Offset>, <TimeOut>)

Description : This instruction starts to receive drive positions by CANopen bus.

Accepted values :< PDO> PDO number : 1 to 8

<COBID> between 0x181 to 0x37F

<Offset> allows compensating transmission delay, between 0 and position send period.

- <Offset> = 0: good accuracy but timing offset equals position send period.
- <Offset> = <Period> + 1: small timing offset or null but low accuracy.

<TimeOut> number of 150µs intervals before CANPOSSTATUS error.

Warning: The instruction PDO can’t be use by other CAN instructions.

SAVEPARAM and SAVEVARIABLE functions cancel CAN position sending.

S) STARTCANSENDPOSITION - Start to send positions on CANopen bus

Syntax : STARTCANSENDPOSITION (<Source>, <PDO>, <COBID>, <Period>)

Description : This instruction starts to send positions on the CANopen bus.

Accepted values :<Source> 0 for slave axis and 1 for master axis.

PDO> PDO number : 1 to 8

<COBID> between 0x181 and 0x37F

<Period> number of 150µs intervals between 2 PDO messages.

Warning: If <Period> = 0 then position is sent as soon as possible.

The instruction PDO can't be use by other CAN instructions.

SAVEPARAM and SAVEVARIABLE functions cancel CAN position sending.

T) STOPCANRECEIVEPOSITION - Stop receiving drive positions by CANopen bus

Syntax : STOPCANRECEIVEPOSITION (<PDO>)

Description : This instruction stops receiving positions by the CANopen bus.

Accepted values :< PDO> PDO number: 1 to 8

U) STOPCANSENDPOSITION - Stop sending positions on CANopen bus

Syntax : STOPCANSENDPOSITION (<PDO>)

Description : This instruction stops sending positions on the CANopen bus.

Accepted values :< PDO> PDO number : 1 to 8

V) VB, VI and VL - Read or write a remote variable

Syntax 1 : VB (<Index>, <Sub-Index>) = <byte or variable>

Syntax 2 : <Variable> = VB (<Index>, <Sub-Index>)

Syntax 3 : VI (<Index>, <Sub-Index>) = <word or variable>

Syntax 4 : <Variable> = VI (<Index>, <Sub-Index>)

Syntax 5 : VL (<Index>, <Sub-Index>) = <double word or variable>

Syntax 6 : <Variable> = VL (<Index>, <Sub-Index>)

Limits : <Index> : : from 0000h to FFFFh

<Sub-index> : from 00h to FFh

Syntax 1 and 2 : <Variable> : from 00h to FFh

Syntax 3 and 4 : <Variable> : from 0000h to FFFFh

Syntax 5 and 6 : <Variable> +/- 7FFFFFFFh

Description: This function reads or writes a remote variable of an IMD drive.

10-2-4- Examples

A) Exchange variables between IMD drives

a) Changing another drive's variable:

```
VR(2,3)=VR1            'send value of VR1
                        'to drive n° 3, in VR2
```

b) Reading another drive's variable list :

```
VB1=0
REPEAT                              'Read from drive n°5
        VR0[VB1]= VR(VB1,5)         ' variables from VR0 to VR9
        VB1=VB1+1
UNTIL VB1=10
```

B) Communication by SDO

a) Read inputs states from IMD drive no. 3

```
CANopenL(1280h,1)=603h 'Initialization of the ClientSDO TX
CANopenL(1280h,2)=583h 'Initialization of the ClientSDO RX
Loop:
        DELAY 10
        Inputs = SDOI(60FDh,0) 'Read Drive 3 inputs state
GOTO Loop
```

b) Write outputs to IMD drive no. 5

```
CANopenL(1280h,1)=605h 'Initialization of the ClientSDO TX
CANopenL(1280h,2)=585h 'Initialization of the ClientSDO RX
SDOI(60FEh,0) = 0 'Write the outputs of drive no. 5 via SDO
OldOutputs = 0
Loop:
        IF OldOutputs <> Outputs THEN
                SDOI(60FEh,0) = Outputs
                OldOutputs = Outputs
        END IF
```

```
GOTO Loop
```

C) Communication by PDO

a) Drive no. 1

```
Prog
    Delay 2000
    ‘ Transmit PDO no. 4
    CANopenL(1803h,01h)=00000481h       ‘COBID  number
    CANopenB(1803h,02h)=0FFh            ‘Transmission type  : cyclic
    CANopenB(1A03h,00h)=01h             ‘Number of mapped PDO
    CANopenL(1A03h,01h)=33000020h       ‘PDO mapping
    ‘ Receive PDO no. 4
    CANopenL(1403h,01h)=00000482h       ‘COBID  number
    CANopenB(1403h,02h)=0FFh            ‘Transmission type  : cyclic
    CANopenB(1603h,00h)=01h             ‘Number of mapped PDO
    CANopenL(1603h,01h)=33000A20h       ‘PDO mapping

    SetupCan(0,1)       ‘NMT
    Can(0)=2
    Can(1)=1
    Can(2)=0
    CanTx

   loop:
        Delay 10
        VL0=VL0+1
    goto loop

EndProg
```

b) Drive no. 2

```
Prog
    Delay 2000
    ‘ Transmit PDO no. 4
```

```
    CANopenL(1803h,01h)=00000482h      'COBID  number
    CANopenB(1803h,02h)=0FFh           'Transmission type  : cyclic
    CANopenB(1A03h,00h)=01h            'Number of mapped PDO
    CANopenL(1A03h,01h)=33000B20h      'PDO mapping
    ' Receive PDO no. 4
    CANopenL(1403h,01h)=00000481h      'COBID  number
    CANopenB(1403h,02h)=0FFh           'Transmission type  : cyclic
    CANopenB(1603h,00h)=01h            'Number of mapped PDO
    CANopenL(1603h,01h)=33000120h      'PDO mapping

    SetupCan(0,1) 'NMT
    Can(0)=2
    Can(1)=1
    Can(2)=0
    CanTx

    loop:
        Delay 10
        VL11=VL11+1
    goto loop

EndProg
```

D) Generic CAN example

```
 SetupCan(1,1)
    Can(0)=2
    Can(1)=1
    Can(2)=0
    CanTx
    VI5=CanErrCounter
    VB5=CanErr
    if CanEvent=0 Goto St
        VB0=Can(0)
```

```
    VB1=Can(1)
    VB2=Can(2)
    VB3=VB3+1
St:
if VF10=0 goto st2
    CanErrCounter=0
    CanErr=0
St2:
```

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.

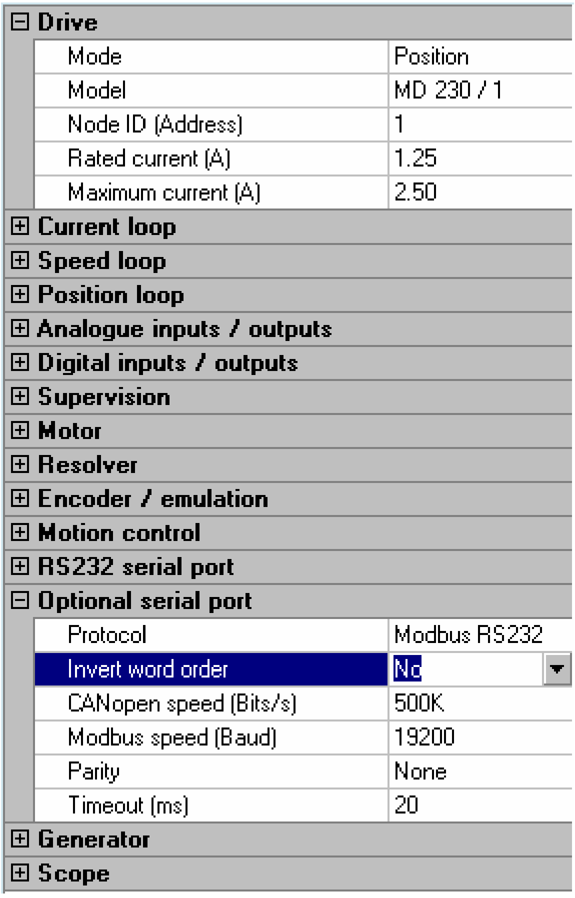
Functions for reading words (function no. 3 or 4) and writing words (function no.16) are recognized by the drive.

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.



System communication	Invert parameter	Data format	VR & VL coding version	Parameter coding version
Enable	X	Float	No	No
Disable	No	Float or Decimal	No	No
Disable	Yes	Float or Decimal	Yes	Yes

* 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 **Help \ Modsbus-CANopen** windows).

- Parameters are accessible between addresses 1000 and 4000
- Flag variables are accessible between addresses E000h and E00Fh
- Byte variables are accessible between addresses E010h and E08Fh
- Integer variables are accessible between addresses E090h and E18Fh
- Long-integer variables are accessible between addresses E190h and E38Fh
- Real variables are accessible between addresses E390h and E58Fh

Difference between IMD and MD table:

Adress	MD	IMD	Comment
0x0000	Reserved	Reserved	
0x0258	Parameter	Reserved	
0x03E8	Reserved	Parameter	
0x2000	Reserved	FRAM	4k word/direct access
0x3000	Reserved	Reserved	
0x8000	Exchange PC	Exchange PC	Reserved
0xEFFF	Variables	Variables	see modbus dictionnary
0xFFFF			

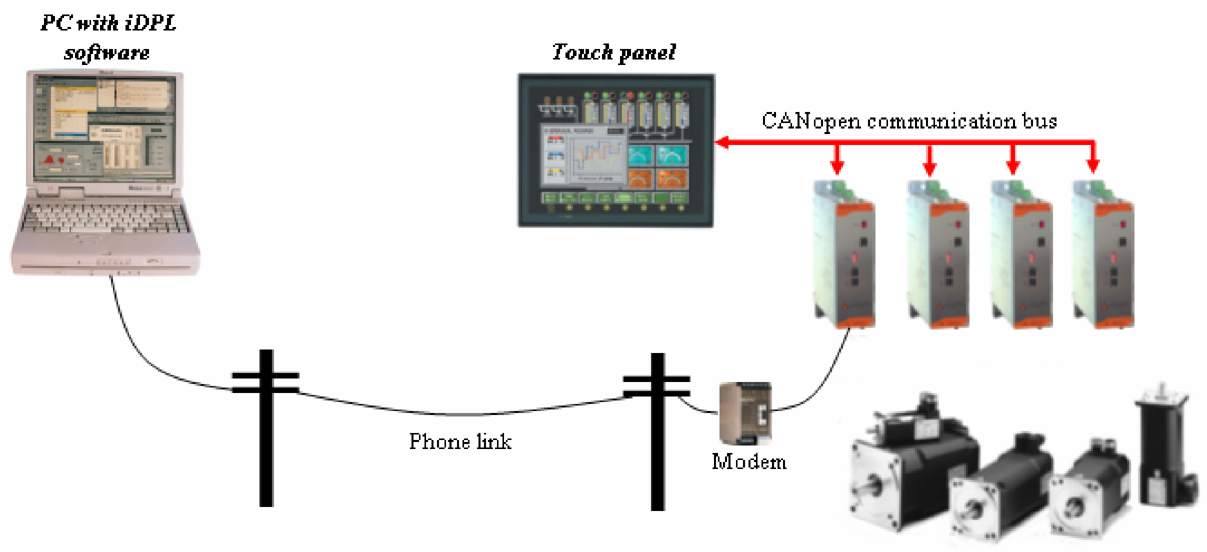
11- Remote control

11-1- Connections

The remote control allows by a simple phone link to remotely control one or several IMD drive with iDPL software. The remote control is composed of an integrated dialler and two modems linked by a phone link.

11-1-1- Structure

The different parts are linked as shown:



11-1-2- RS 232 link between the modem 1 and the MCS 32 EX

9 points SUBD pin assignment :

Pin	IMD	Modem
1		CD
2	RXD	RXD
3	TXD	TXD
4		DTR
5	GND	GND
6		DSR
7		RTS
8	CTS	CTS

9		

Use a shielded cable with shield connected at each end.

Linking :

11-1-3- RS 232 link between the modem 2 and the PC

This link between the modem and the PC is made with the cable provided with the modem.

11-2- Link establishment

11-2-1- Setting up the modem 1 connected to the IMD drive

The set-up of the modem connected to the IMD drive is made by connecting this modem to a PC. A terminal software is used to send commands to the modem.

This set-up have to following objectives :

- Initialising the modem
- Defining the number of ringing before the modem pick up to allow an automatic establishment of the link.
- Removing all hardware and software flow controls.
- Storing this configuration into the non-volatile memory of the modem.
- Selecting these parameters in the non-volatile memory as parameter to be used at power on.

Example :

Parameters for an « 3Com Us Robotics Sportster » modem type :

- Command : AT&F0

  Meaning : Using default factory settings.

- Command : ATS0=3

  Meaning : Automatic pick up after 3 ringing.

- Command : AT&H0

  Meaning : Disable the flow control when sending

- Command : AT&I0

  Meaning : Disable the flow control when receiving

- Command : AT&W0

  Meaning : Store current parameters into the non-volatile memory bank #0

- Command : ATY0

  Meaning : Selecting these parameters in the non-volatile memory as parameter to be used at power on.

When the modem take these commands into account it answers « OK » .

Parameters for an « Wertermo TD31 or TD32 » modem type :

- Command : AT&F

  Meaning : Using default factory settings.

- Command : ATS0=3

  Meaning : Automatic pick up after 3 ringing.

- Command : AT&C1

  Meaning : Activate DCD when connected

- Command : AT&K0

  Meaning : Disable the flow control

- Command : AT&W0

  Meaning : Store current parameters into the non-volatile memory bank #0

- Command : AT&Y0

  Meaning : Selecting these parameters in the non-volatile memory as parameter to be used at power on.

When the modem take these commands into account it answers « OK » .

11-2-2- Setting up the modem 2 connected to the PC

The setting up of the modem connected to the PC is done by modifying the information in the « Modem » part of the DPL.INI file that is in the iDPL\Data directory.

This set-up have to following objectives :

- Initialising the modem
- Remove handling of the DSR and DTR signals to avoid automatic hang-up when the communication port is closed.
- Defining the way the calls are made and how to hang-up the line.
- Defining the messages sent by the modem.
- Parameters are setup for standard modem.

Example :

Parameters for an « 3Com Us Robotics Sportster » modem type :

- Parameter : Init1

  Value : ATZ

  Meaning : Using default factory settings.

- Parameter : Init1TimeOut

  Value : 5

  Meaning : Maximal waiting delay in 1/10 before the modem answer.

- Parameter : Init2

  Value : AT&D0&S0

  Meaning : Remove the DTR and DSR handling

- Parameter : Init2TimeOut

  Value : 5

  Meaning : Maximal waiting delay in 1/10 before the modem answer.

- Parameter : Dial

  Value : ATDT for vocal dial. ATDP for a pulse dial

  Meaning : Selecting the way to call.

- Parameter : DialTimeOut

  Value : 600

  Meaning : Maximal waiting delay in 1/10 before the modem connection.

- Parameter : Ok

  Value : OK

  Meaning : Modem answer if the command have been handled correctly.

- Parameter : Connect

  Value : CONNECT

  Meaning : Modem answer when connecting.

- Parameter : Busy

  Value : BUSY

  Meaning : Modem answer if the line is busy.

- Parameter : Hangup

  Value : ATH

  Meaning : Selecting the way to hang-up.

- Parameter : HangupOk

  Value : NO CARRIER

  Meaning : Modem answer when hanging-up

- Parameter : CommandTimeOut

  Value : 20

  Meaning : Maximal waiting delay in 1/10 before the modem going to the command mode.

- Parameter : HangupTimeOut

  Value : 20

  Meaning : Maximal waiting delay in 1/10 before the hanging up.

All missing parameter is automatically set to the default values indicated on the first using.

Parameters for an « Westermo TD31 or TD32 » modem type :

- Parameter : Init1

  Value : ATZ

  Meaning : Using default factory settings.

- Parameter : Init1TimeOut

  Value : 20

  Meaning : Maximal waiting delay in 1/10 before the modem answer.

- Parameter : Init2

  Value : AT&F&K0

  Meaning : Remove the DTR and DSR handling

- Parameter : Init2TimeOut

  Value : 20

  Meaning : Maximal waiting delay in 1/10 before the modem answer.

- Parameter : Dial

  Value : ATDT for vocal dial. ATDP for a pulse dial

  Meaning : Selecting the way to call.

- Parameter : DialTimeOut

  Value : 600

  Meaning : Maximal waiting delay in 1/10 before the modem connection.

- Parameter : Ok

  Value : OK

  Meaning : Modem answer if the command have been handled correctly.

- Parameter : Connect

  Value : CONNECT

  Meaning : Modem answer when connecting.

- Parameter : Busy

  Value : BUSY

  Meaning : Modem answer if the line is busy.

- Parameter : Hangup

Value : ATH

Meaning : Selecting the way to hang-up.

- Parameter : HangupOk

  Value : NO CARRIER

  Meaning : Modem answer when hanging-up

- Parameter : CommandTimeOut

  Value : 20

  Meaning : Maximal waiting delay in 1/10 before the modem going to the command mode.

- Parameter : HangupTimeOut

  Value : 20

  Meaning : Maximal waiting delay in 1/10 before the hanging up.

The dialler expect that the modem is setup to send an echo for all sent command and to receive a text message as answer. If not the communication is unable. It's possible to be sure to start with a good set-up for the modem by using the factory settings as default parameters.

A terminal software is used to send commands to the modem.

Parameters for an « 3Com Us Robotics Sportster » modem type :

- Command : AT&F

  Meaning : Using default factory settings.

- Command : AT&W0

  Meaning : Store current parameters into the non-volatile memory bank #0

- Command : ATY0

  Meaning : Selecting these parameters in the non-volatile memory as parameter to be used at power on.

Parameters for an « Wertermo TD31 or TD32 » modem type :

- Command : AT&F

  Meaning : Using default factory settings.

- Command : AT&W0

  Meaning : Store current parameters into the non-volatile memory bank #0

- Command : AT&Y0

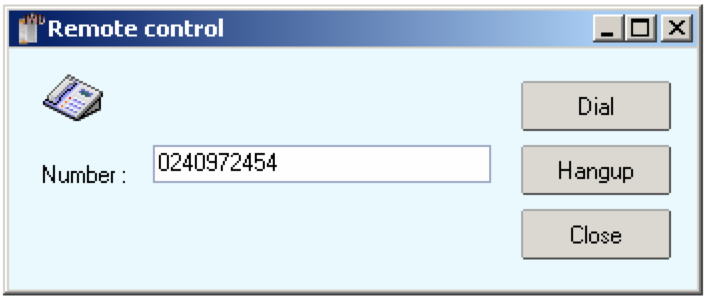
  Meaning : Selecting these parameters in the non-volatile memory as parameter to be used at power on.

ATTENTION :

- For Westermo modem , it's also recommended to let the Dips configuration as default (all OFF).

11-2-3- Call :

By using the phone dialler integrated in the iDPL software, we can establish and interrupt the phone link. The phone dialler is accessible form the Communication menu / Remote control.



After entering the phone number, click on «Dial» button to establish the link. The «Hang up » button allows to interrupt the link.

When the link is established, we can use all the MCB functions including :

- Send and receive the configuration, variables, tasks, CAM, FRAM memory ...
- Start and stop the tasks.
- Access to debug tools : Hyper-terminal, Scope, Trace, Manual mode.
- Reload OS
- Access to all network drives.

11-3- List of the validated modems

- 3 Com / US Robotics :

  - Sportster Voice 33600 Fax Modem

  - Sportster 56 K Fax Modem

- Westermo :

  - TD 31

  - TD 32

Index

A

B

C

D

M

N

O

P

R

S

T

U

V