



InterpCNC V3 Manuals

Contents

PLC_Basic_Interpreter	5
Introduction	6
General	7
Error processing	7
Quick PLC Basic manual	8
FUNCTIONS	9
PLC Basic functions specific to InterpCNC V3	10
Accessing User Bits and Registers (Read only)	11
Accessing System Bits and Registers (Read only)	11
Accessing Input Registers (Read Only)	12
Mathematic functions	13
Axis motion functions	13
Timers functions	14
Input and Output functions	14
Edge detection functions for Inputs or User Bits (coils)	15
Character strings manipulation functions	15
COMMANDS	18
Commands from standard Basic	18
Program management functions	18
PLC Basic commands specific to InterpCNC V3	18
Write Access to User Registers and User Bits (Holding Registers and Coils) .	19
Commands specific to InterpCNC V3 hardware	20
Recipe management commands	21
Axis motion commands	21
Timers commands	22
Input and Output commands	23
IMPLEMENTING A GRAFCET (SFC) USING « Select Case» instruction	24
USING INTERRUPTS	25
USING THE FAST DIGITAL INPUTS (from #16 to 22)	27
COMMUNICATION WITH MODBUS DRIVES OR OTHER DEVICES	28
ETHERNET COMMUNICATION	29
USING DMX COMMUNICATION	31
USING REAL TIME CLOCK (RTC)	34
ICNCStudio	36
User_Registers	37
Saved_registers	38
Axes_window	39
Monitor	39
Digital_inputs	40
Digital_outputs	41
Coils	41
Text_editor	42
Find_Replace	46
Main	47
Parameters_table	48
Recipes	50
PLC_variables	52

Custom_Data	53
Charts	55
Analog_Counters	56
Firmware_update	56
ICNCStudio_update	57
Encryption	57
Parameters	59
General_configuration	60
Axes	61
DIN	61
AIN	64
IN_ENA	64
Serial_port	66
Polling	67
Ethernet	67
Digital_control	68
Plasma_thc	68
Release_note	70
InterpCNC V3 manual	72
Presentation	73
Setup	76
Wiring	77
MODBUS_documentation	79
Introduction	80
Identification of InterpCNC PLCs connected to the Ethernet network	81
Identification of InterpCNC PLCs connected via USB	81
Read/Write Parameters	82
Adresses of Input Bits (Read only)	82
Adresses of Input registers (Read only)	84
Adresses of Holding registers (Read/write)	89
About sending Modbus commands	90
Commande100 : Stop an axis	90
Command 101 : Stop one or several axes	91
Command 102: Move an Axis at a given Speed	91
Command 103: Move an Axis to Target Position	91
Command 104: Move an axis specified number of steps from current position	92
Command 105: Write Current Position Counter (same as writing to position registers)	92
Command 106: Launch homing of an axis	92
Command 107: Launch a Probe on an input	94
Command 108: Start Probing on Multiple Inputs	94
Command 109: Launch of a Probing on analog input threshold	95
Command 110: Force Inputs	97
Command 200 : PLCBasic command	98
Using indexed Reads/Writes	98
CNC control functions	99
Part 1: Buffered commands	100
Command 1000 : Executing a Gcode instruction	100
Command 1001 : Definition of the machining speed in mm/min	101
Command 1002: Interpolated Linear Move of Axes to Target Positions	

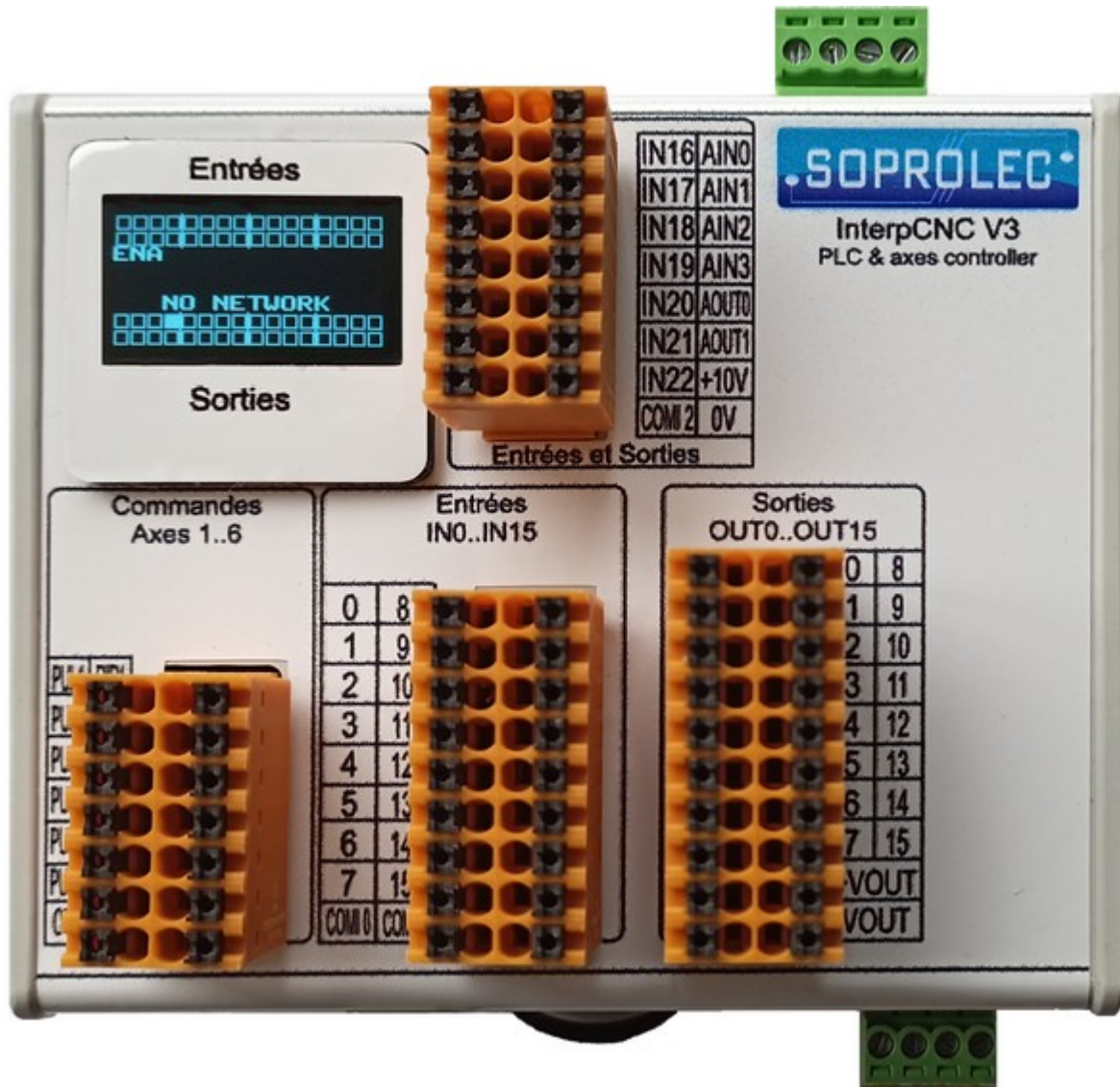
(Absolute Positions)	101
Command 1003 : Circular interpolation	102
Command 1010: Synchronized Action (Digital Output, Analog Output, Register)	102
Command 1011: Buffered Timeout	104
Command 1012: Wait for State or Event	104
Part 2: Unbuffered Commands	106
Command 1100: Edit Override Machining and Rapid Traverse	106
Command 1101: Pause machining in progress	106
Command 1102: Resume interrupted machining	106
Command 1110: Execute a homing machine sequence	107
Command 1111: Execution of a manual move (Jog)	107
Command 1200: Direct execution of a command	108
Using the internal clock (RTC)	108
Command 112: Set Date on RTC Clock	109
Command 113: Set Time to RTC Clock	109
Command 114: Simultaneous setting of date and time on the RTC clock	110
Miscellaneous settings	110
Password protection	110
Standard inputs	111
Fast inputs	112
Using Analog inputs as Digital inputs	113
I/O expansion modules	114
Exports and Imports of files	115

PLC_Basic_Interpreter

SOPROLEC
ZAC DE L'EPINE
72460 SAVIGNE L'EVEQUE
FRANCE
Tél : +33 (0)2 4376 4476



InterpCNC V3
SOPROLEC
Axes controller card



Integrated PLC Basic interpreter

Introduction

The InterpCNC V3 has a powerful built-in PLC Basic language interpreter. This interpreter makes it possible to develop standalone automation applications or in association with a man/machine interface (HMI).

The InterpCNC V3 communicates via an Ethernet connection (Modbus TCP or Modbus UDP protocol), or Serial (RS485, Modbus RTU protocol), or USB (virtual Com port, Modbus RTU protocol).

This interpreter works in parallel with the other functions of the card.

It is therefore possible to use the InterpCNC in digital control applications controlled by a PC while executing the Basic program for the processing of particular actions (for example, remote console for manual control of the machine).

Programming is done using the ICNCStudio software.

General

From a hardware point of view, the InterpCNC is a PLC card whose main characteristics are:

- 6-axes control (Pulses / direction)
- 16 NPN and PNP compatible inputs
- 16 PNP outputs 500mA
- 4 analog inputs 0 to 10V
- 2 analog outputs 0 to 10V
- 7 quick entries
- Connectivity: Ethernet, USB, and 2 x RS485
- 1 OLED screen 0.96"
- 32-bit microcontroller
- Supply voltage: 24V

It has the advantage of integrating a PLC Basic interpreter, which makes it, as with all SOPROLEC cards, one of the most user friendly in terms of programming. This interpreter works with variables which can be character strings or real numbers. All numeric values are real type coded on 32 bits single precision. The maximum value is 17549435e-38 and the minimum value is 3.40282347e+38. Integers that can be manipulated without loss of precision must be within ± 16777100 .

Error processing

Being an interpreted language, errors in the program are detected during operation. It is therefore important to be able to set up an error handler to interrupt any movements when an error occurs.

The PLC Basic interpreter will automatically perform a **GoTo** at the **OnError: label** if a runtime error occurs.

If this label is not present in the program, no special error processing will be performed.

In the following example, the main program is placed in a loop

```
DO...LOOP.
```

If a processing error occurs, there will be an automatic GOTO at the **OnError: label**

In the processing of the error, one stops all the possible displacements in progress and one deactivates all the exits. The program is then interrupted.

It is of course possible to add a GOTO type jump to the error processing to return to the start of the program or resume execution at a particular label.

```
' Main program
```

```
Do
```

```
' Application code
```

```
Loop
```

```
' Error handling
```

```
OnError:
```

```
StopAxes &H3F ' Stop all axes (binary weight in hexadecimal)
```

```
OUTPort 0, 0 'Set outputs 0 to 7 to 0
```

```
OUTPort 1, 0 'Set outputs 8 to 15 to 0
```

You can therefore implement such a handler in your PLC Basic program using the reserved

label **OnError**.

Similarly, the **OnStopPLC**: label defines the code to be executed if the program stops. It can be brought to stop either by a **STOP** command provided in its execution, or by a voluntary action (click on the green light in ICNCStudio).

Example:

```

OnStopPLC:
' Stop interrupts
SetTick 1, 0, OnTimerEtiq1
SetInputINT 4, -1, INPUT_INT_DFM_ONESHOT, OnCellObject

' Stop all Axes
SetAna OUT_ANA_SPEED, 0
SetAna OUT_ANA_COUPLE, 0
StopAxes &h3F

? "OnStopPLC"
end
    
```

We can execute the same portion of code in both cases, by combining **OnStopPLC**: and **OnError**: in a common label

OnStopPLC:

OnError:

```

...
...
...
end
    
```

The **end** instruction is only necessary if there is still code after it.

Quick PLC Basic manual

Of course, the InterpCNC V3 PLC Basic interpreter uses all the expressions, instructions, operators, commands and functions, as well as the functions for manipulating character strings, existing in all Basic languages.

Great programming classics, only a few of them will be explained in this manual.

The table below summarizes it:

<p>Literal Expressions / User Variables</p> <p>Literal expressions Strings are contained in quotes, ex : "InterpCNC". Les nombres peuvent être décimaux ou représentés par: &Hnn Hex Literal, ex : &H3C (60) &Bnn... Binary Literal, ex : &B00100011</p>	<p>String formatting</p> <p>% [flags] [width] [.prec] type flags: - Left justify 0 Use 0 as shift character (not space). + The + sign denotes positive values. space Space as a sign, unless negative.</p> <p>width: minimum number of output characters, minus cause of offset, plus cause of expansion.</p>	<p>SELECT CASE Variable CASE Value CASE ValueFrom TO ValueTo CASE Value IS < Limit CASE ELSE END SELECT</p> <p>Strings / Characters ASC (str\$) CHR\$ (nbr) FORMAT\$ (nbr [,format\$])</p>
--	---	--

<p>(35) n.nE+n Scientific, e.g. 1.6E+4 (16000)</p> <p>User Variables Variable names begin with an alphanumeric character or an underscore and can contain any alpha or numeric character, period (.) and underscore (_); the maximum length is 32 characters. Character string names are terminated by the null character. Numeric variable names are not terminated with the \$ symbol.</p> <p>Operators</p> <p>Arithmetic ^ * / Exponentiation, Multiplication, Division MOD \ Modulus (remainder), Integer Division + + - Addition, String concatenation, Substraction</p> <p>Logical NOT Logic Inverse = <> Equality, Inequality > < Greater than, less than <= or =< Less than or equal to >= or => Greater than or equal to AND OR Conjunction, Disjunction XOR Exclusive Or</p>	<p>.prec: number of fraction digits for types e, or f, or the max. significant digits for type g. Must be preceded by a period (.) if used. Type: g or G format for the best presentation. f or F Decimal format with decimal point and digits e or E Exponential format</p> <p>Commands / Declarations Array declaration : DIM variable(elements...)</p> <p>Execution control CONTINUE DO <declarations> LOOP DO WHILE expression <declarations> LOOP DO <declarations> LOOP UNTIL expression ELSE ELSEIF expression THEN ENDIF END EXIT EXIT FOR FOR counter = beginning TO end [STEP increment] GOSUB GOTO IF expression THEN IRETURN NEXT [variable-counter] [, variable-counter]... RUN STOP LIST NEW</p>	<p>INSTR ([start,] search\$, pattern\$) LEFT\$ (str\$, nbr) LEN (str\$) LCASE\$ (str\$) MID\$ (str\$, start [,nbr]) RIGHT\$ (str\$, nbr) SPACE\$ (nbr) SPC (nbr) STRING\$(nbr, val str\$) TAB(nbr) UCASE\$ (str\$) VAL (str\$) INKEY\$</p> <p>Functions</p> <p>Maths / Nombres ABS (nbr) ATN (nbr) CINT (nbr) COS (nbr) EXP (nbr) FIX (nbr) HEX\$ (nbr) INT (nbr) LOG (nbr) OCT\$ (nbr) RND (nbr) SGN (nbr) SIN (nbr) SQR (nbr) STR\$ (nbr)</p>
--	--	---

FUNCTIONS

A **function** differs from a command in that it can return a Value, a result. To use it, just like for a command, we pass arguments to it as parameters, the number of which is as many as necessary.

As in most programming languages, you can create your own functions yourself, using **function** and **end function** statements.

To do so, the syntax is as follows:

function **FunctionName**(Param1,Param2,Param3,...)

...
... *' here lines of code using the Values contained in the variables passed as parameters*

...
if ... then Return=1

else Return=0

FunctionName=Return ' *the return of a Value is done by assigning the variable bearing the name of the function*
end function

To return a result we use the name of the function as a global variable, which we will assign to the calculated result or the Value that we wish to return.

PLC Basic functions specific to InterpCNC V3

Functions

GetMB(memo)

GetMW(adr)

GetMDW(adr)

GetMI(adr)

GetMDI(adr)

GetME(adr)

StsBit(BitNbr)

GetInputMB(BitNbr)

GetInputMW(adr)

GetPrm(nbr)

IsEEdataChanged

IsRPCChanged

IsMBPrmChanged

Min(nbr, nbr)

Max(nbr, nbr)

Limit(nbr, Min, Max)

GetPos(nbr)

GetCapturePos(nbr)

GetTimer (str)

Toc(nbr)

Cyclestat(nbr)

IN(nbr)

Ain(nbr)

AinV(nbr)

Out(nbr)

GetEncoder(nbr)

GetCnt(nbr)

DE(nbr)

DEM(nbr)

DED(nbr)

DEMBit(nbr, nbr)

DFDBit(nbr, nbr)

Accessing User Bits and Registers (Read only)

GetMB (Memo) OU GetMB (Register, Bit)	Reading a Memo bit (Coil) in the Memos space (user Coils), or a particular bit in the registers area. <u>Example :</u> if GetMB (<i>MBB_ONOFF_CONVEYOR</i>) then ... 'if the conveyor ON/OFF bit is at 1, then...' if GetMB (3100, 15) then... 'if the 15th bit of address 3100 (U16, here in Ram) is to 1, then...' if GetMB (3100, 31) then... 'if the 31st bit of address 3100 (U32, here in Ram) is at 1, then...'
GetMW (Register)	Reading an unsigned 16-bit register (U16) <u>Example :</u> ResolutionAxe2 = GetMW (<i>EE_RESO_AXE2</i>) /10 'Assigment in a variable, of the Value stored in the corresponding register (U16) in EEPROM'
GetMDW (Register)	Reading an unsigned 32-bit register (U32) <u>Example :</u> TotalCounter = GetMDW (<i>EE_CNT_TOTAL</i>) 'Assigment in a variable, of the Value stored in the corresponding register (U32) in EEPROM'
GetMI (Register)	Reading a signed 16-bit register (I16) <u>Example :</u> Offset = GetMI (<i>RCP_OFFSET_BOTTLE</i>) *ResOrientator/360
GetMDI (Register)	Reading a signed 32-bit register (I32) <u>Example :</u> Target = GetMDI (<i>POSITION_SPOT</i>) + Offset
GetMF (Register)	Reading a 32-bit register treated as a Float (FLOAT) <u>Example :</u> ResolConveyor = GetMF (<i>EE_RES_CONVOYEUR</i>) 'Assigment in a variable, of the Value stored in the corresponding register (FLOAT) in EEPROM'

Accessing System Bits and Registers (Read only)

StsBit (BitNo) BitNo : Status bit number from 0 to 359	Reading of the state of one of the card's register bits. <u>Example :</u> If not StsBit (256) then... 'If Axis 1 is no longer moving, then...'
GetInputMB (BitNo)	Read the state of one of the card's register bits (Input Bits, from 0 to 359, read-only). Same as StsBit . If not GetInputMB (256) then... 'If Axis 1 is no longer moving, then..'
GetInputMW (RegisterNo)	Read the status of one of the card registers (Input

	Registers, from 1000 to 1143, read-only). <u>Example :</u> FirmVerHigh = GetInputMW(1115) FirmVerLow = GetInputMW(1116) ? « Firmware Version : », FirmVerHigh, " ", FirmVerLow <i>'Shows the Firmware version of the card in the Monitor</i>
GetPrm (ParameterNumber) ParameterNumber : parameter ID from 0 to 999	Reading the Value of a card parameter by knowing its identifier. ? GetPrm (20) <i>'Displays in the monitor, the Value of parameter 20.</i>
IsEEdataChanged	Returns 1 if the contents of non-volatile user memory have been modified (NB: the state of IsEEdataChanged is immediately reset after being read/tested). <u>Example :</u> If IsEEdataChanged then CalculateParameters()
IsRCPChanged	Returns 1 if the contents of the EEPROM user memory dedicated to recipes has been modified (NB: the state of IsRCPChanged is immediately reset to zero after being read/tested). <u>Example :</u> If IsRCPChanged then ? « Recipe modified! »
IsMBPrmChanged	Returns 1 if the content of the user EEPROM storing the card parameters has been modified (NB: the state of IsMBPrmChanged is immediately reset after being read/tested). <u>Example :</u> If IsMBPrmChanged then CalculateParameters()

Accessing Input Registers (Read Only)

As a reminder, the Input Registers are located between addresses 1000 and 2512, and between addresses 11000 and 12220 for the buffers (see the Modbus documentation for the correspondence of these registers).

The same reading functions as for Holding Registers can be used, provided that 1xxxxx is added to the register address.

Examples :

GetMW (Register+100000)	Reading an unsigned 16-bit Input Register (U16) <u>Example :</u> FirmVerLow = GetMW(101115) FirmVerHigh = GetMW(101116) <i>'Assignment in a variable, of the Value stored in the corresponding register (U16)</i>
GetMDW (Register+100000)	Reading an unsigned 32-bit Input Register (U32) <u>Example :</u> NbFramesDMXRecues = GetMDW(101998) <i>'Assignment in a variable, of the Value stored in the corresponding register (U32)</i>
GetMI (Register+100000)	Reading a signed 16-bit Input Register (I16)

	<p><u>Example :</u> I16Value = GetMI(1xxxxx) 'Assignment in a variable, of the Value stored in the corresponding register (I16)</p>
GetMDI (Register+100000)	<p>Reading a signed 32-bit register (I32) <u>Example :</u> PositionAxe1 = GetMDI(101030) 'Assignment in a variable, of the Value stored in the corresponding register (I32)</p>
GetMF (Register+100000)	<p>Reading a 32-bit Input Register treated as a Float (FLOAT) <u>Example :</u> FloatValue = GetMF(1xxxxx) 'Assignment in a variable, of the Value stored in the corresponding register (FLOAT)</p>

NB : GetInputMW(1115) is equivalent to GetMW(101115) but for consistency with other commands, it is best to use GetMW(Register+100000)

Mathematic functions

Min (Value1, Value2)	<p>Returns the minimum Value between the Values (or variables) Value1 and Value2. Caution: Value 1 and Value 2 are treated as integers. Example : MinVoltage = Min(AinV(1), AinV(2)) 'MinVoltage reads the smallest Value (voltage) on 2 analog inputs</p>
Max (Value1, Value2)	<p>Returns the maximum Value between the Values (or variables) Value1 and Value2. Caution: Value 1 and Value 2 are treated as integers. Example : VoltageMax = Max(Ain(3), Ain(4)) 'VoltageMax records the highest Value (points) on the 2 analog inputs</p>
Limit (Value, Mini, Maxi)	<p>Returns Value, bounded between Min and Max. Very useful to test a variable avoiding a succession of "if... then... else..." Example : NewPosition = Limit(CurrentPosition, 10000, 30000) 'Whatever the Value of CurrentPosition, NewPosition will remain limited between 10000 and 30000</p>

Axis motion functions

<p>GetPos(AxisNumber) AxisNumber : number of the Axis, from 1 to 5</p>	<p>Reading of the position counter of an axis. The return is a signed 32-bit integer. <u>Example :</u> PositionAxis_1 = GetPos(1)</p>
--	--

Timers functions

<p>GetTimer(VariableTimer)</p>	<p>Reading a timer initialized with SetTimer. Returns the time remaining on a timer initialized with the SetTimer command. When the timer has elapsed, this function returns 0. Example : SetTimer Tempo1, 100 if GetTimer(Tempo1) then out 1,1 endif</p>
<p>Toc(instance number from 0 to 15)</p>	<p>Returns the period of time elapsed since the Tic command (in microseconds) Example : ? Toc(1)</p>
<p>Cyclestat(instance number from 0 to 15)</p>	<p>Updated time measurement instance. Used to measure a cycle time, Values to be retrieved from the 3 MW registers defined during the CyclestatInit command. The times are given in 1/10 ms Example : CycleStat(1) <i>'Refreshes stats in the 3 designated registers</i></p>
<p>Timer</p>	<p>Returns the elapsed time (in ms) since the card was powered on. Example : Print Timer/1000; "seconds elapsed"</p>
<p>Time\$</p>	<p>Returns in the form of a character string, the time elapsed since power-on. Format: hh:mn:s,ms Example : ? "Last Powered On At:", Time\$</p>

Input and Output functions

<p>IN(InputNumber)</p> <p>InputNumber : Input number 0 to 255</p>	<p>Reading the state of an input. Result = 0 or 1. Example : if not IN(8) then ? « Low air pressure » <i>'If input 8 is at 0, then print « Low air pressure »</i></p>
<p>Ain(ChannelNumber)</p> <p>ChannelNumber : Analog input number 0 to 3</p>	<p>Reading the state of an analog input in millivolts. Example : If Ain(1) > 5000 then Out 1, 1 else Out 1, 0 endif</p>
<p>AinV(CanalNumber)</p> <p>ChannelNumber : Analog input number 0 to 3</p>	<p>Reading the state of an analog input in Volts Example : If AinV(1) > 3,30 then Out 1, 1 else</p>

	Out 1, 0 endif
Out (OutputNumber) OutputNumber = 0 to 15	Reading the current state of an output. <u>Example</u> : ? Out (15) 'Displays in the monitor, the state of output 15

Edge detection functions for Inputs or User Bits (coils)

DF (number or name of the input)	Rising or falling edge detection of an input. This function is used to detect the transition from state 0 to state 1 between two calls to this function. <u>Examples</u> : if DF (2) then SetAlarme() 'If input 2 state changes, then... or if DF (IN_LOCK) then ? « Intrusion ! » SetAlarme() endif <i>'if the input detecting the lock changes state, then we execute our alarm function</i>
DFM (number or name of the input)	Rising or falling edge detection of an input. <u>Example</u> : if DFM (CELLULE_OBJET) then ? « Object detected » ...
DFD (number or name of the input)	Falling edge detection of an input. This function is used to detect the transition from state 1 to state 0 between two calls to this function. <u>Example</u> : if DFD (PRESSURE_SWITCH) then ClearAlarme()
DFMBit (instance number (1 to 64), Value of the tested bit)	Detection of a rising edge on a tested bit. <u>Example</u> : if DFMBit (2, GetMB(MBB_ALARM)) then ? « Interruption of cycles on Alarm » ...
DFDBit (Numero d'instance(1 à 64), Value of the tested bit)	Detection of a falling edge on a tested bit. <u>Example</u> : if DFDBit (3, GetMB(MBB_ALARM)) then ? « Alarms Acknowledgment »

Character strings manipulation functions

REMINDER: The **PRINT** command is one of the fundamentals of all Basic language variants.

It is used to display the content of character strings, variables, or a result returned by a function, in the console (Monitor) of the Basic interpreter.

It can be used in conjunction with the comma -> to display multiple expressions separated by a space.

Example:

```
A$="Sir"
PRINT "Hello",A$
-> Will render:
Hello Sir
```

It can also be used with the semicolon -> to display several expressions in a row without space between them. This is the **concatenation of character strings**.

Example:

```
A$="Sir"
PRINT "Hello";A$
-> Will render:
HelloSir
```

The concatenation of character strings can also be achieved using the + operator.

Example:

```
A$="12345": B$="6789"
C$ = A$+B$ 'Concatenation of both strings
PRINT C$
-> Will thus render: 123456789
```

Function	Description	Example
ASC (characters_string)	Returns the ASCII code (integer value) of (or the first) string character.	A\$ = "Hello" b = ASC(A\$) PRINT b ou PRINT ASC("Hello") → Returns H
BIN\$ (integer)	Returns the binary weight of the integer part of a number.	PRINT BIN\$(15) → Returns 1111
CHR\$ (ascii_code)	Returns the character corresponding to the ASCII code.	PRINT CHR\$(68) → Returns D
HEX\$ (integer)	Returns the Hexadecimal value corresponding to the integer part of the number.	PRINT HEX\$(65535) → Returns FFFF
INSRT (length, string1, string2)	Search all or part of a character string in another, and returns the start position.	A\$ = "Learn PLC-BASIC" B\$ = "BAS" C = INSTR(2, A\$, B\$) PRINT "The word BA starts at position "; C → Returns 11

LCASE\$(characters_string)	Returns the character string transformed into letters lowercase.	A\$="COMPUTER" PRINT LCASE\$(A\$) -> Returns computer
LEFT\$(characters_string, length)	Returns a portion of the string from the indicated length, starting from the left.	A\$ = "INTERPCNC" L\$ = LEFT\$(A\$, 6) PRINT L\$ → Returns INTERP
LEN(characters_string)	Returns the length of the character string.	A\$ = "INTERPCNC" L = LEN(A\$) PRINT L → Returns 9
MID\$(characters_string, position, length)	Returns a portion of the string from the indicated position, for the given length.	A\$ = "INTERPCNC V3" S\$ = MID\$(A\$, 7, 3) PRINT S\$ → Returns CNC
OCT\$(integer)	Returns the octal value of the integer part of a number.	PRINT OCT\$(8) → Returns 10
RIGHT\$(characters_string, length)	Returns a portion of the string from the indicated length, starting from the right.	A\$ = "INTERPCNC" L\$ = RIGHT\$(A\$, 3) PRINT L\$ → Returns CNC
SPACE\$(integer)	Returns a string with the number of spaces specified.	FOR i = 1 TO 6 A\$ = SPACE\$(i) PRINT A\$; i NEXT I
STRING\$(repetition, character)	Returns a string containing the character specified repeated n times.	PRINT STRING\$(2,68) Ou PRINT STRING\$(2,"D") → Returns DD
STR\$(real_number)	Returns the string representation of a real number of characters.	N = 453.1 PRINT STR\$(n) → Returns 453.1
TIMES\$	Returns the time elapsed since power on of the card. (NB: The assignment like: TIME\$="00:00:00" cannot modify this duration.)	PRINT "Sous tension depuis: "; TIME\$ → Returns Time in the format(example): 00:25:30
UCASE\$(characters_string)	Returns the string converted to capital letters.	A\$="computer" PRINT UCASE\$(A\$) → Returns COMPUTER
VAL(characters_string)	Returns the numeric value located at the beginning of a character string. Returns 0 (zero) if there is none.	N\$="1234B5" X=VAL(N\$) PRINT "Valeur attendue: "; X → Returns 1234

COMMANDS

A **command** differs from a function, in that it does not return any value, nor result. To use it, we pass arguments to it as parameters or settings. When using a command, only an action is expected (writing or reading bits or registers, moving axes, starting a timer, etc...)

Commands from standard Basic

Run	Starts the PLC Basic program
Stop	Stops Basic PLC Program Execution
List	Displays the listing of the PLC Basic program present in Ram, in the Monitor window
Print (or ?)	Displays the expression in inverted commas, or the value of a variable, in the Monitor window. The use of semi-colon will append several expressions in the same Print command. The use of comma will separate several expression with spaces. Examples: ? "Hello";"World" -> will show "HelloWorld" Print "Production =",Counter,"units" -> will show "Production = 33 units"

Program management functions

New	Erase the PLC Basic program in Ram
SaveProgram	Saves the program present in RAM in Flash memory
LoadProgram	Loads the program present in Flash memory to place it in RAM. This command is called automatically at power-up if the "Automatic PLC Start" parameter is active. It will then be followed by a RUN command also called automatically.

PLC Basic commands specific to InterpCNC V3

Command

[SetMB memo, nbr](#)

[SetMW adr, nbr](#)

[SetMDW adr, nbr](#)

[SetMI adr, nbr](#)

[SetMDI adr, nbr](#)

[SetME adr, nbr](#)

[IncMDW adr\[, nbr\]](#)

[CopyReg adr, nbr](#)

Unlock

Lock
ListFlash
Msg
SaveProgram
LoadProgram
SetPrm nbr, nbr

CopyRCP
InsertRCP
RemoveRCP

SetPos nbr, nbr
MoveAxe nbr, nbr, nbr, nbr
MoveSpeed nbr, nbr, nbr
Home nbr, nbr, nbr, nbr, nbr, nbr, nbr, nbr
Probe nbr, nbr, nbr, nbr, nbr, nbr, nbr
StopAxes nbr
StopAxeID nbr

Pause nbr
SetTimer str, nbr
Timer
Time\$
Tic nbr
CycleStatInit nbr, adr, adr, adr

SetIN nbr, nbr
OUT nbr, nbr
OUTPort nbr, nbr
SetAna nbr, nbr
SetAnaV nbr, nbr

SetEncoder nbr, nbr
SetCnt nbr, nbr

SetTick nbr, nbr, str
SetInputInt nbr, nbr, str
SetCaptureInt nbr, nbr, str
SetCaptureIDOnInputInt nbr, nbr

Write Access to User Registers and User Bits (Holding Registers and Coils)

<p>SetMB Memo, Value ou SetMB Register, Bit, Value</p>	<p>Set or Reset of a Memo bit (Coil) in the Memos (User Coils) area, or a particular bit in the register area. <u>Example :</u> SetMB MBB_ALARM, 1 'set an alarm bit to 1 SetMB 3100,15,0 'setting to 0 of the 16th bit of register 3100 (Ram) SetMB 3100,31,1 'set the 32nd bit of register 3100 to 1 (Ram)</p>
<p>SetMW Register, Value</p>	<p>Write to an unsigned 16-bit register (U16).</p>

	<p><u>Example :</u> SetMW GCYCLE, 10 'writes value of 10, to the address named GCYCLE</p>
SetMDW Register, Value	<p>Write to an unsigned 32-bit register (U32) <u>Example :</u> SetMDW EE_COUNTER_TOTAL, GetMDW(EE_COUNTER_TOTAL)+1 'increment by 1 of the EEPROM register storing the value of the total counter</p>
SetMI Register, Value	<p>Write to a signed 16-bit register (I16) <u>Example :</u> SetMI 3010, -32767 'writes -32767, to adress 3010 (defined here as I16, in Ram)</p>
SetMDI Register, Value	<p>Write to a signed 32-bit register (I32) <u>Example :</u> SetMI 3010, -999999 'writes value -999999, at address 3010 (here defined as I32, in Ram)</p>
SetMF Register, Value	<p>Writing a FLOAT type number to a 32-bit register (FLOAT) <u>Example :</u> SetMF 3200, 3.14159 'writes value 3.14159, to address 3200 (here defined as FLOAT, in Ram)</p>
IncMDW Register[, +/-Value of the increment =1]	<p>Used to increment a 32-bit register by the Value indicated or by +1 if the second parameter is not specified. Example 1 : IncMDW 3020, 5 ' Increment by 5 Example 2 : IncMDW 3030 ' increment by 1</p>
CopyReg Address of 1st source Register (0 to 65535), Destination Address (Holding Register from 0 to 65535), Number of registers (0 to 65535)	<p>Copies a memory area containing the specified number of registers to a destination address. <u>Example with Holding Registers :</u> CopyReg 3000, 3200, 16 'copies registers from 3000 to 3015, to registers from 3200 to 3215' <u>Example with Input Registers :</u> CopyReg 101030, 3000, 12 'copies the 6 axes position registers (Input Registers from 1030 to 1041), to addresses from 3000 to 3011 (Holding Registers)'</p>

Commands specific to InterpCNC V3 hardware

Unlock	<p>Unlocking the board. When the board is unlocked, all Axes moves or output activation commands are made possible.</p>
Lock	<p>Locking the board. When the board is locked, all Axes moves or output activation commands are inactive.</p>
ListFlash	<p>Displays the list of PLC Basic program present in Flash memory, in the monitor window</p>

SaveProgram	Saves the PLC Basic program currently in Ram, in Flash memory (same action as clicking on the SD Card icon)
LoadProgram	Loads the PLC Basic editor program into RAM.
SetPrm Number, Value Number : parameter ID, from 0 to 999	Writing the Value of an InterpCNC parameter knowing its id. <u>Example</u> : SetPrm 20, 1300 ' Initial frequency of Axis 1 movements, set to 1300hz
Msg	Displays a message on the Oled screen of the card. Max length is 15 chararcters (if more, will be ignored). The message is always centered on the line. Same use as the Print command. <u>Example</u> : Msg "Prod =",Counter,"units" -> will show "Prod = 33 units"

Recipe management commands

CopyRCP Source recipe, Target recipe	Function that copies a Recipe to another Recipe location. <u>NB</u> : RCP_SIZE (adress 9995 -> number of registers allocated for each recipe) must have been defined beforehand. <u>Example</u> : CopyRCP 0,1 'Copies recipe 0 to recipe 1
InsertRCP Destination Position, Source Recipe	Inserts a copy of a recipe at the specified location. The following recipes are staggered. <u>Example</u> : InsertRCP 1,4 'Inserts a copy of recipe 4, in position 1
RemoveRCP RecipeNumber	Deletes a recipe. The following recipes are staggered. <u>Example</u> : RemoveRCP 1 'Supprime la recette n°1

Axis motion commands

SetPos AxisNumber, Value AxisNumber : Axis number 1 to 5 Value : Position value	Writing of the position counter of an axis. This function should not be called while the axis is moving. The AxisNumber parameter is used to indicate the axis concerned by the command. <u>Example</u> : SetPos 3, 1000 'Writes 1000 to the Y axis counter
MoveAxe AxisID, Accel, Speed, Decel, Position AxisID : Axis ID (1 to 6) Accel : Acceleration, in Hz/s Speed : Pulse Frequency, in Hz Decel : Deceleration, in Hz/s Position : Target, in motor steps	Moves the axis to a target position. <u>Example</u> : MoveAxe 1, 1500, 10000, 1500, 50936 'X movement
MoveSpeed AxisID, accel or decel (according to target), speed (signed)	Rotation to a specific speed (signed speed) <u>Example</u> : MoveSpeed 1, 1500, 10000
Home AxisID, InputNumber, InputState, Accel, HighSpeed, Decel, (+/-)MaxStep, LowSpeed,	Starts a homing sequence. Several commands can be launched simultaneously on different axes. This

<p>[HomePosition], [MoveStepAfterHome]</p> <p>AxisID : Axis to initialise (1 to 6) InputNumber: Entry number used for referencing InputState : State of the input when the contact is activated (0 or 1) Accel : acceleration for the rapid movement towards the sensor HighSpeed : Fast speed to sensor Decel : deceleration for the rapid movement towards the sensor (+/-)MaxStep : gives the direction of movement and the Max travel for the Homing LowSpeed : sensor clearance speed [HomePosition] Value at which the position counter is initialized before clearing [MoveStepAfterHome] target position for clearance (relative to home position)</p>	<p>function is used to find a home position using a sensor placed on the stroke of an axis. <u>Example :</u> Home 2, 2, 0, 25, 20000, 25, 1000000, 1000, 0 <i>"Axis 2 origin on input N°2 NC type, over a maximum travel of 1000000 steps. Home position initialized to 0</i></p>
<p>Probe AxisID, InputNumber, InputState, Accel, Speed, Decel, (+/-)MaxStep</p> <p>AxisID : Axis to initialise (1 to 6) InputNumber: Entry number used for referencing InputState : State of the input when the contact is activated (0 or 1) Accel : acceleration for the movement towards the sensor Speed : Speed towards sensor Decel : deceleration for the movement towards the sensor (+/-)MaxStep : gives the direction of movement (signed value) and the maximum stroke for probing</p>	<p>Launching a probing. <u>Example :</u> Probe 1, 3, 0, 25, 50000, 25, 1000000 <i>'Probing on Axis 1, sensor on input 3, NC type, acceleration and deceleration of 25kHz, speed of 50000 Hz, on 1000000 steps Max '</i></p>
<p>StopAxes Value Value : binary (selected axis=Bit to 1) or Hexadecimal, or decimal</p>	<p>Stopping of one or more axes. <u>Example :</u> StopAxes &B111111 'Stop for all 6 axes (binary axis selection) StopAxes &H3F 'Stop for the 6 axes (Value 3F in Hex for the selected bits) StopAxes 63 'Stop for all 6 axes (Decimal value for selected bits)</p>
<p>StopAxisID Value</p>	<p>Stopping an axis by its identifier (number and name). <u>Example :</u> StopAxisID 2 ou StopAxisID LABELLER AXIS</p>

Timers commands

<p>Pause ppp</p>	<p>Pause of ppp millisecond in program execution.</p>
-------------------------	---

<p>ppp = lasting time in ms</p>	<p>Interrupt processing, on the other hand, is not interrupted during a pause. This function can also be used in interrupt processing. <u>Example :</u> <i>'activation of output 1 for one seconde:</i> OUT 1,1 'Activation output 1 PAUSE 1000 'Pause of 1000ms OUT 1,0 'Desactivation of output 1</p>
<p>SetTimer NameOfVariable, Duration_ms</p>	<p>Initialization of a timer variable. The duration is expressed in milliseconds. Command for use with the GetTimer function. <u>Example :</u> SetTimer Tempo1, 500</p>
<p>Tic Number of instance (1 to 15)</p>	<p>Starts a time capture, in microseconds. <u>Example :</u> Tic 1</p>
<p>CycleStatInit Number of instance (1 to 5), min. time, current time, max. time</p>	<p>Initializes a time measurement function. The Values will be stored in the 3 MW registers (16bits) indicated. The times are given in 1/10 ms. <u>Example :</u> CycleStatInit 1, 3015, 3016, 3017 or CycleStatInit 1, TPLC_MIN, TPLC_ACTUAL, TPLC_MAX</p>

Input and Output commands

<p>SetIN Input Number, State Input : 0 to 255 State : -1 No forcing, forcing to 0, forcing to 1</p>	<p>Forcing the state of an input. <u>Example :</u> SetIN 3, 1 'forcing input 3 to 1</p>
<p>OUT OutputNumber, Value Number : 0 to 95 Value = 0 or 1</p>	<p>Activation/deactivation of a discrete output (OUT 0 to OUT 15). Outputs OUT 0 to OUT 15 are physical outputs. Outputs 16 to 95 are outputs that can be used via an external device (Modbus or USB interfaces) <u>Example :</u> OUT 4,1 'Sets the state of output 4 to 1</p>
<p>OUTPort NumPort, Value NumPort : 0 to 11 Value : 0 to 255</p>	<p>Used to define the state (0 or 1) on an 8-output (8-bit) port. <u>Example :</u> for i = 0 to 11 : OUTPort i, 0 : Next i 'Reset all outputs</p>
<p>SetAna Channel, Value Channel = 0 or 1 (AOUT0 ou AOUT1) Value = 0 to 10000</p>	<p>Sets the level of an analog output in mV points (0 to 10000). <u>Example :</u> SetAna 1, 5000 'Analog output AOUT1 is set to 5V</p>
<p>SetAnaV Channel, Tension Channel = 0 or 1 (AOUT0 or AOUT1) Voltage = 0 to 10V</p>	<p>Defines the level of an analog output in Volt (0 to 10v). <u>Example :</u> SetAnaV 1, 5.51 Analog output AOUT1 set to 5,51V</p>

IMPLEMENTING A GRAFCET (SFC) USING « Select Case» instruction

The “**Select Case**” instruction easily allows the creation of a **Grafcet** (aka **SFC**), and gives greater readability to your program, compared to a sequence of the type “If Step=10 then... Else... Endif”

The associated available instructions are:

Select Case Variable
Case Value
Case ValueFrom **to** ValueTo
Case Value **is** < Limit
Case Else
End Select

Example of use:

Suppose we want to create a PLC cycle of 4 steps (step 0, 10, 20, and 30) in which we will activate/deactivate output 1 according to the state of input 1, then activate/deactivate output 2 with a time delay of 1000ms.

We will use for Example a variable named “GCycle1” , which in turn will take the current step Value. This will be reassigned at the end of each step, which will allow you to move on to the next one on the next round of the DO ... LOOP loop.

```

GCycle1 = 0 ' Initialisation of the sequence to step 0
DO
  Select Case GCycle1

    Case 0
      if IN(1) then
        Out 1,1
        GCycle1 = 10
      Endif

    Case 10
      if not IN(1) then
        Out 1,0
        GCycle1 = 20
      Endif

    Case 20
      Out 2,1
      SetTimer tempo1, 1000
      GCycle1 = 30

    Case 30
      if GetTimer(tempo1)=0 then
        Out 2,0
        GCycle1 = 0
      Endif

    Case Else
      ? "Programming error"
  End Select
LOOP

```

Thus, each "Case .." instruction will make it possible to process, for each step Value of the

GCycle1 sequence, the code to be executed.

USING INTERRUPTS

You can program three types of interruptions.

- Periodic interruptions
- Interrupts related to the state of the inputs
- Interrupts on axis positions

The latency for processing an interrupt is $< 5\mu\text{s}$.

You can implement up to 32 different interrupt treatments (periodic or input-related).

Interrupt processing must end with the **iReturn** instruction

When interrupt processing is no longer required, you can disable it by specifying:

A period of 0 for Periodic interrupts,

An input number = -1 for an input related interrupt.

Periodic interruptions

Set up using the SetTick command.

A jump to the indicated label will be carried out according to the period specified when setting up the interruption.

SetTick InterruptNumber, Period, Label

InterruptNumber : Number of the interrupt from 1 to 32

Period : Interrupt period in milliseconds

Label : Label where to find the interrupt handler.

Example of periodic interruption:

```
SetTick 1, 500, OnInt1 ' Implementation of a periodic interrupt of 500ms
do
... ' application code
Loop
OnInt1 :
Out 5, not Out(5) ' Changes state of output 5
iReturn
```

Interrupts Related to the State of the Inputs

In order to lighten the writing of the application and react quickly to a change of input state, you can set up an interrupt treatment which will carry out this check and execute the desired code.

This type of treatment can support:

- The change of state of an input (change to 0 or 1),
- The transition from state 0 to state 1 of an input (rising edge),
- The transition from state 1 to state 0 of an input (falling edge).

This processing is implemented using the SetInputInt command.

SetInputInt :

Syntax : InterruptNumber, InputNumber, Type, Label

InterruptNumber : Number of the interrupt from 1 to 32

InputNumber : Number of the input to be monitored (0 to 255)

Type : Type of control

1 to control all status changes (INPUT_INT_DF)

2 to control the transition from state 0 to state 1 of the input (INPUT_INT_DFM)

3 to control the transition from state 1 to state 0 of the input (INPUT_INT_DFD)

4 to check all status changes, only once (INPUT_INT_DF_ONESHOT)
 5 to check the transition from state 0 to state 1 of the input, once only
 (INPUT_INT_DFM_ONESHOT)
 6 to check the transition from state 1 to state 0 of the input, once only
 (INPUT_INT_DFD_ONESHOT)
 Label : Label where to find the interrupt handler.

NB : to cancel an interrupt, just run the exact same SetInputInt command again, but use "-1" as the input number.

Example : Management of a limit switch input
*' Normally Closed limit switch on input 8
 SetInputInt 1, 8, 3, OnFDC1
 do
 ... 'Application code, here
 loop
 ' limit switch input processing
 OnFDC1:
 StopAxeID 1 'Axis 1 stopped
 iReturn*

Interrupts on Axis positions

SetCaptureInt : Interruption on axis position reached. The interrupt is automatically cleared. AxeNumber=0 to cancel the interrupt.

Syntax : **SetCaptureInt** IntNumber, AxisNumber, CapturePosition, LabelCode
 IntNumber : instance number (from 1 to 32)
 AxisNumber : 1 to 6 (on Motorized axes), 7 to 9 (on Encoder inputs), 10 to 16
 (on Counters on Fast Inputs)
 CapturePosition : target position (in motor steps)
 LabelCode : Interrupt code located at the Label

Example : **SetCaptureInt** 10, 2, LabellerTarget, OnLabellerPos

SetCaptureIDOnInputInt : Configuration of position captures on fast input interrupts (16 to 22). For use with GetCapturePos(AxeID)

The interest of this command is to allow the capture of positions on the axes, independent of the execution of the PLC Basic program. Thus, the captures having priority over the execution of the PLC Basic program, these take place with perfect regularity without variation in the response times.

Syntax : **SetCaptureIDOnInputInt** InputNumber, AxisID

Example : **SetCaptureIDOnInputInt** IN_CEL_LABEL, LABELLER_AXIS

NB: These interrupts do not work on a manual override of inputs, only on a physical signal. The input must also have been configured in IT or counter mode (parameters 220 to 226).

Only one input can be assigned to an axis. Any reassignment of an axis on an input cancels and replaces the previously declared interruption.

GetCapturePos(: Reading of the position captured on an axis during the interruption. (Configured with SetCaptureIDOnInputInt).

Syntax : **GetCapturePos**(AxisID)

Example : ? "Captured Position = ", **GetCapturePos(ORIENTATOR_AXIS)**

USING THE FAST DIGITAL INPUTS (from #16 to 22)

In Encoder mode

We can have 3 encoders because 2 fast inputs are required for each of the encoders.
 Channel 0: inputs 21 and 22. Frequency up to 1Mhz
 Channel 1: inputs 19 and 20. Frequency up to 50 khz
 Channel 2: inputs 17 and 18. Frequency up to 50 khz
 Card parameters 221 to 226 must be configured in Mode 4 (4X, all edges are taken into account), or in Mode 3 (2X, 1 edge out of 2 is taken into account).

GetEncoder(: Returns the position of an encoder input

Syntax : **GetEncoder**(Channel) '*Channel from 1 to 3.*

Example : SetMDW 3018, **GetEncoder**(3)

SetEncoder: Assignment of a Value to an encoder input

Syntax : **SetEncoder** Channel, Value '*Channel : from 1 to 3.*

Example : **SetEncoder** 3, 0 '*Setting encoder input 3 to 0*

In Counter mode

It is therefore possible to have 7 counters. Card parameters 220-226 need to be configured:

In mode 0 (standard), the refreshing of the table storing the state of the inputs takes place every millisecond.

In mode 1 (on "IT" interrupt), the refreshing of the table storing the state of the inputs takes place on each interrupt accessing it.

In mode 2 (Counter mode), the refresh of the table storing the state of the inputs takes place on each edge (rising or falling, depending on the configuration of the polarity of the inputs (card parameter 200)).

The period between 2 events can be read in the 32-bit modbus registers (Input registers) 1130 to 1142.

GetCnt(: Reading of one of the counters associated with fast inputs 16 to 22.

The result is an unsigned integer.

Syntax : **GetCnt**(CounterNumber)

CounterNumber = 1 to 7

Example : SetMDW 3018, **GetCnt**(4) '*Writes to address 3018 the Value read from counter n°4*

SetCnt : Writing in the counters of fast inputs 16 to 22. The InterpCNC has 7 fast inputs which can be used as counting inputs.

Syntax : **SetCnt** CounterNumber, Value

Counter Number = 1 to 7

Value = Value for the counter

Example : **SetCnt** 4, 0 '*Resets counter number 4*

COMMUNICATION WITH MODBUS DRIVES OR OTHER DEVICES

Each of the COM1 and COM2 ports can be used for **MODBUS** control of drivers or VFD. In this case, the InterpCNC V3 card will be "Master".

Beforehand, it will be necessary to study the documentation of your driver/VFD, to know the Modbus addresses of its main registers to use:

Basic communication

→ Mode, Baud Rate, Data Bits, Parity, Stop Bits, ID as Slave

In principle, these settings must be defined either from the keypad on the front of the drive, or using parameter setting software and connection to the PC.

Other settings may be necessary, such as input operating mode, direction of rotation, etc...

It will obviously be necessary to enter identical parameters on the side of the InterpCNC V3 card (registers 400 to 405 for the use of COM1, registers 420 to 425 for the use of COM2). Additional parameters:

COM1 : register 408 (minimum delay between each frame)
 register 409 (number of attempts before returning a communication error)

COM2 : register 426 (minimum delay between each frame)
 register 427 (number of attempts before returning a communication error)

Use of MBRTU Commands

The MBRTU.Send command will send a Modbus frame via COM1 or 2, destined for the drive ID (Slave), with a specified data type (Example: Holding Register).

The Value to be sent will be read from a source register or bit (or several consecutive ones) in the Master (the InterpCNC V3 card), to be written to the specified register or bit (or several consecutive ones) of the slave.

Thus, for example, we can write to the speed register of the drive to vary the latter.

Structure of the **MBRTU.Send** command:

(the members of a command can be either variables or direct Values)

MBRTU.Send COMPort, JobType, SlaveID, MasterAddress, SlaveAddress, DataSize, MIRegisterNumber

COMPort is the number of the COM port used (1 or 2)

JobType is the type of action (Read or Write) depending on the type of data access (Read/Write, Read only):

MB_RTU_WRITE_HOLDING_REG or «5», and **MB_RTU_READ_HOLDING_REG** or «2»

MB_RTU_WRITE_COIL or «4», **MB_RTU_READ_COIL** or «0»

MB_RTU_READ_INPUT or «1», **MB_RTU_READ_INPUT_REG** or «3»

MasterAddress is the source address on the Master side

SlaveAddress is the destination address on the Slave side

DataSize is the number of consecutive data (registers or bits)

MIRegisterNumber is a register in which an error code will be returned (-14 = no error)

It should be noted that the mode of access to Input Registers or Input Bits by adding

+100000 to the address also works.

It can, for example, be used to read them directly on the Master in the **MB_RTU_READ_HOLDING_REG** mode (instead of **MB_RTU_READ_INPUT_REG**) in order to directly copy the Value of an Input Register on the Master, to a Holding Register on the slave. Example :

```
const INPUTS_SHADOWS = 3022
const RESULT_COM_VAR5 = 3036
```

```
MBRTU.Push 2, MB_RTU_WRITE_HOLDING_REG, 5, 101000, INPUTS_SHADOWS, 2,
RESULT_COM_VAR5
```

This command will therefore copy via COM2, the state of physical inputs 0 to 31 of the InterpCNC card to registers 3022 and 3023 of the slave. (The result of the communication will be stored at the Master in address 3036).

The **MBRTU.Push** command is identical, except that successive commands will be in a queue, executed in the order of arrival.

MBRTU.Push COMPort, JobType, SlaveID, MasterAddress, SlaveAddress, DataSize, MIRegisterNumber

Example :

```
MBRTU.Push COMVariateurs, MB_RTU_WRITE_HOLDING_REG, VAR_CONVOYEUR,
HZ_CONVEYOR_BELT, REGISTER_SPEED_VAR, 1, RESULT_COM_VAR5
```

could also be written:

```
MBRTU.Push 2, 5, 5, HZ_CONVEYOR_BELT, 3, 1, 3030
```

Commande **MBRTU.Flush**(PortCOM) clears the not yet processed MBRTU command queue. Example : **MBRTU.Flush**(1)

Function **MBRTU.Count**(PortCOM) returns the number of orders currently in the queue.

Example :

```
if MBRTU.Count(COMDrive)>25 then
  ? "Drive command sending error"
endif
```

It can be used for example to detect a communication problem. Indeed, a number of unprocessed commands that accumulate in the queue are revealing.

ETHERNET COMMUNICATION

Each InterpCNC V3 card is equipped with an Ethernet port, which not only allows it to be used remotely by ICNCStudio, but also to communicate with each other.

The interest may be for Example to constitute a network of several PLC stations or axis controllers on a production line, needing only one program on the card called "Master" (or "server") .

In Ethernet, 2 protocols coexist: UDP and TCP.

UDP is the easiest to use. The TCP meanwhile, if heavier to implement, is much more reliable and more secure.

As a result, TCP was chosen for communication between InterpCNC boards. Thus we will be able to access the registers, in read or read/write, of each of the connected cards, and even send them Modbus commands.

Use, controls and functions

Here is a simple demo program:

```

1  ' SOPROLEC InterpCNC V3 PLCBasic
2
3  ' Copie état des entrées ICNC vers registre robot adresse 0
4  ' Copie Entrées robot adresse 10000 vers sorties ICNC
5  const ROBOT_OUTPUT_ADDR = 0      ' Adresse pour écriture des sorties sur robot
6  const ROBOT_INPUT_ADDR = 10000   ' Adresse pour lecture des entrées robot
7  const SCAN_RATE=1               ' Periode de rafraichissement (ms)
8
9  const ICNC_INPUT_ADDR=1000       ' Adresse modbus de l'état des entrées 0..15 de l'ICNC (Input Register)
10 const ICNC_OUTPUT_ADDR=2160      ' Adresse modbus de l'état des sorties OUT0..15 de l'ICNC (Holding register)
11
12 const IP_ROBOT="192.168.10.11"
13
14 ' Pour statistique communication (Adresses de registres internes ICNC)
15 const ETHERNET_RX_FPS= 1198
16 const ETHERNET_TX_FPS= 1199
17 const ETHERNET_TCP_CLIENT_ERROR=1201
18 const ETHERNET_TCP_CLIENT_SUCCESS=1203
19
20 ' Création socket de communication
21 SockRobot = MBClient.Open(IP_ROBOT, 502, MBB SOCK_BUSY)
22
23 do
24   ' Quand le socket est disponible
25   if not getMB(MBB SOCK_BUSY) then
26     ' Copy IN0..15 dans holding Register
27     SetMW REG ICNC_INPUT_COPY, GetInputMW(ICNC_INPUT_ADDR)
28     ' Envoie entrées ICNC 0..15 sur sorties robot
29     MBClient.WriteVar SockRobot, MB_WRITE_HOLDING_REG_TO_HOLDING_REG, REG_ICNC_INPUT_COPY, 1, ROBOT_OUTPUT_ADDR, 0
30     'Copie entrées robot vers sorties 0..15 de ICNC INPUT_ADDR
31     MBClient.ReadVar SockRobot, MB_READ_HOLDING_REG,ROBOT_INPUT_ADDR, 1,ICNC_OUTPUT_ADDR,SCAN_RATE
32   endif
33
34   ' Pour statistique communication
35   SetMW REG ETHERNET_RX_FPS, GetInputMW(ETHERNET_RX_FPS)
36   SetMW REG ETHERNET_TX_FPS, GetInputMW(ETHERNET_TX_FPS)
37   SetMW REG TCP_CLIENT_ERROR, GetInputMW(ETHERNET_TCP_CLIENT_ERROR)
38   SetMW REG TCP_CLIENT_SUCCESS , GetInputMW(ETHERNET_TCP_CLIENT_SUCCESS)
39
40 loop
41

```

We assume here that our Server card addresses a Robot equipped with a Client card whose IP address is 192.168.10.11

→ `const IP_ROBOT="192.168.10.11"`

We first define as constants the addresses of the memory areas where to write and read on the Client card (`const ROBOT_OUTPUT_ADDR = 0` and `const ROBOT_INPUT_ADDR = 10000`),

(`const ICNC_INPUT_ADDR=1000` and `const ICNC_OUTPUT_ADDR=2160`).

The following constants are not essential, they are just used here to calculate connection statistics.

1) Open a « Socket »

Prerequisite: choose a user bit (coil), which will report the availability of the socket to receive requests.

Example : `MBB SOCK_BUSY`, at Modbus address 96

```

SockRobot = MBClient.Open(IP_ROBOT, 502, MBB SOCK_BUSY)

```

Function **MBClient.Open** opens a Socket, and returns a Socket number that will be stored in a variable (here: SockRobot)

→ parameters: client IP address, port no., status bit name

2) Send read or write requests

NB : The Socket is available to receive read or write requests when its status bit (here,

MBB_SOCK_BUSY) is at 0.
 → `if not getMB(MBB_SOCK_BUSY) then`

REG_ICNC_INPUT_COPY is a 16-bit user register, say at address 3000, and **ICNC_INPUT_ADDR** (read-only address 1000) corresponds to the 16-bit mapping of digital inputs 0 to 15 of the Server card).

Thus :

SetMW **REG_ICNC_INPUT_COPY**, GetInputMW(**ICNC_INPUT_ADDR**) will copy the state of inputs 0 to 15 from the Server, to address 3000.

Write request :

MBClient.WriteVar SockRobot, **MB_WRITE_HOLDING_REG_TO_HOLDING_REG**, **REG_ICNC_INPUT_COPY**, **1**, **ROBOT_OUTPUT_ADDR**, **0**

Function **MBClient.WriteVar** allows to send a write request to the Client:

→ parameters :

IP address of client, command **MB_WRITE_HOLDING_REG_TO_HOLDING_REG**, starting address to copy (Server side), number of registers to copy, destination address (Client side), time allocated in ms.

Requête en lecture :

MBClient.ReadVar SockRobot, **MB_READ_HOLDING_REG**, **ROBOT_INPUT_ADDR**, **1**, **ICNC_OUTPUT_ADDR**, **SCAN_RATE**

Function **MBClient.ReadVar** allows to send a read request to the Client:

→ settings:

client's IP address, command **MB_READ_HOLDING_REG**, start address to copy (Client side), number of registers to copy, destination address (Server side), allocated time in ms.

NB: You can stack up to 10 requests (Read or Write).

The MBB_SOCK_BUSY Status Bit drops to zero after sending them.

3) Close a Socket

We use **MBClient.Close**

→ parameters : socket number

Example : **MBClient.Close** SockRobot

NB : Sockets are automatically closed when stopping the PLC program.

USING DMX COMMUNICATION

Introduction :

DMX is a very widespread communication protocol in the world of entertainment, allowing the control of various devices (light games, motorized actuators (patiences, spinners, etc.).

DMX 512 exists in Device mode (for drives or projectors) or in Master mode to control the Devices (Example: lighting console).

The InterpCNC allows you 2 operating modes:

Device mode available on the COM1 port

Master mode available on the COM2 port.

In Device mode, you can therefore control outputs or motors from a DMX console.

In master mode, you can program device control sequences by interacting with the PLC inputs.

I) Connection:

Connecting your DMX512 console to the InterpCNC V3 card is very simple. Just connect the

D+ (green wire) and D- (yellow wire) signals from the DMX socket of your console, on the D+ and D- inputs of the COM1 or COM2 port of the InterpCNC card.

II) DMX Device Mode Setup:

In the table of parameters, it is a question of configuring the COM1 port in DMX mode:

The screenshot shows the 'Paramètres' window for 'Port COM1'. The 'Mode' is set to 'DMX Slave'. The 'Communication settings' include: Baud rate: 256 000 bauds, Data bits: 8 bits, Parity: None (Aucune), and Stop bit: 1 bit. The 'Modbus Slave settings' include: Slave ID: 1. The 'Modbus master settings' include: Inter frame delay (ms): 0 and Max retry: 3. The 'DMX Settings' include: Base address: 1 and Channels used: 512.

The Base address is the number of the 1st DMX channel from which you want to work, and the number of channels used will indicate the range of channels used from the Base Address.

The “Base address” parameter is used to modify the DMX address of the PLC without renumbering the channels used in the PLC program.

III) Using DMX Device in your program:

The available functions are :

StsBit(*STS_DMX_CONNECTED*) → This status bit indicates whether a DMX link is established (If at 1).

IsDMXReceived → Indicates that a DMX frame has just been received (If at 1). It is automatically cleared after reading.

Examples :

if IsDMXReceived then

...

```

    endif
or again
    SetMB DMXReceived, IsDMXReceived
    (→ copy of the state of the system bit, in a user bit named DMXReceived)

```

When this bit is at 1, it is time to read the channels using the following functions :

GetDMX(channel number (1 to 512)) → Retrieves the Value between 0 and 255 present on this channel.

Note: The channel number actually used depends on the "Base address" parameter. If the parameter is set to 1, the GetDMX(1) command returns the Value of the first DMX channel. If the "Base address" parameter is 10, the GetDMX(1) command will actually return the Value of channel 10.

Examples :

```

    speed = GetDMX(2)
    or
    if GetDMX(1)>127 then
        Out Output1 = 1
    else
        Out Output1 = 0
    endif

```

or again

GetDMX16(n° of the 1st channel (1 to 511)) → Retrieves Values from 0 to 255 of 2 consecutive channels,
and restores a Value from 0 to 65535
Formule : Value of 1st channel + Value of 2nd channel*256

III) Using DMX Master in your program :

SetDMX channel number, ValueDMX → Writing an 8-bit Value to a DMX channel

SetDMX16 channel number, ValueDMX → Writing of a 16-bit Value on 2 consecutive DMX channels.

SetDMXMaster MasterLevel → Weighting of all DMX channels.

Example :

SetDMXMaster 255 ' the DMX channels will be transmitted as given by the SetDMX or SetDMX16 commands

SetDMXMaster 0 ' The channels will be sent with a Value 0 (black out)

SetDMXMaster 127 ' The channels will be sent with a Value divided by 2

All the channels and also the DMX Master Value are accessible via modbus in the read/write registers from addresses 5000 to 5512.

USING REAL TIME CLOCK (RTC)

Real Time Clock (RTC) functions

The InterpCNC has an internal clock to manage the date and time. This clock however, is not saved when the power is turned off. It should therefore be initialized before using its functions.

Initialization can be done by:

The modbus commands 112, 113 and 114 (detailed in Modbus InterpCNC documentation)

The PLC program using the RTC command

Automatically via an SNTP server if the InterpCNC has internet access.

Automatic synchronization:

For automatic update by STNP, parameters 546 and 547 must be correctly set. The SNTP server used is "sntp.pool.org".

The clock will then be initialized taking into account the time zone indicated in parameter 547 and

summer/winter time if bit b1 of parameter 547 is active.

You have 2 status bits that allow you to determine the synchronization status:

stsBit(**STS_RTC_SYNCHRONIZED**) which indicates that the clock has been set,

stsBit(**STS_SNTP_CONNECTED**) which indicates that an SNTP connection is established.

Synchronization by SNTP server, if activated, is automatically renewed every hour.

PLC Basic commands and functions:

You have 3 instructions to exploit the RTC clock. For each of them, several function codes will be detailed below:

RTC **RTC_SubCommandCode**, ... for commands that do not return a response,

RTC(**RTC_SubFunction**, ...) for functions that return a numeric Value,

RTC\$(**RTC_SubFunction**, ...) for functions that return a character string

Function codes are pre-defined constants in ICNCStudio (System\RTC)

Also note that several of the functions detailed below use variables of type

UnixTime. This is an internal format and therefore cannot be directly manipulated by users. arithmetic function (addition, subtraction). For operations on these variables, please use the functions provided.

The information managed by the RTC (date and time) is also available in the Modbus registers (Input registers) 1987 to 1995

You can therefore access it from the PLC with the command GetMW(at addresses 101987 to 101995.

RTC command details:

RTC command code (Syntax: RTC RTC_SubCommand,

RTC **RTC_SetTime**, hh, min, ss

=> Manual clock time setting. If SNTP synchronization is active, this operation is not necessary because the time will be synchronized automatically.

RTC **RTC_SetDate**, dd, mm, yy

=> Manual clock date setting. If SNTP synchronization is active, this operation is not necessary because the date will be synchronized automatically .

RTC **RTC_SetAlarmeA**, hh, min, s, dayOfMonth_or_DayOfWeek[, d_meaning=0][,

mask=0]

=> Programming of alarm A. The triggering of alarm A causes a call to the sub routine onAlarmA() which must exist in your program.

- dayOfMonth_or_DayOfWeek can be any day in month from 1 to 31 if d_meaning =

RTC_DAY_OF_MONTH or a weekday from 1 to 7 if d_meaning = **RTC_DAY_OF_WEEK**

- mask allows advanced programming of the alarm.

bit 0 of mask is used to mask the seconds of the time

bit 1 of mask is used to mask the minutes of the hour

bit 2 of mask is used to mask the hours of the hour

bit 3 of mask hides the day of the date

Setting a mask bit to 1 triggers the alarm regardless of the Value of the concerned field.

Example:

mask = &b1000, day is masked => the alarm will go off every day at hh:mn:ss (whatever dd)

mask = &b1100, day and time are masked => Periodic alarm every hour at mn:ss (whatever dd, hh)

mask = &b1110, day, hour and min are masked => Periodic alarm every minute at ss (whatever dd, hh, mn)

mask = &b1111, day, hour and min and second are masked => Periodic alarm every seconds (regardless of dd, hh, mn, s)

RTC RTC_SetAlarmeA, hh, min, s, dayOfMonth_or_DayOfWeek[, d_meaning=0][, mask=0]

=> identical to alarm A but call to a subroutine called onAlarmB

RTC RTC_StopAlarmA

=> Disable alarm A

Alarms are automatically disabled when the PLC enters STOP mode

RTC RTC_StopAlarmeB

=> Disable alarm B

Alarms are automatically disabled when the PLC enters STOP mode

RTC RTC_SetAlarmAutime, utime[, mask=0]

=> operation identical to RTC_SetAlarmeA but the date and time of the alarm are given by

an instance of UnixTime type.

UnixTime type instances are obtained with the RTC(...) functions and allow you to simple time operations.

RTC RTC_SetAlarmButime, utime[, mask=0]

=> identical to alarm A but call to a subroutine called onAlarmB

RTC RTC_CalcSunPosition, utime, latitude, longitude, MF_AZIMUTH, MF_ELEVATION =>

Calculates the position of the sun at a given time. The results are given in registers of type Float (register numbers given in MF_AZIMUTH and MF_ELEVATION)

Example :

const MY_LATTITUDE = 48.064422

const MY_LONGITUDE = 0.28127

Actualutime = RTC(**RTC_GetActualUTime**) ' Current date and time

RTC RTC_GetSunPosition, Actualutime, **MY_LATTITUDE**, **MY_LONGITUDE**, **SUN_AZIMUT**, **SUN_ELEVATION**

Details of the RTC(function

utime = RTC(**RTC_GetActualUTime**)

=> returns a UnixTime type instance representing the current date and time

utime = RTC(**RTC_GetMakeUTime**, d,m,y,h,mn,s)

=> creation of a UnixTime type instance corresponding to the date and time indicated
 Delta_s = RTC(**RTC_DiffTime**, utime1, utime2)

=> returns the time difference in seconds between two instances of UnixTime type. Let
 utime2-
 utime1 in seconds.

sum_utime = RTC(**RTC_AddToUtime**, utime, +/-second

=> returns a UnixTime type instance corresponding to a temporary delay of the instance
 ultimate. The offset is expressed in seconds and can be positive or negative.

DayLen = RTC(**RTC_DayLength**, day, month, year, lon, lat[, phase=0]) => Returns the
 duration in
 time of day.

sunriseHour = RTC(**RTC_SunRise**, day, month, year, long, lat[, phase=0][,
 do_adjustDST=1]) => Sunrise time at the specified longitude and latitude (in hours)

sunsetHour = RTC(**RTC_Sunset**, day, month, year, long, lat[, phase=0][,
 do_adjustDST=1]) =>

Sunset time (in hours) at the indicated longitude and latitude

SunRiseUtime = RTC(**RTC_SunRiseUTime**, UTime, longitude, latitude[,
 type=RTC_SUNSET_TIME][,do_adjustDST=1]) => Sunrise time in UnixTime format.
 Can be used to perform calculations on hours,

SunSetUTime = RTC(**RTC_SunsetUTime**, UTime, longitude, latitude[,
 type=RTC_SUNSET_TIME][,do_adjustDST=1]) => Sunset time in format
 UnixTime. Can be used to perform hour calculations.

Details of the RTC\$(function

RTC\$(**RTC_GetTimeStr**)

=> Returns a string representing the current RTC time in "hh:mn:ss" format

Example: ? RTC\$(RTC_GetTimeStr); "Time-stamped information message"

RTC\$(**RTC_GetDateStr**)

=> Returns a character string representing the current RTC time in "dd/mm/yy" format

Example: ? RTC\$(RTC_GetDateStr); "Time-stamped information message"

RTC\$(**RTC_GetDateTimeStr**)

=> Returns a character string representing the current RTC time in the format

"dd/mm/yyh:mn:ss"

Example: ? RTC\$(RTC_GetDateTimeStr); "Time-stamped information message"

ICNCStudio

ICNCStudio - Development and diagnostic software

The InterpCNC V3 card comes with the ICNCStudio software, designed specifically for
 development, diagnostics, and debugging of your PLCBasic programs.

It allows access to all the functions of the card and to all parameters.

Please note that this software and in particular the movement functions must be reserved
 for experienced users with advanced knowledge of programming and/or automation.

More than a tool, ICNCStudio is a real software development studio (hence its name) for your InterpCNC V3 card.

Unlike the previous generation of "Test Center" software, it brings together in a single screen all the windows dedicated to each type of parameter.

These windows are no longer floating and independent because they are part of a larger common window.

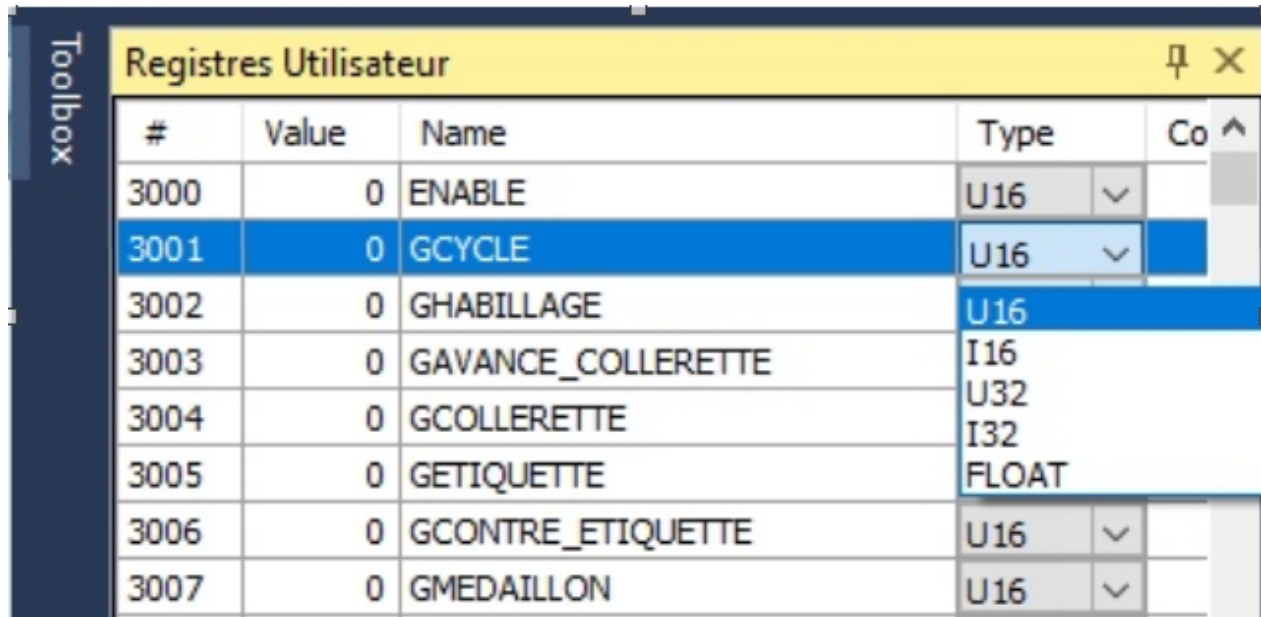
However, they remain resizable and movable at your convenience on the application space, and you can "Lock the layout of windows" (see "Tools" tab).

Integrated context-sensitive help is available by pressing the "F1" key wherever your cursor is, or for the text selected in the editor (Example: finding the syntax for using a command or a function).

User_Registers

- **User Registers** window (= "Registres Utilisateur") -> RAM
These registers correspond to Modbus addresses 3000 to 3999.

For each User Register, you directly set the type in the "type" column of the register, as follows:



#	Value	Name	Type	Co
3000	0	ENABLE	U16	
3001	0	GCYCLE	U16	
3002	0	GHABILLAGE	U16	
3003	0	GAVANCE_COLLERETTE	I16	
3004	0	GCOLLERETTE	U32	
3005	0	GETIQUETTE	I32	
3006	0	GCONTRE_ETIQUETTE	U16	
3007	0	GMEDAILLON	U16	

U16 and **U32** : 16 and 32 bit integers.

I16 and **I32** : 16 and 32 bit signed integers.

FLOAT : floating numbers.

You can also for each line, insert a comment (for Example to specify a measurement unit, etc...)

Contextual Menus :

- **On the User Register tables:**

« Insert Line » and « Remove Line »

Saved_registers

- **Saved Registers** window (= "Registres sauvegardés")

We may still call it EEPROM in this manual, but in fact this is rather non volatile RAM. These registers correspond to Modbus addresses 4000 to 4999.

NB: This saved memory area does not correspond to a classic EEPROM, but to Ferromagnetic RAM memory.

The latter, of a much more advanced technology, allows almost unlimited write cycles, and much faster access times.

For each Saved Register, you directly define the typing in the "type" column of the register, as follows:

Registres sauvegardés				
#	Value	Name	Type	Commer
4000		EE_R_COLLERETTE	FLOAT	pulse...
4001			U16	
4002		EE_R_ETIQUETTE	FLOAT	pulse...
4003			U16	
4004		EE_R_CONTRE_ETIQUETTE	FLOAT	pulse...
4005			U16	
4006		EE_R_MEDAILLON	FLOAT	pulse...
4007			U16	
4008		EE_R_ORIENTEUR_ENTREE	FLOAT	pulse...
4009			U16	
4010		EE_R_CONVROYEUR	FLOAT	pulse...
4011			U16	
4012		EE_V_ORIENT_ETIQ_INIT	U16	Hz ...
4013		EE_AD_ORIENT_ETIQ_INIT	U16	Hz ...

U16 and **U32** : 16 and 32 bit integers.

I16 and **I32** : 16 and 32 bit signed integers.

FLOAT : floating numbers.

You can also for each line, insert a comment (for Example to specify a measurement unit, etc...)

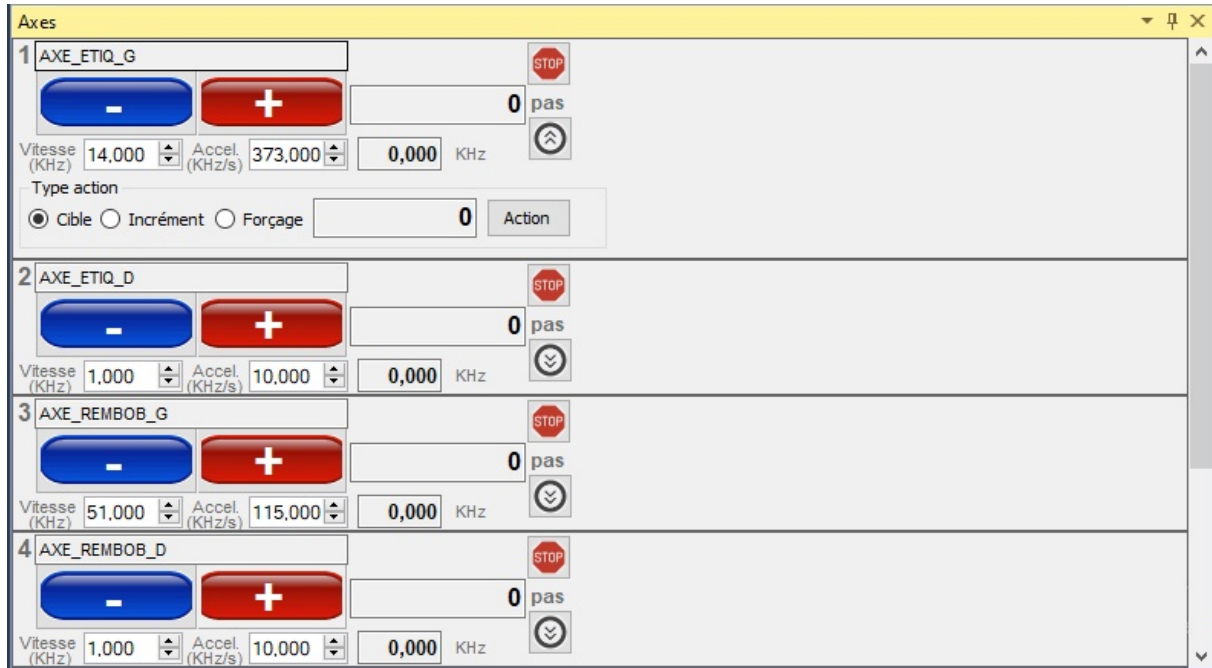
Contextual Menus :

- **On Saved Register tables:**
« Insert Line » and « Remove Line »

Axes_window

Axes window, allowing manual control and testing of movements on the 6 axes.

The card must first be in Enable and Unlock mode.



The icon under the word "pas"(=steps) displays a line of options.

1) Choose option

It is thus possible to go:

- To a target absolute position ("Cible")
- To a position incremented from the current relative position ("Incrément")
- Forcing ("Forçage") simply allows you to force the Value of the position counter for the axis.

2) Enter a Value (in steps, i.e in pulses)

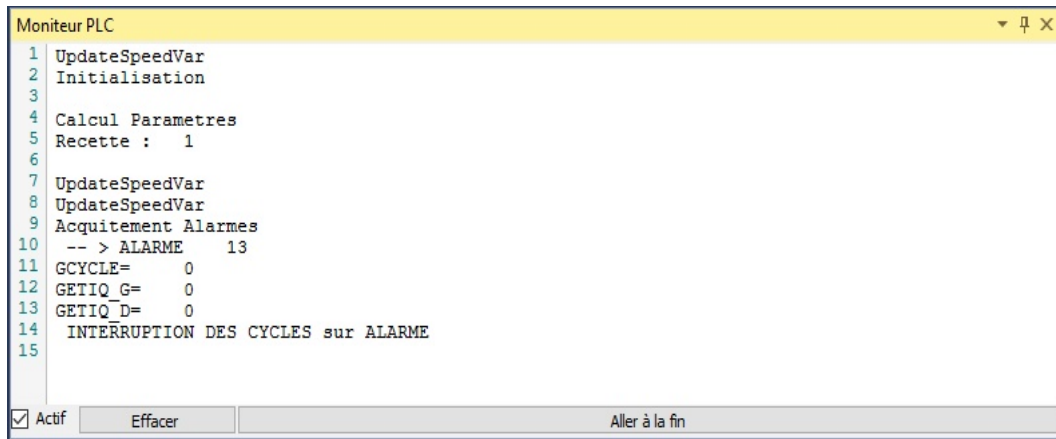
3) Click on "**Action**"

The STOP button is used to immediately stop the movement in progress.

Monitor

- **Monitor PLC** window (=Console)

In this window are displayed your Prints ("?",) during the execution of the program, thus allowing debugging, or to follow the good progress of the steps.



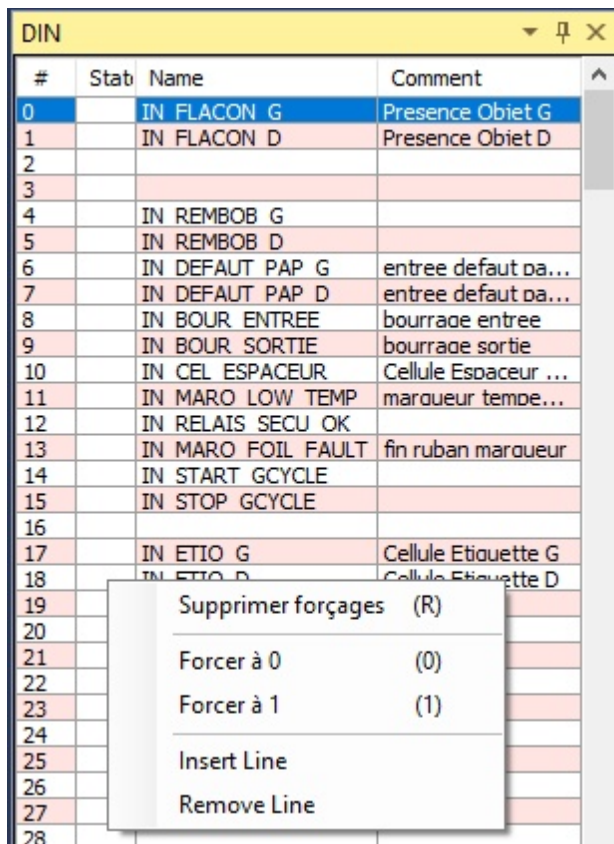
Digital_inputs

Digital Inputs window (DIN= Digital IN)

This window is actually tables refreshed almost in real time when the connection is established between ICNCStudio and your card.

In this table, you can enter and assign a name to the corresponding Bits, and force their state (0 or 1).

You can also for each line, insert a comment (for Example to specify the role of this entry...)



Contextual Menus :

- **On the Inputs table (DIN) :**
« Remove Forcing (R) », « Force to 0 », « Force to 1 », « Insert Line », « Remove Line »

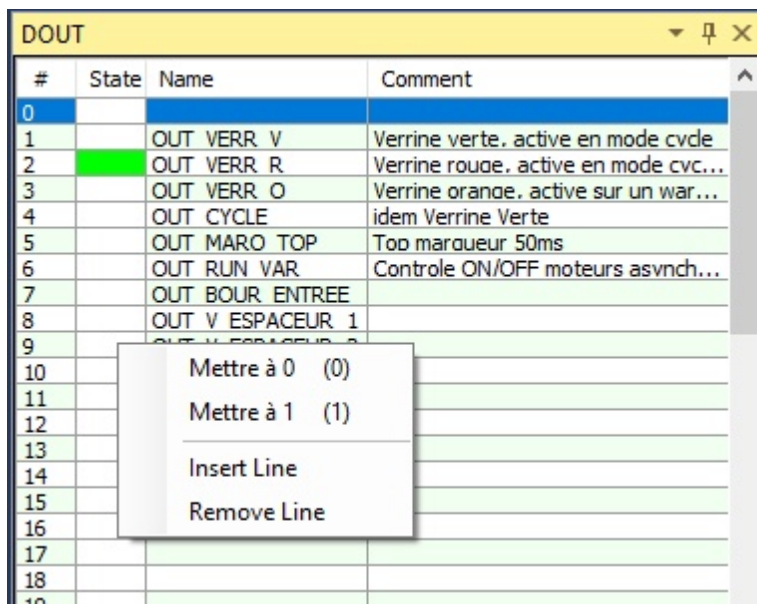
Digital_outputs

Digital Output window (DOUT = Digital OUT)

This window is actually tables refreshed almost in real time when the connection is established between ICNCStudio and your card.

In this table, you can enter and assign a name to the corresponding Bits, and force their state (0 or 1).

You can also for each line, insert a comment (for Example to specify the role of this output...)



#	State	Name	Comment
0			
1		OUT VERR V	Verrine verte. active en mode cvde
2		OUT VERR R	Verrine rouge. active en mode cvc...
3		OUT VERR O	Verrine orange. active sur un war...
4		OUT CYCLE	idem Verrine Verte
5		OUT MARO TOP	Too maraueur 50ms
6		OUT RUN VAR	Controle ON/OFF moteurs asvnh...
7		OUT BOUR ENTREE	
8		OUT V ESPACEUR 1	
9		OUT V ESPACEUR 2	
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			

Context menu options:

- Mettre à 0 (0)
- Mettre à 1 (1)
- Insert Line
- Remove Line

Contextual Menus :

- **On the Outputs table (DOUT) :**
« Force to 0 », « Force to 1 », « Insert Line », « Remove Line »

Coils

User Bits window (COILS, also called Memo Bits)

This window is actually tables refreshed almost in real time when the connection is established between ICNCStudio and your card.

In this table, you can enter and assign a name to the corresponding Bits, and force their state (0 or 1).

You can also for each line, insert a comment (for Example to specify the role of this bit...)

#	State	Name	Comment
96		MBB GCYCLE	
97		MBB GETIO G	
98		MBB GETIO D	
99		MBB GMARQUEUR	
100		MBB RAZ CPT CYCLE	
101		MBB RAZ CPT CUMUL	
102		MBB WARNING	
103		MBB BOUR SORTIE	
104			
105		MBB CONVOYEUR	
106		MBB TAPIS SUP	
107			
108			
109			
110		MBB ALARME	bit ouverture Window Alarme IHM
111		BASE ALARME	adresse de base des bits d alarme
112			
113			
114			
115			
116			
117			
118			
119			
120			
121		MBB ETIO ACTIVE G	Etiquette gauche active
122		MBB ETIO ACTIVE D	Etiquette droite active
123			
124			
125		MBB EFFACE CARTE	
126			

Contextual Menus :

- **On the COIL table (= Memo Bits) :**
« Set to 0 », « Set to 1 », « Insert Line », « Remove Line », « Search in the Program »

Text_editor

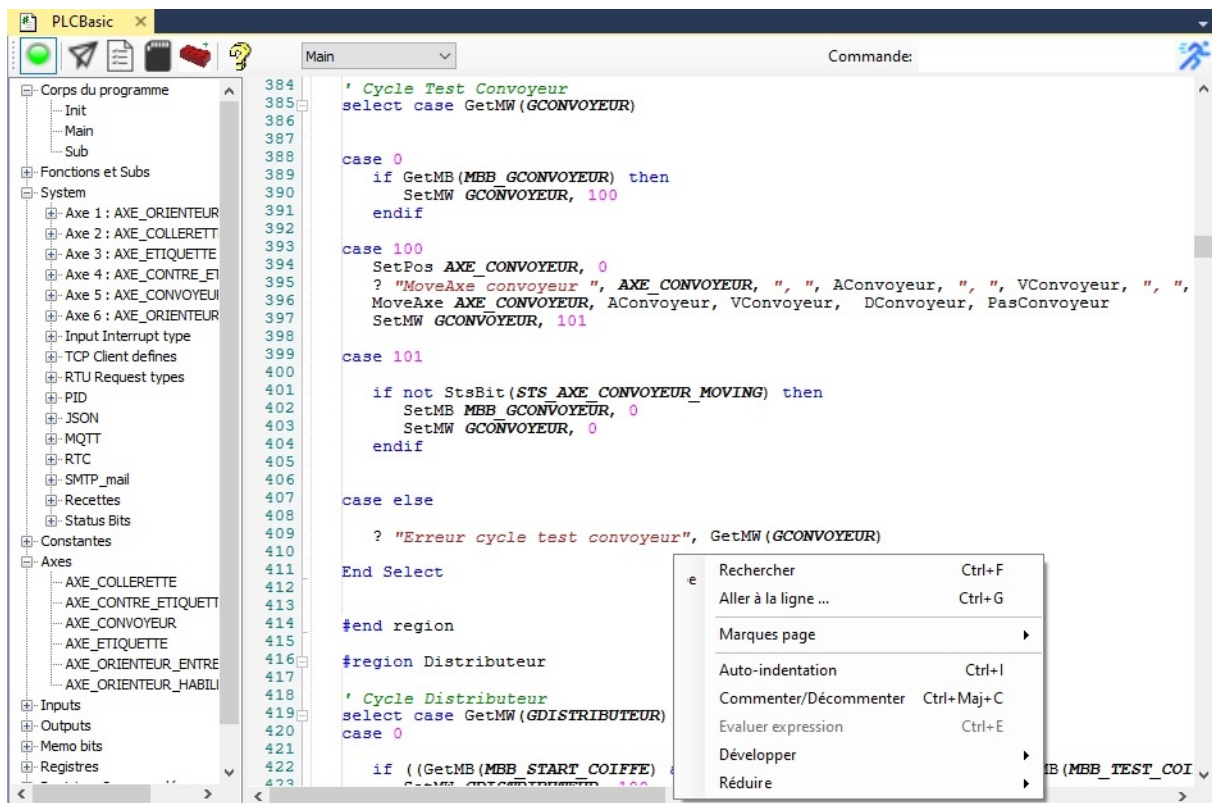
- **PLCBasic Program** window, with to its left the Definition List of all the parameters used by your Basic program:
- Program Body, your Functions and Subroutines, System (bit and registers), Constants, Axes, Inputs, Outputs, User Bits (Memo), User Registers, Saved Registers, and Recipes.

NB : The **Definition List** is automatically fed by your declarations of constants in the program, as well as your naming in the arrays. It is scrollable (clicks on +/-).

From this list, you can also "Drag and drop" the name of a variable, constant, register, input, output or a system parameter, to the input area of the text editor for the y copy directly, which is a form of autocompletion.

A click on **the green light**, allows you to run or stop the Basic program currently in Ram or in the non-volatile memory of the card.

- **The editor's Command bar**
This is part of the Basic program window. When the program is not running (ie stopped), you can run a command in Basic.
- NB: the variables used in the program are available for the command line.
- Example 1: ? CounterProd
- → will display in the monitor the last Value contained in your CounterProd variable
- Example 2: for i= 3000 to 3511:SetMW i,0: Next i
- → command erasing all the user registers of the Ram.
-
- The central window is common to the Basic program, the list of board Parameters, and the Recipe Editor. The tabs allow you to switch from one to the other.



Optimized readability of your programs

All: Register name, Bit name, Input name, and Output name, used become more readable (Caps+Italics) if they are already declared in the corresponding table. So, in the absence of Shift+Italics, you know that you are surely in the presence of a typing error.

- Basic functions, commands and instructions appear in **Blue**.
- Numerical Values appear in **Rose**.

- Comments (after quotes or between quotes: ' or ' ') appear in **Green**.
- The contents for the "print" ("?",) command are in **Red**.
- When you click on a variable, Bit name, Register name, Input or Output name, all occurrences are shown with an **Orange** highlight.
- With the card connected, when you leave your cursor over a constant associated with a register, its contextual information will appear (Modbus address, type of register (MW, MDW, etc.)

Structure of a Basic program

By default, the Program Body includes an **"Init"** block, as well as a **"Main"** block.

The **Init** block must contain all your constant declarations, as well as the code used, as its name suggests, to initialize your program.

The principle is that the code contained in this block will be executed once when the program starts, then the execution will continue on the Main block.

The **Main** block will contain your main code (combinatorial code, PLC cycles, etc...). All the code contained in this block will be repeated in a loop, just as if it were between **DO... LOOP** tags

You can create as many additional blocks as necessary, by clicking on the icon symbolizing a **red brick**. Their naming is free.

The ideal is to place in this type of block your functions, your subroutines, as well as the code corresponding to the labels of your interrupts.

Ultimately, this is code that is executed occasionally, ie called by the main program (from the **Main** block).

Including one block in another

At any point in your program, whether in the Main block or in any other block, it is possible to link the current execution with the contents of another complete block.

We will use the **Include** statement for this.

Example:

Suppose we have created a block named "Functions".

If in the **Main** block we encounter:

#Include Functions

Then the execution of the Main block will proceed transparently as if all the lines of code of the Functions block were in place of the **"#Include** Functions" line.

Regions

Derived from other languages, the notion of region has also been implemented in ICNCStudio, as it also contributes to the readability of your program.

#region Name and **#end region** tags, are used to delimit certain portions of the program, which you can then hide (click on -) or display again (click on +).

This way you have fewer lines to cover if you hide areas that have already been completed and tested, or that do not concern a feature that you are developing.

Integrated Help

By clicking on the yellow question mark "?", or by pressing the **F1** key, you display the help topics that you can browse or in which you can enter a search (keyword).

If you have previously made a selection in the editor (of an instruction, a command, a function, etc.), then the help will offer you either to choose an occurrence if there are several, or will point directly to the corresponding item.

Keyboard shortcuts

Ctrl+A : Select All

Ctrl+B : Add Bookmark

Ctrl+Maj+B : Delete Bookmark

Ctrl+C : Copy selection

Ctrl+Maj+ C : Comment/Un-comment

Ctrl+E : Evaluate an expression → If ICNCStudio is connected to the card, returns the current Value of the expression

Ctrl+F : Search for an expression (Find)

Ctrl+G : Go to line #xxx

Ctrl+H : Search and Replace

Ctrl+I : Auto-indentation of the selection (of the whole page if no selection)

Ctrl+N : Next bookmark

Ctrl+Maj+N : Previous bookmark

Ctrl+O : Exit ICNCStudio

Ctrl+S : Save changes

Ctrl+U : Set selection to Uppercase

Ctrl+V : Paste selection

Ctrl+X : Cut selection

Ctrl+Z : Cancel last action

Ctrl+Maj+Z : Re-do last action

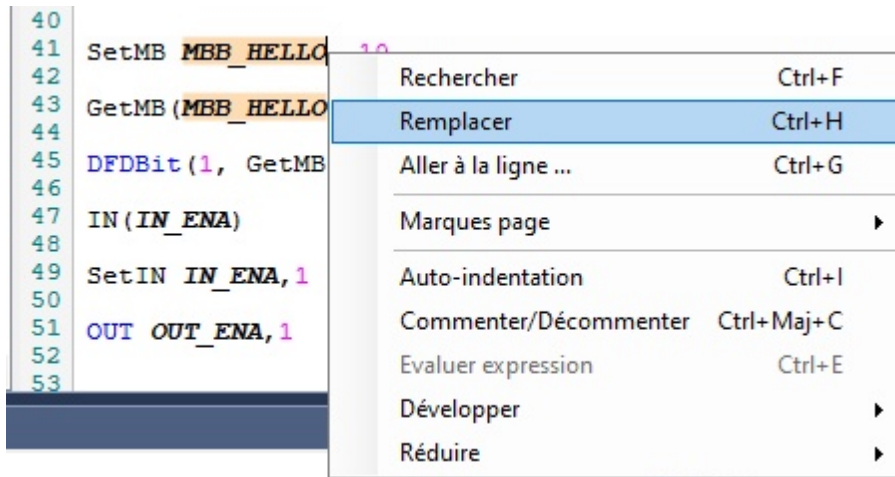
Shift+Ctrl+C : Comment selection

Find_Replace

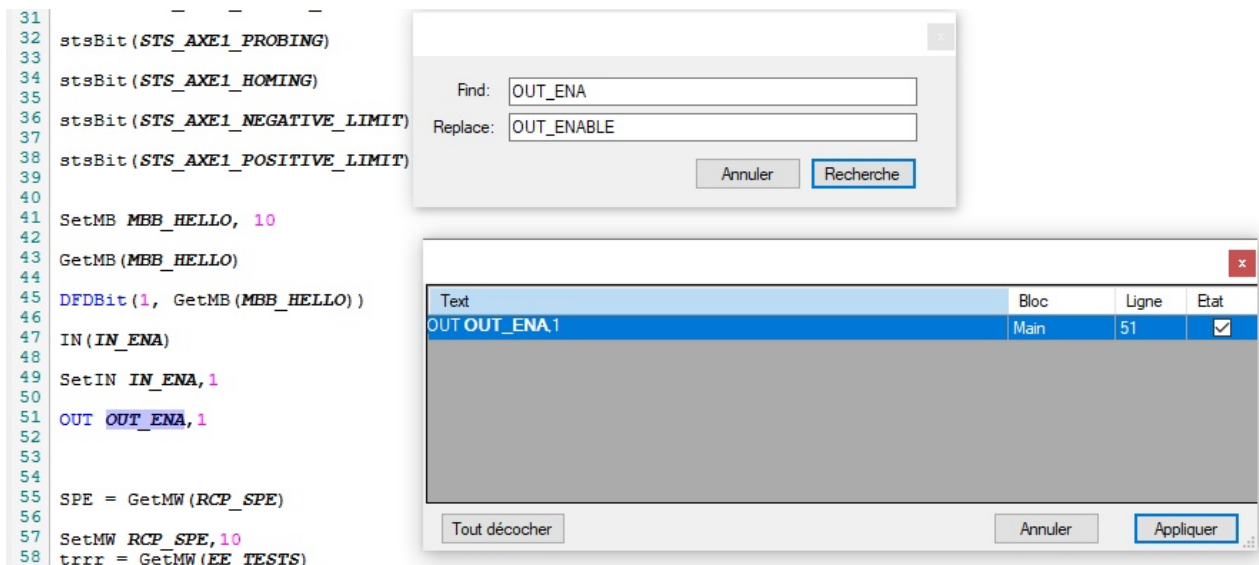
The **Find and Replace** function of ICNCStudio is very powerful, because it allows you to automatically substitute the name of a variable in your PLC Basic program, or that of a Register, or even that of a Bit, by another name for all occurrences and in all areas of your PLC project.

Thus this renaming, requested from the text editor, will be carried out automatically not only in all the program blocks (Init, Main, and others created by you...), but also in all the tables concerned (User registers, Saved registers, DIN, DOUT, COILS, Axes, Recipes, ...

The **Find and Replace** window is launched either with a right click on the target expression, then choose **“Replace”**(Remplacer)



or with the keyboard shortcut **Ctrl + H**:



NB: All occurrences encountered then appear in a window such as the one above.

Then simply keep the occurrences for which you want to apply the renaming checked (and

uncheck the others).

The search applies to all cases of case (upper and lower case), however, the case for replacing the expression will be exactly identical to what is entered in the "**Replace**" field ("Remplacer").

Main

Overview of Menu bar and Toolbar:



The Menu bar:

- **File** : Open Project, Open recent project, Save Project, Save as, Export PLC program in HTML format, Export **Declaration Kinco Address Tag**, Exit.
- **View** : choice of windows or tables to display, display of bars.
- The "Graph" option opens in the central window, a graph allowing to visualize on the Time (X) and Position (Y) axes, the speed curves of manual movements launched from the "Axes" window (choose the Axis).

The option **All PLC variables (= "Toutes variables PLC")** is another version of the card's system settings table. It allows access to these parameters no longer from their parameter number, but rather with their Modbus address.

- The interest is also to clearly identify their format (32 bit, 16 bit, signed or not, bit, etc.), and to have sorting possibilities to find them.
- **PLC Edit** (List of definitions visible, Program Card visible, Export analyzed program).
- "Carte programme Visible" (Visible Program Map): This feature gives a very condensed overview of the general structure of your program. It uses your own visual memory.
So you can go directly to a place whose structure you visually recognize.
Example: declaration of constants, declarations of variables, Select... case, loops, etc...
The course on the program is done with the mouse wheel, or left click + up or down on the visible program card.
- **Tools**: Firmware Update, Theme, Document Style, Reverse View, Settings File Export.
The Firmware Update window also allows you to display information specific to your card: serial number (CPLUID), MAC address, Firmware version, etc...
- **Window**: selection of a window from the already existing tabs in the central window.
- **Help** / About

The Tools bar:

- Connection icon:

It is used to call up the window used to select and/or configure the type of connection to be established with the card (Serial (USB or RS485) or Ethernet (TCP or UDP, +port)).

Common function icons:

New Project, Open Project, Save Current Project, Save As...

Display icons of different windows (if hidden, otherwise reminder):

Monitor Display, Motion Test Display, Inputs Display, Outputs Display, COILS Display, User Registers Display, Saved Registers Display.

In the central window: display of the PLC Program, display of the table of constants, display of the Table of parameters (settings) of the card, or display of the Recipe editor.

The Status bars :

- **The upper bar** states:

→ the type of connection established with the card (USB, TCP IP, etc.), as well as the location of the current program file.

- **The lower bar** states:

→ the processor occupation rate in real time (% CPU available), as well as the card availability rate.

→ if a program is running (PLC RUNNING/PLC STOPPED), with Start/Stop command

→ If the card ENABLE input is active (ENABLE/STOP)

→ Data exchanges with the card (packets sent (S:), packets received (R:), transmission errors (Err:), and connection status (CONN: True or False).

**Parameters_table**

There are 2 ways to access the card settings.

Either from the Parameters icon:



Either from the Menu:

View -> Parameters InterpCNC

In this case the parameters have a number located between 20 and 1515.

Description of the most usefull parameters

Some of the most useful settings parameters include:

The initial frequency of Axes movements → parameters 20 to 25

Also called "Frequency Start", this parameter reduces the time required for the acceleration phase (without increasing its speed) by not starting at frequency 0. Please note, however, that some applications may not support certain settings (bottles on conveyor, etc.)

Reversal of the direction of rotation → parameter 30

0: default direction. 1: reverse direction. 1 bit per axis.

Input Polarity → parameters 200, 201 and 202

These are 3 32 bit registers, mapping the operating state of each input (1 bit per input). Value: 0 or 1.

This setting can, for example, allow you to use an input signal level opposite to that provided in the program, without having to modify the existing program.

On 0 (normal mode) → a low level gives a reading of 0, a high level gives a reading of 1

On 1 → a high level gives a reading of 0, a low level gives a reading of 1

The configuration of Inputs 0 to 15 → registers 210 to 213:

See chapter DIN parameters of the card.

The configuration of Inputs 16 to 22 (the Fast Inputs) → registers 220 to 226

These 7 inputs are configurable as follows:

0: refresh on each line of the program.

1: refresh on each interrupt accessing this input.

2: counter mode

3: 2X encoder mode

4: 4X encoder mode

Analog input configuration

The gain: Setting to 1, therefore directly the resolution of the converter.

Default Value 20.07979 is an adjustment for input resistances.

The offset: allows you to set the Value 0 (because you can measure from -10V to +10V)

The scale of measurement:

- to 5 (default value), the measurement range extends from -10V to +10V

- at 4, the measurement range extends from -5V to +5V, which also improves accuracy.

Communication configuration (COM1 and COM2)

- Parameters 400 and 420 define the operating mode of COM1 and 2 ports:
0=disabled, 1=Slave, 2=Master, 3=DMX mode (see Basic interpreter manual)

- Parameters 401 to 404 and 421 to 424 are great classics for setting serial communication ports (Baud rate, number of data bits, parity, stop bits)

- Parameters 405 and 425 define the card ID when configured as a slave (see parameters 400 and 420)

- Parameters 408 and 426 define the minimum delay in milliseconds between the sending of 2 Modbus frames (some drives for example require at least 5 ms between 2 frames received)

- Parameters 409 and 427 define the number of unsuccessful attempts before returning a

transmission error (visible in "All PLC Variables", Modbus RTU master error (or success) registers.

State of outputs 0 to 31 at boot → Register 270.

This is a 32-bit register. This parameter is used to define (force) the state of the digital outputs when the card is powered up (reminder: outputs 0 to 15 are physical outputs, outputs 16 to 31 are virtual outputs (for example for a module external)).

The state of the analog outputs (AOUT0 and AOUT1) on loss of the ENable → Registers 330 and 331.

These outputs will be forced to the voltages corresponding to these Values, when the link of the ENA is interrupted (case of emergency stop).

The "PLC Basic AutoRUN" parameter → Parameter 510

Value: 0 or 1.

If set to 1, the PLCBasic program present in the card's EEprom will run automatically on power-up.

Network Configuration settings → Settings 520 to 544

Here you can give a Netbios name to the card on your network, choose the DHCP mode (to 1), or define the IP address of your card, subnet mask, gateway, port, etc...

Recipes

Theory of recipes

The creation of recipes allows you to plan for your program, different "sets" of parameters that can be used according to the type of production to be carried out (variants).

Example: parameters for labeling objects of different sizes, parameters for applying labels of different sizes, etc...

The **Recipe Editor** ("Editeur Recettes") allows you to assign for each register:

→ a name, a type, a Value, a comment.

Editeur Recettes

Taille des recettes : 100

Page 0 Page 1 Page 2 Page 3

Numéro recette 0

#	Value	Name	Type	Comment
10000	28514	RCP_RECETTE_NOM	U16	Nom de la recette (10 caractères maxi)
10001	29813		U16	
10002	26981		U16	
10003	27756		U16	
10004	8293		U16	
10005	0	RCP_ONOFF_ORIENT_ENTREE	U16	0/1 , avec/sans detection spot 0=sans
10006	0	RCP_ONOFF_COIFFE	U16	
10007	0	RCP_ONOFF_SPOT_ETIQUETAGE	U16	
10008	0	RCP_ONOFF_ETIQUETTE	U16	
10009	0	RCP_ONOFF_CONTRE_ETIQUETTE	U16	
10010	0	RCP_ONOFF_MEDAILLON	U16	
10011	0	RCP_ONOFF_COLLERETTE	U16	
10012	0	RCP_ONOFF_TABLE_ENTREE	U16	
10013	0	RCP_ONOFF_ETOILE	U16	
10014	0	RCP_ONOFF_DISTRIBUTEUR	U16	
10015	0	RCP_ONOFF_MAGASIN	U16	
10016	0	RCP_ONOFF_MARQUEUR	U16	
10017	65535		U16	
10018	2652	RCP_R_ORIENT_ETIQ	FLOAT	pulses/tr 0 decimale , resolution orienteur poste...
10019	---		U16	
10020	16,2	RCP_R_PUL_MM_COLLERETTE	FLOAT	pulses/mm 1 decimale , resolution en pulses/mm...
10021	---		U16	
10022	9,5	RCP_R_PUL_MM_ETIQUETTE	FLOAT	pulses/mm 1 decimale , resolution en pulses/mm...
10023	---		U16	
10024	1	RCP_R_PUL_MM_MEDAILLON	FLOAT	resolution en pulses/mm au niveau du medaillon
10025	---		U16	
10026	138	RCP_CIRCONF_COLLERETTE	U16	mm 0 decimale , circonference bouteille au ni...
10027	276	RCP_CIRCONF_ETIQUETTE	U16	mm 0 decimale , circonference au niveau etiq...
10028	65535		U16	
10029	5	RCP_AV_COLLERETTE	U16	mm 0 decimale , avance collerette

How to use

First of all, you need to define the number of parameters you will need for your recipes. NB: Each parameter occupies by default one 16-bit register (U16).

For each register, you must specify its type (U16, U32, I16, I32, FLOAT).

In the case of a 32-bit register (DWORD or FLOAT), the following 16-bit register will be automatically reserved, and its Value field will then become non-editable since the 32-bit Value will appear on the previous line.

The memory area dedicated to recipe storage (EEPROM) is a single space of 16 KB (i.e. 8192 16-bit registers).

The size of a recipe is limited to 1000 registers.

You can therefore, for example, manage more than 40 recipes of 200 registers (8192 /200

= 20+) or even more than 100 recipes of 80 registers (8192 / 80 = 100+).
 You have 4 pages (0 to 3) allowing you to work on 4 recipes simultaneously.
 For each page, you have **an Index** ("Numéro recette") allowing you to select the active recipe.

How it works:

Read :

Depending on the recipe number entered in the Index of a page, the system will transparently address/display the block of data corresponding to the recipe at the start addresses of the page: 10000, 11000, 12000 or 13000.
 Thus, your registers (we advise you by convention to name them RCP_...) will take different Values in turn, just by changing the Value of the Index.

Write : The Values written at these first addresses will (always transparently) be effectively stored in the memory area pointed to by the Index ("Numéro recette"=recipe number).

Advantage with an HMI :

From a screen dedicated to entering recipe parameters, your enterable fields are always read and written to the same Modbus addresses, and by changing only the Value of the Index they will actually be stored in/read from the memory area assigned to the recipe number.

PLC_variables

On the **View** tab -> **All PLC variables ("Toutes variables PLC")**

This page allows you to access and visualize in real time all the system registers, in the form of a table and with search and sorting possibilities.

It is possible to display these logs or not, depending on their type and access method (check):

Input Registers (read-only), **Input Bits** (read-only bits)

Holding Registers (read/write), and **Coils** (bits read/write).

It can also be convenient to open "All PLC variables" several times (= several instances), in order to access registers separately according to their type.

For a search, you can enter a name, a keyword, an address, a Value, a format, etc... You can also refine by limiting the search to a type of register and/or a column.

You can of course enter Values directly in the Holding Registers and Coils, from the table (Value column).

Display

Input Registers Input Bits

Holding Registers Coils

Search: Name analog

Section	Address	Name	Value	Comment	Format
Buffer commandes Modbus	2141	CMD_Buffer 2141	0	CMD[141]	U16
Buffer commandes Modbus	2142	CMD_Buffer 2142	0	CMD[142]	U16
Buffer commandes Modbus	2143	CMD_Buffer 2143	0	CMD[143]	U16
Buffer commandes Modbus	2144	CMD_Buffer 2144	0	CMD[144]	U16
Buffer commandes Modbus	2145	CMD_Buffer 2145	0	CMD[145]	U16
Buffer commandes Modbus	2146	CMD_Buffer 2146	0	CMD[146]	U16
Buffer commandes Modbus	2147	CMD_Buffer 2147	0	CMD[147]	U16
Buffer commandes Modbus	2148	CMD_Buffer 2148	0	CMD[148]	U16
Buffer commandes Modbus	2149	CMD_Buffer 2149	0	CMD[149]	U16
Analog outputs	2150	Analog output 0	0	Analog OUT[0]	I16
Analog outputs	2151	Analog output 1	0	Analog OUT[1]	I16
Analog outputs	2152	Analog output 2	0	Analog OUT[2]	I16
Analog outputs	2153	Analog output 3	0	Analog OUT[3]	I16
Analog outputs	2154	Analog output 4	0	Analog OUT[4]	I16
Analog outputs	2155	Analog output 5	0	Analog OUT[5]	I16
Analog outputs	2156	Analog output 6	0	Analog OUT[6]	I16
Analog outputs	2157	Analog output 7	0	Analog OUT[7]	I16
Digital outputs words	2160	Digital output word 0	0	DOUT as word	U16
Digital outputs words	2161	Digital output word 1	0	DOUT as word	U16
Digital outputs words	2162	Digital output word 2	0	DOUT as word	U16
Digital outputs words	2163	Digital output word 3	0	DOUT as word	U16
Digital outputs words	2164	Digital output word 4	0	DOUT as word	U16
Digital outputs words	2165	Digital output word 5	0	DOUT as word	U16


The main interest is therefore to be able to access the content of system bits and registers as easily as possible, in order for example to test the proper functioning of your Basic program, and its development.

(Without this, it would be necessary to stop the running program and launch a Print from the command line, having first searched for the address of the register in question...)

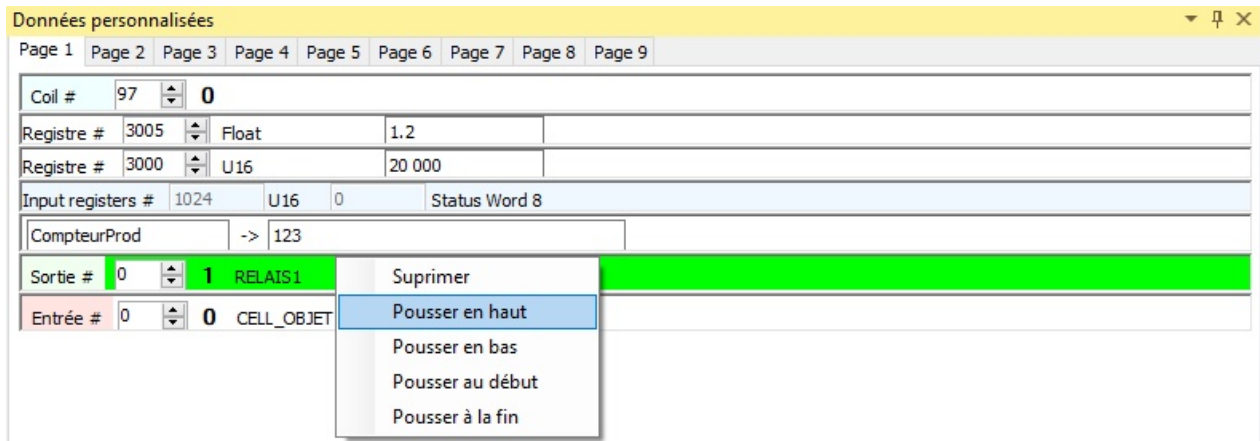
Custom_Data

To debug a PLC program under development, ICNCStudio offers a tool allowing you to make an "à la carte" selection of registers, bits (Coils, Inputs, or Outputs), PLC variables and program variables whose evolution you wish to follow running.

This is the **list of Custom Data**.

This is available from the **View** tab -> **Custom data**, or from the  icon

The Custom Data window will appear by default at the bottom-left of the screen, but can be dragged and dropped to any other area as desired.



The Custom Data window has 9 separate pages (1 to 9), which can be made up of registers, bits and variables inserted as you wish. This is possible from the **User Registers, Saved Registers, DIN, DOUT, COIL, Recipe Editor, and All PLC Variables** windows.

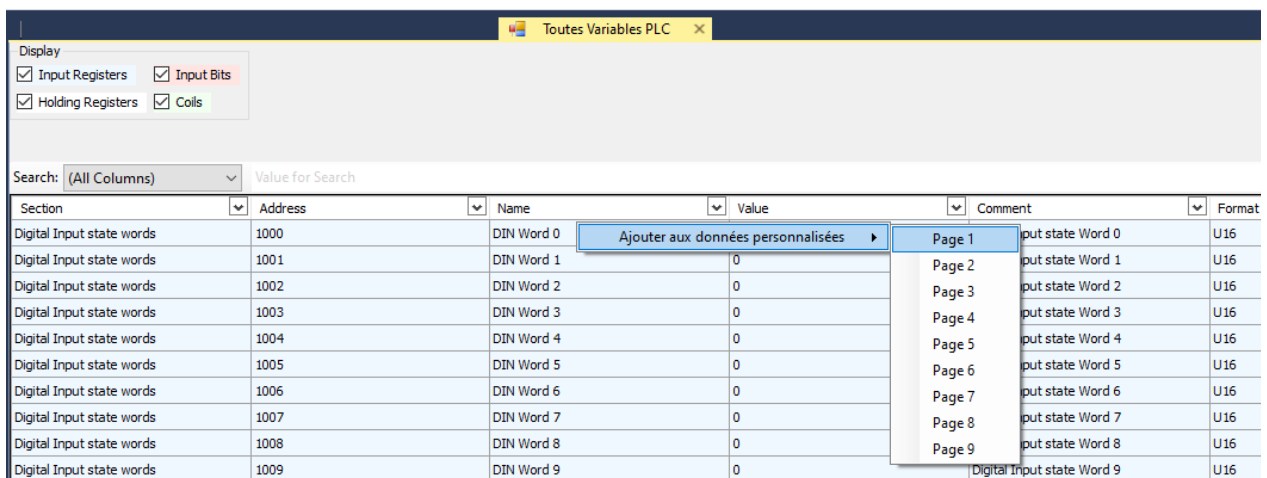
The advantage of having different pages can be for example to have distinct lists of registers, bits, and variables, according to the current PLC cycle, according to the current operating mode, according to the function (Ethernet, Modbus, etc. ...)

Each of these 9 pages can be renamed (**right click** on the tab -> **Rename Tab**).

The list of these data is composed in the order of additions. However, it is possible to change this order: right click, then

- > Delete
- > Push up
- > Push down
- > Push to start
- > Push to end

Adding a register, bit, or variable to one of the 9 pages is done by: **right click -> Add to custom data -> Page x** from the relevant register line, bit or PLC variable.



The values of these registers, bits and variables can be changed in real time directly from the Custom Data window.

For Coils and DOUT outputs, the boolean value can be changed directly by double-clicking

on the line concerned, in the Custom Data window page.

Charts

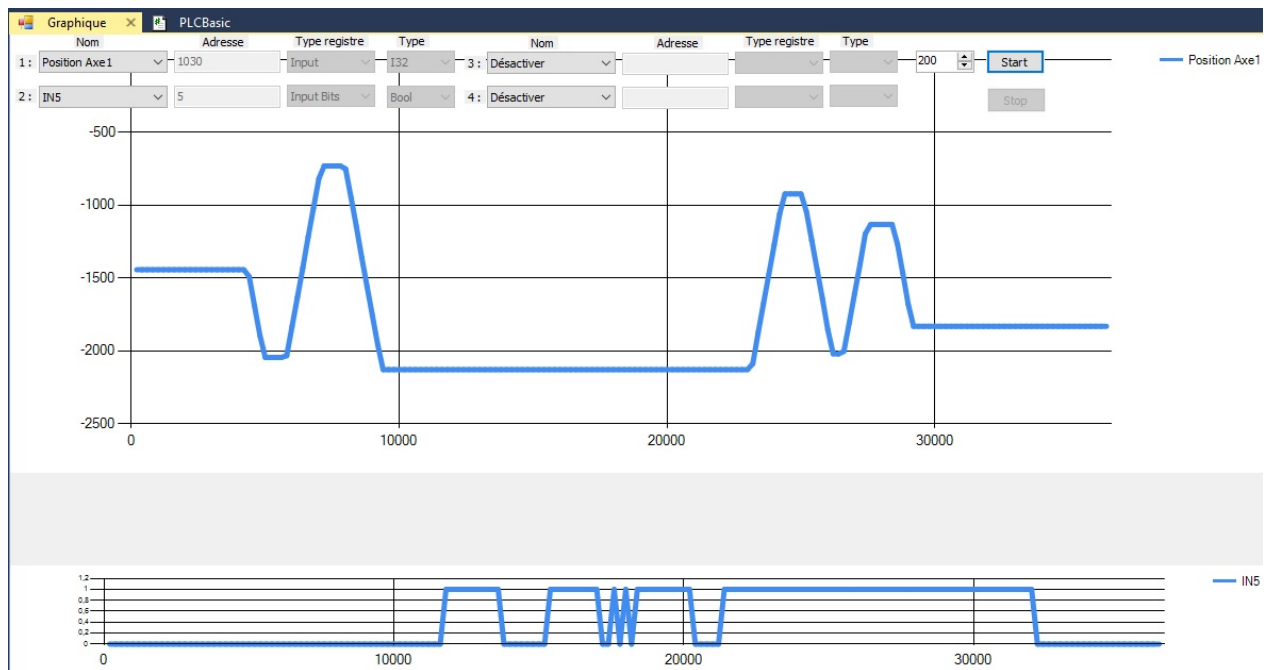
This window ("Graphique"=chart) allows you to monitor, on a time scale, the evolution of up to 4 values, which can be:

- Current position of **one or more Axes**, or speed, or target position
- State of digital inputs or outputs (Boolean)

The choice can be made using the preselections in the drop-down menus, or by selecting "Custom" which allows you to monitor the contents of a register, specifying its address, its Type, and its Format

Example: Address 3010 (Ram), Holding Register, U16

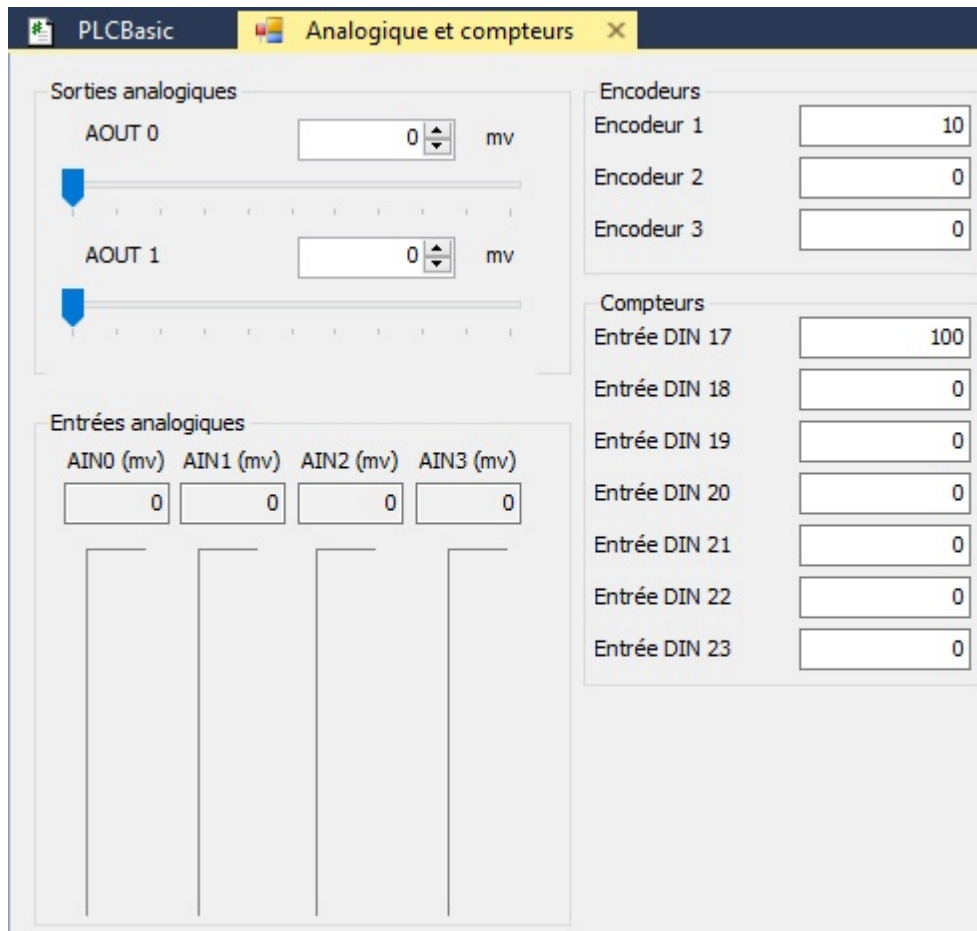
The refresh delay is adjustable in milliseconds. Set to 200ms here:



The axes will always appear on the same graph. The scale is self-adjusting over time.

The Booleans will always appear on a different graph than the axes, for a question of scale.

Analog_Counters



This window allows instant reading of the 4 analog inputs, as well as manual adjustment of the voltage of the 2 analog outputs (via slider, or by entering the desired value).

It also allows reading of quick entries, for:

- the positions of the 3 encoders
- counter values

Firmware_update

Firmware update of the InterpCNC V3 board

Definition

Firmware is the board's embedded system software. It allows the correct operation of all its components, between them and in their optimal implementation. He is also responsible for all the functionalities offered by the card.

It is advisable to update the firmware whenever a new version is available. This allows your applications to benefit from the latest fixes or performance improvements, as well as any new features developed.

NB: if you have used the InterpCNC board for a project that is not likely to evolve and is already working perfectly as is, an update is not recommended.

Firmware update is on demand:
→ Tools Tab / Firmware Update

Note: Prior to detect if a new firmware version is available, ICNCStudio must be up to date.

Of course, ICNCStudio must first be connected to the card.
The Firmware update window informs you of the current version.
The selection of the version is done just below (button [...] to browse).
The [Release Notes] button allows you to view the release notes of the selected and previous versions.

Update procedure

Warning: Never interrupt the +24V power supply to the card, nor unplug its network or USB connection during an update

(Despite the software security in place, there could still be a risk of incomplete programming, which would have the effect of rendering the card non-functional, and would then require it to be returned to the workshop for repairs)

Once the version is selected, click on [Submit]:
→ the procedure starts (progress bars) and the new firmware is transferred to the board.
Then click on [Reboot and Update]
→ the firmware is installed (see progress on the card display).
At 100%, the update is complete and the board is operational.

ICNCStudio_update

ICNCStudio Update

ICNCStudio has an automatic online update system.
Periodically, when starting the application, the version of ICNCStudio installed compares its version number with the latest available, and will inform you if an update is available. You can choose whether or not to install it. If so, the update will then be downloaded and installed automatically (follow the instructions on the screen).

You can also self-initiate an update search:
→ Tools tab: ICNC Studio update

The version in use can be viewed in the Help/About ICNCStudio tab.

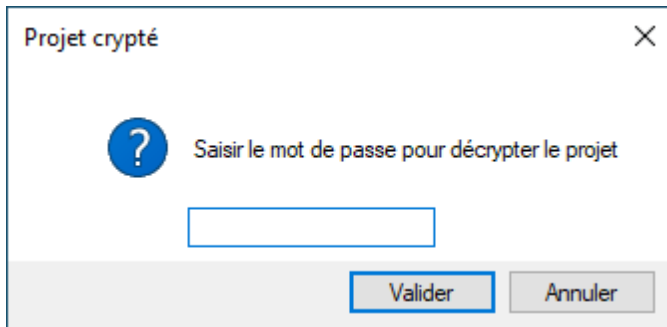
Encryption

Beyond password protection, and always in order to protect your PLC Basic program against counterfeiting, ICNCStudio also includes an **encryption system**.

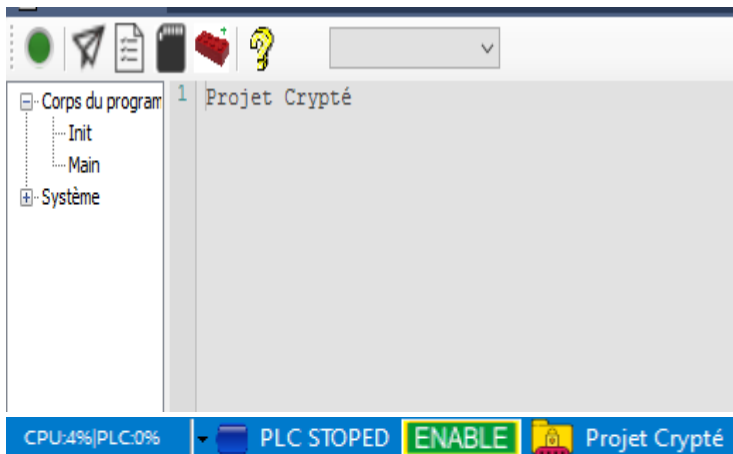
Thus an advanced user who could have obtained the source file of your PLC program, if you have encrypted it, will not be able to see its content in ICNCStudio, or even recognize

portions of code with a text editor.

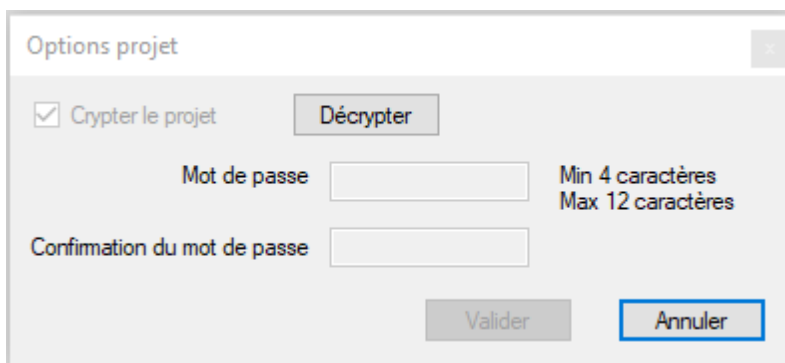
If you open an encrypted program, this input window appears:



If you cancel password entry, the program is loaded, but the text editor appears as follows:



Clicking on the yellow/red icon displays the decryption password entry window:



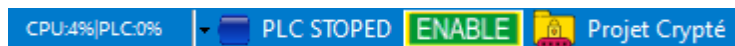
Clicking "Decrypt" (= "Décrypter") will bring back the initial password entry window. The content of the program then becomes fully accessible:



Clicking on the icon that has become gray, encrypts the unencrypted project:

-> Check "Encrypt the project" ("Crypter le projet"), choose and confirm the password, then Validate.

You can then still view the program you encrypted, but if you save it it will be in encrypted form:



If you click again on the yellow/red icon, the "Reset" ("Réinitialiser") button allows you to change the current password:

Parameters

This chapter describes the different configuration screens accessible from the parameters icon.



Unlike the table accessible from **View -> InterpCNC Parameters** (Paramètres InterpCNC) only the main parameters are offered, but the readability and selection of settings is facilitated by drop-down menus, input fields and "check boxes".

General_configuration

The locking function

Protects your embedded PLC Basic program against reading and/or overwriting by setting a password.

Thus, it is impossible to clone the machine for which you have designed the PLC program, or even users who are too curious to delete it inadvertently.

Lecture Protégée (Read protect): The program cannot be read, neither from the Ram, nor from the Eprom.

Écriture protégée (Write protect): A program can be sent in Ram and be executed (Run), but cannot be written in the Eprom (overwriting of the program already in place is therefore not authorized).

For more details, see the explanations in the Configurations chapter.

The « PLC program AutoRUN » parameter

Value : 0 or 1.

If set to 1, the PLCBasic program present in the card's EEprom will run automatically on power-up.

Real Time Clock (= RTC) "Horloge temps réel"

The InterpCNC V3 board does not have a battery or internal battery, it can set the time automatically (on power-up, or on command), by connecting to an SNTP server.

Note: SNTP = "Simple Network Time Protocol"

In this case activate "Synchro on SNTP server". Your time zone and "Automatic summer/winter time" are the parameters used to adjust the time setting to your geographical position.

Rotation 180°

Allows easier reading of the LCD screen depending on the mounting direction of the card on a DIN rail.

Sortie PUL/DIR en TTL (PUL/DIR as TTL outputs)

Allows the PLC, when the axis commands are not used, to use the pulse and direction outputs as 0-5V discrete outputs, which will be remapped as outputs n° 16 to 27.

Axes

The initial frequency of Axes movements (Fréquence initiale des mouvements)

Also called "Frequency Start", this parameter reduces the time required for the acceleration phase (without increasing its speed) by not starting at frequency 0.

Please note, however, that some applications may not support certain settings (bottles on conveyor belt, etc.)

Limit switch (Fin de course)

This involves declaring here the input used for each limit switch, according to the direction of the axis, as well as its polarity.

The type of action on the limit switch can be the rapid stop of the axis (taking into account the Value of the Rapid Deceleration field) or its immediate stop.

This stop can also relate to all axes.

DIN

Configuration of standard inputs 0 to 15

– Input Polarity

This setting can, for example, allow you to use an input signal level opposite to that

provided in the program, without having to modify the existing program.
 On 0 (normal mode) → a low level gives a reading of 0, a high level gives a reading of 1
 On 1 → a high level gives a reading of 0, a low level gives a reading of 1

– **Debounce filter setting for each input (time in ms)**

1) Filter delay →

4 setting values are possible, i.e. 2 bits per input.

00: 0 ms delay → Filter off

01: 10ms delay

10: 30ms delay

11: 100ms delay

Thus any edge variation during this delay will be ignored.

2) Filter mode

These settings are based on the debounce time selected above.

3 Setting Values are possible, i.e. 2 bits per input:

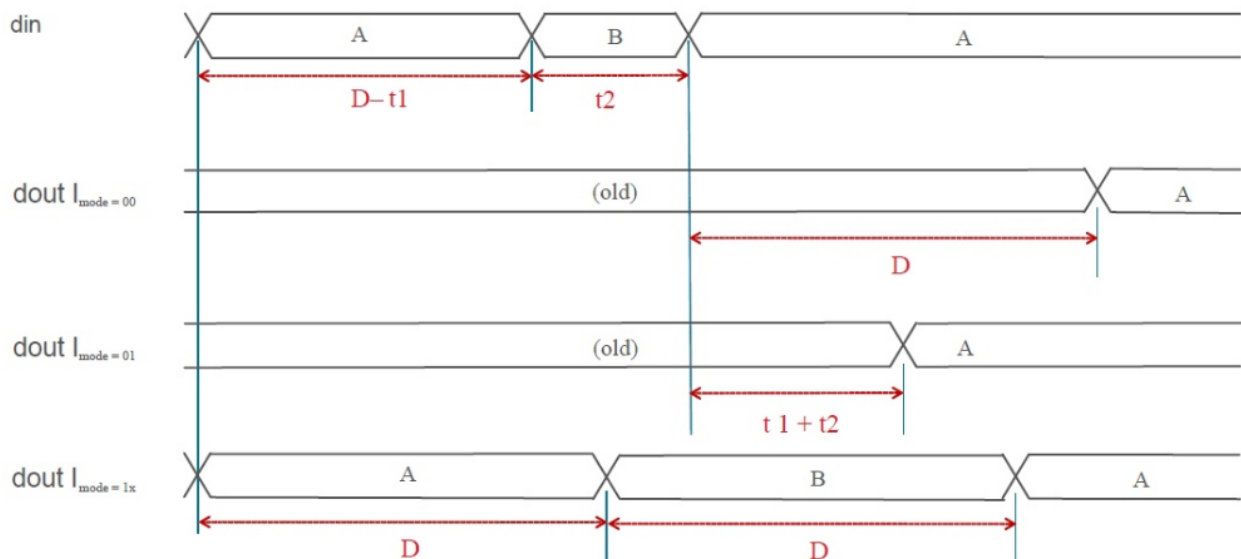


Figure 2.7. Debounce Filter Modes Timing Diagram

00: No filter. This mode only reports state changes whose duration is greater than X ms (as defined by the debounce).

01: Low Pass Filter. This mode only reports state changes that last longer than: (time missing to go to X ms + bounce duration).

10 or 11: This mode samples every X ms (as defined by the debounce), and reports the state detected at that instant.

Configuration of Inputs 16 to 22 (the Fast Inputs)

Entrées avancées DIN16 à DIN22

Entrée # 16

Polarité inversée (contact type NC) Standard Interruption Compteur

Entrée # 17

Polarité inversée (contact type NC) Standard Interruption Compteur Codeur 2X Codeur 4X

These 7 inputs are configurable as follows:

- 0: refresh on each line of the program (Standard).
- 1: refresh on each interrupt accessing this input.
- 2: counter mode
- 3: 2X encoder mode
- 4: 4X encoder mode

The polarity of each of these inputs can also be reversed here.

Ditto for virtual inputs, which can be:

- or analog inputs used as discrete inputs and remapped as inputs numbered from 23 to 26
- either entries

Entrées analogiques DIN23 à DIN26

DIN23 - Polarité inversée (contact type NC) DIN25 - Polarité inversée (contact type NC)

DIN24 - Polarité inversée (contact type NC) DIN26 - Polarité inversée (contact type NC)

Polarité entrées virtuelles

<input type="checkbox"/>	32	<input type="checkbox"/>	48	<input type="checkbox"/>	64	<input type="checkbox"/>	80		
<input type="checkbox"/>	33	<input type="checkbox"/>	49	<input type="checkbox"/>	65	<input type="checkbox"/>	81		
<input type="checkbox"/>	34	<input type="checkbox"/>	50	<input type="checkbox"/>	66	<input type="checkbox"/>	82		
<input type="checkbox"/>	35	<input type="checkbox"/>	51	<input type="checkbox"/>	67	<input type="checkbox"/>	83		
<input type="checkbox"/>	36	<input type="checkbox"/>	52	<input type="checkbox"/>	68	<input type="checkbox"/>	84		
<input type="checkbox"/>	37	<input type="checkbox"/>	53	<input type="checkbox"/>	69	<input type="checkbox"/>	85		
<input type="checkbox"/>	38	<input type="checkbox"/>	54	<input type="checkbox"/>	70	<input type="checkbox"/>	86		
<input type="checkbox"/>	39	<input type="checkbox"/>	55	<input type="checkbox"/>	71	<input type="checkbox"/>	87		
<input type="checkbox"/>	40	<input type="checkbox"/>	56	<input type="checkbox"/>	72	<input type="checkbox"/>	88		
<input type="checkbox"/>	41	<input type="checkbox"/>	57	<input type="checkbox"/>	73	<input type="checkbox"/>	89		
<input type="checkbox"/>	42	<input type="checkbox"/>	58	<input type="checkbox"/>	74	<input type="checkbox"/>	90		
<input type="checkbox"/>	27	<input type="checkbox"/>	43	<input type="checkbox"/>	59	<input type="checkbox"/>	75	<input type="checkbox"/>	91
<input type="checkbox"/>	28	<input type="checkbox"/>	44	<input type="checkbox"/>	60	<input type="checkbox"/>	76	<input type="checkbox"/>	92
<input type="checkbox"/>	29	<input type="checkbox"/>	45	<input type="checkbox"/>	61	<input type="checkbox"/>	77	<input type="checkbox"/>	93
<input type="checkbox"/>	30	<input type="checkbox"/>	46	<input type="checkbox"/>	62	<input type="checkbox"/>	78	<input type="checkbox"/>	94
<input type="checkbox"/>	31	<input type="checkbox"/>	47	<input type="checkbox"/>	63	<input type="checkbox"/>	79	<input type="checkbox"/>	95

AIN

AIN #3

Plage de mesure Gain Offset

Filtre pass bas (s)

Réglage niveaux pour états logiques (DIN23 à DIN26) des entrées analogiques

Niveau haut en mV Niveau bas en mV

– Analog input configuration

The gain: Setting to 1, therefore directly the resolution of the converter.

The default 4.885198 is an adjustment for input resistances.

The offset: allows you to set the value 0 (because you can measure from 0V to +10V). The measurement scale ("Plage de mesure"):

- to 5 (default value), the measurement range extends from 0V to +10V

- at 4, the measurement range extends from 0V to +5V, which also improves accuracy.

IN_ENA

In the **Parameters menu**, the **ENABLE/DISABLE input configuration page** gives access to configuration of the state of outputs 0 to 31, which will be forced according to the physical state (0 or 1) of the **ENA** (Enable) input.

So you can force the state for selected outputs to 0 or 1, according to the actual ENable input state (0 or 1).

NB: Just avoid contradictory selections (activate and deactivate at the same time for the same state, which are not managed).

– State of outputs 0 to 31 at boot (Register 270) "Etat des sorties au boot"

This is a 32-bit register. This parameter is used to define (force) the state of the digital outputs when the card is powered up

(reminder: outputs 0 to 15 are physical outputs, outputs 16 to 31 are virtual outputs (e.g. for an external module)).

Etat des sorties au boot

<input type="checkbox"/> 0	<input type="checkbox"/> 1	<input type="checkbox"/> 2	<input type="checkbox"/> 3	<input type="checkbox"/> 4	<input type="checkbox"/> 5	<input type="checkbox"/> 6	<input type="checkbox"/> 7
<input type="checkbox"/> 8	<input type="checkbox"/> 9	<input type="checkbox"/> 10	<input type="checkbox"/> 11	<input type="checkbox"/> 12	<input type="checkbox"/> 13	<input type="checkbox"/> 14	<input type="checkbox"/> 15
<input type="checkbox"/> 16	<input type="checkbox"/> 17	<input type="checkbox"/> 18	<input type="checkbox"/> 19	<input type="checkbox"/> 20	<input type="checkbox"/> 21	<input type="checkbox"/> 22	<input type="checkbox"/> 23
<input type="checkbox"/> 24	<input type="checkbox"/> 25	<input type="checkbox"/> 26	<input type="checkbox"/> 27	<input type="checkbox"/> 28	<input type="checkbox"/> 29	<input type="checkbox"/> 30	<input type="checkbox"/> 31

Sortie à activer lorque ENA passe à 1

<input type="checkbox"/> 0	<input type="checkbox"/> 1	<input type="checkbox"/> 2	<input type="checkbox"/> 3	<input type="checkbox"/> 4	<input type="checkbox"/> 5	<input type="checkbox"/> 6	<input type="checkbox"/> 7
<input type="checkbox"/> 8	<input type="checkbox"/> 9	<input type="checkbox"/> 10	<input type="checkbox"/> 11	<input type="checkbox"/> 12	<input type="checkbox"/> 13	<input type="checkbox"/> 14	<input type="checkbox"/> 15
<input type="checkbox"/> 16	<input type="checkbox"/> 17	<input type="checkbox"/> 18	<input type="checkbox"/> 19	<input type="checkbox"/> 20	<input type="checkbox"/> 21	<input type="checkbox"/> 22	<input type="checkbox"/> 23
<input type="checkbox"/> 24	<input type="checkbox"/> 25	<input type="checkbox"/> 26	<input type="checkbox"/> 27	<input type="checkbox"/> 28	<input type="checkbox"/> 29	<input type="checkbox"/> 30	<input type="checkbox"/> 31

Sortie à désactiver lorque ENA passe à 1

<input type="checkbox"/> 0	<input type="checkbox"/> 1	<input type="checkbox"/> 2	<input type="checkbox"/> 3	<input type="checkbox"/> 4	<input type="checkbox"/> 5	<input type="checkbox"/> 6	<input type="checkbox"/> 7
<input type="checkbox"/> 8	<input type="checkbox"/> 9	<input type="checkbox"/> 10	<input type="checkbox"/> 11	<input type="checkbox"/> 12	<input type="checkbox"/> 13	<input type="checkbox"/> 14	<input type="checkbox"/> 15
<input type="checkbox"/> 16	<input type="checkbox"/> 17	<input type="checkbox"/> 18	<input type="checkbox"/> 19	<input type="checkbox"/> 20	<input type="checkbox"/> 21	<input type="checkbox"/> 22	<input type="checkbox"/> 23
<input type="checkbox"/> 24	<input type="checkbox"/> 25	<input type="checkbox"/> 26	<input type="checkbox"/> 27	<input type="checkbox"/> 28	<input type="checkbox"/> 29	<input type="checkbox"/> 30	<input type="checkbox"/> 31

Sortie à activer lorque ENA passe à 0

<input type="checkbox"/> 0	<input type="checkbox"/> 1	<input type="checkbox"/> 2	<input type="checkbox"/> 3	<input type="checkbox"/> 4	<input type="checkbox"/> 5	<input type="checkbox"/> 6	<input type="checkbox"/> 7
<input type="checkbox"/> 8	<input type="checkbox"/> 9	<input type="checkbox"/> 10	<input type="checkbox"/> 11	<input type="checkbox"/> 12	<input type="checkbox"/> 13	<input type="checkbox"/> 14	<input type="checkbox"/> 15
<input type="checkbox"/> 16	<input type="checkbox"/> 17	<input type="checkbox"/> 18	<input type="checkbox"/> 19	<input type="checkbox"/> 20	<input type="checkbox"/> 21	<input type="checkbox"/> 22	<input type="checkbox"/> 23
<input type="checkbox"/> 24	<input type="checkbox"/> 25	<input type="checkbox"/> 26	<input type="checkbox"/> 27	<input type="checkbox"/> 28	<input type="checkbox"/> 29	<input type="checkbox"/> 30	<input type="checkbox"/> 31

Sortie à désactiver lorque ENA passe à 0

<input type="checkbox"/> 0	<input type="checkbox"/> 1	<input type="checkbox"/> 2	<input type="checkbox"/> 3	<input type="checkbox"/> 4	<input type="checkbox"/> 5	<input type="checkbox"/> 6	<input type="checkbox"/> 7
<input type="checkbox"/> 8	<input type="checkbox"/> 9	<input type="checkbox"/> 10	<input type="checkbox"/> 11	<input type="checkbox"/> 12	<input type="checkbox"/> 13	<input type="checkbox"/> 14	<input type="checkbox"/> 15
<input type="checkbox"/> 16	<input type="checkbox"/> 17	<input type="checkbox"/> 18	<input type="checkbox"/> 19	<input type="checkbox"/> 20	<input type="checkbox"/> 21	<input type="checkbox"/> 22	<input type="checkbox"/> 23
<input type="checkbox"/> 24	<input type="checkbox"/> 25	<input type="checkbox"/> 26	<input type="checkbox"/> 27	<input type="checkbox"/> 28	<input type="checkbox"/> 29	<input type="checkbox"/> 30	<input type="checkbox"/> 31

AOUT sur perte de ENA

Etat de la sortie AOUT0 en mV (0 à 10000)

Etat de la sortie AOUT1 en mV (0 à 10000)

- **The state of the analog outputs (AOUT0 and AOUT1) on loss of the ENable (Registers 330 and 331).**

These outputs will be forced to the voltages corresponding to these Values, when the link of the ENA is interrupted (case of emergency stop).

Serial_port

Port COM1

Mode :

Communication settings

Baud rate :

Data bits :

Parity :

Stop bit :

Modbus Slave settings

Slave ID :

Modbus master settings

Inter frame delay (ms) :

Max retry :

DMX Device Settings

Base address :

Chanel used :

Port COM2

Mode :

Communication settings

Baud rate :

Data bits :

Parity :

Stop bit :

Modbus Slave settings

Slave ID :

Modbus master settings

Inter frame delay (ms) :

Max retry :

DMX Master settings

Chanel used (transmitted) :

Communication configuration (COM1 and COM2)

- These parameters (400 and 420) define the operating mode of the COM1 and 2 ports: 0=disabled, 1=Slave, 2=Master, 3=DMX mode (Slave for COM1, Master for COM2). See also the Basic Interpreter manual.

- The **Communication settings** (401 to 404 and 421 to 424) are great classics for setting serial communication ports (Baud rate, number of data bits, parity, stop bits)

- The **Slave ID** parameters (405 and 425) define the card ID when configured as a slave (see parameters 400 and 420)

- The **Inter frame delay** parameters (408 and 426) define the minimum delay in milliseconds between the sending of 2 Modbus frames (some drives for example require at least 5 ms between 2 received frames)

- The **Max retry** parameters (409 and 427) define the number of unsuccessful attempts before returning a transmission error (visible in "All PLC Variables" -> Modbus RTU master error (or success) registers).

Polling

The InterpCNC card as master, can communicate via Modbus with other peripherals (drives, HMI, PLC, ...) by sending read or write requests to a slave peripheral.

Polling, also called mapping (or mapping in French) consists of creating a self-updating copy of a data zone, to another data zone.

Thus, for example, bits or registers can be written locally, and these Values will be automatically sent every X ms to update the target registers of the slave device. The same in reading: bits and registers of the slave can be periodically copied every X ms to registers or local bits of the master.

Master COM port :	Slave modbus ID	Polling period (ms)
COM 1	0	50
Action		
Read slave coils and set local digital outputs or coils		
Data address on slave	Nombre de données	Local data address
0	1	0

To do this, specify the **COM port** to which the slave is connected, as well as its **Modbus ID** and the desired frame refresh rate (in ms).

The **Action** can relate to the reading or writing of bits or registers, located on the slave at the address of the **Data address on slave** field, for the **Number of successive data**, and will be mapped locally on the master (the card InterpCNC) from the address of the **Local data address** field.

Ethernet

Reseau

Attribution IP Configuration manuelle Discovery port (UDP)

Nom NETBIOS

Configuration IP

Adresse IP	Masque reseau	Passerelle
<input type="text" value="192.168.10.12"/>	<input type="text" value="255.255.255.0"/>	<input type="text" value="192.168.10.254"/>

Configuration Modbus

TCP server port	UDP server port
<input type="text" value="502"/>	<input type="text" value="500"/>

- **Network Configuration parameters** (Parameters 520 to 544)

Here you can give a Netbios name to the card on your network, choose the DHCP mode (to 1), or define the IP address of your card, subnet mask, gateway, port, etc...

Digital_control

Configuration générale axes CNC

Durée des pulses (μ s)

Ecart de jonction (mm)

Tolérance Arc (mm)

Axe 1

Inverser sens de rotation Pulses actifs à 1

Résolution axe (pulses/mm)	<input type="text" value="250"/>	Vitesse maximale (mm/mn)	<input type="text" value="500"/>
Course maximale (mm)	<input type="text" value="200"/>	Accélération (mm/s ²)	<input type="text" value="250"/>

These parameters relate to the advanced configuration of pulse and direction signals in digital control mode, for each of the axes.

The **GALAAD** software will modify some of them with the settings that will be made.

The user is advised not to modify these values.

Plasma_thc

The IntercpCNC V3 card has been made compatible with **GALAAD** (<https://www.galaad.net>), as a plasma cutting and machining software.

The torch is normally mounted on the Z axis. **THC** stands for: Torch Height Control. THC is a system handled by the firmware to manage optimal positioning of the Z axis during plasma cutting, according to speed and arc voltage.

2

Paramètres généraux

Mode fonctionnement THC ARC OK sur entrée DIN. Régulation sur tension d'arc ▾

Délai entre activation THC et début d'opération (ms)

Mesure tension d'arc

Source mesure tension Entrée AIN 0 ▾ Registre personnalisé

Tension ARC réelle pour entrée AINx à 0V (offset) (mV)

Tension ARC réelle pour entrée AINx à 10.0V (mV)

Note : Le réglage du filtre sur l'entrée analogique utilisée permet de régler la stabilité de la mesure.

Limites de fonctionnement

Montée axe Z maximale (mm) Tension Maxi. échantillonnage (mV)

Descente axe Z maximale (mm) Tension Mini. échantillonnage (mV)

Seuil vitesse engagement effectif THC (% de la vitesse de coupe)

Seuil désengagement THC (% de la vitesse de coupe)

Régulation

Gain proportionnel Vitesse

Zone morte de régulation (fenêtre de tolérance) (mV)

"**Délai entre activation THC et début d'opération**" is a duration in ms to delay activation of THC.

Plasma inverters generally have an internal divider so that arc voltage can be measured by an analog input of the InterpCNC card. So some inverters have an output which varies from 0V to 10V, or 0V to 5V, or 1V to 4V, or other, according to brands or model.

Of course, arc voltage measurement is more accurate over the full 0V to 10V range. This is what the optional **SOPROLEC THC amplifïer/isolator** module is made for. Please contact us if you need to purchase one.

"**Source mesure tension**" defines which Analog input of the card will be used for arc voltage measurement.

Then both "**Tension ARC réelle...**" parameters will inform of the inverter output range, or these need to be set to 0 and 10000 mV if you have the SOPROLEC THC module.

"**Montée axe Z max**" and "**Descente axe Z max**" will respectively define max ascent and max descent in mm for the torch, depending on material thickness.

THC needs to be deactivated when cutting speed is too slow (in curves for instance), and

reactivated when speed retrieves.

"**Seuil vitesse engagement effectif THC**" defines the % of the cutting speed to reactivate THC.

"**Seuil désengagement THC**" defines the % of the cutting speed to deactivate THC.

"**Zone morte de régulation**" is a deadzone in mV, set to ignore unwanted voltage fluctuation observed on the Analog input dedicated to arc voltage measurement.

Note: This settings are written to by the **GALAAD** software. Please read the THC section of the **GALAAD** manual for complete information.

Release_note

ICNCStudio Version: 1.0.0.87

Fixes:

- Fixed a bug on the "Analog, Counters" settings: modifications to the Encoder 2 and Encoder 3 fields were not taken into account
- Search and Replace function was no longer functional
 - Custom Data functionality:
- Fix on an unwanted color change when changing the state of Coils
- Authorization to write a Holding Register from the Custom Data table
- The text ****Modified**** no longer appears in the header of ICNCStudio after loading a PLC program

Developments:

Custom Data feature:

- When opening the Custom data window, automatic positioning at the bottom left
- Changing borders + aligning controls
- Added a Toggle ("Change state") button for outputs and coils
- Adding forcing of inputs ("Forcing to 1" button, and "Add forcing" button)
 - Search and Replace functionality:
- Redefinition and application of a new mode of operation

ICNCStudio Version: 1.0.0.86

Fixes:

- Resetting the position of the "search" tab following the start of ICNCstudio.
- Changed the layout of card protection settings.
- Eliminated text editor flashing after selecting a word in the "search" tab (if not necessary).
- Eliminated text editor blinking following selection of a sub-function or function in the tree (if not necessary).
- General improvement to how the custom chart works.
- Fixed bug detecting the version of the loaded project in ICNCstudio.
- Fixed the error displaying the encoder number in the "analog and counters" tab.
- Display of the encoder number following a change in the fast input parameters.
- Changed maximum filter values on analog inputs.
- Fixed the bug related to constantly changing the size of the user register array and saving registers.

- Correction when sending an Encoder parameter to the card (from the parameters table)
- Fix on reading registers U16 and I16 in custom graph display.

Developments:

- Added functions and commands for autocompletion in the text editor.
- Ability to reset a project password using a unique code for each project.
- Navigation in the search table using the arrows.
- Added Custom Data functionality, for debugging.

ICNC Studio Version: 1.0.0.85

Fixes:

- Bug when opening the search window in the PLC editor and searching for special characters.

Evolutions:

- Scroll disabled for recipe selection.

ICNC Studio Version: 1.0.0.84

Fixes:

- The cursor remains visible in the basic editor.
- Changed the display of the project name.
- Copy/Paste variable names from different tables without retrieving the whole row of the table.
- The size of the column of values in the tables of user registers and saved registers is stable, which means that a constant change in size with each new value is eliminated.

Evolutions:

- The table cells turn gray when the card is disconnected.
- To open the project, you can simply drag and drop the file into the text editor.
- Added new parameters for card locking (read/write protection of saved PLC program).
- Added new settings for the Linky interface.
- Continuous numbering of program lines between the different blocks.
- Checking for conflicts of identical variable names in different tables.
- Checking for conflicts of identical names of constants in the basic editor before sending the program to the board.
- Optimized error messages, added HTML text that links to the error block and line.
- New contextual help system for the basic editor (search for commands) and for the different sections of ICNCStudio by pressing the F1 key on the element or the text sought.
- Added encryption/decryption of projects.
- Detection of functions and subroutines (sub) in the tree structure.
- New search window in all blocks by pressing (Control + F) on the Basic editor or on the various tables.

ICNC Studio Version: 1.0.0.83

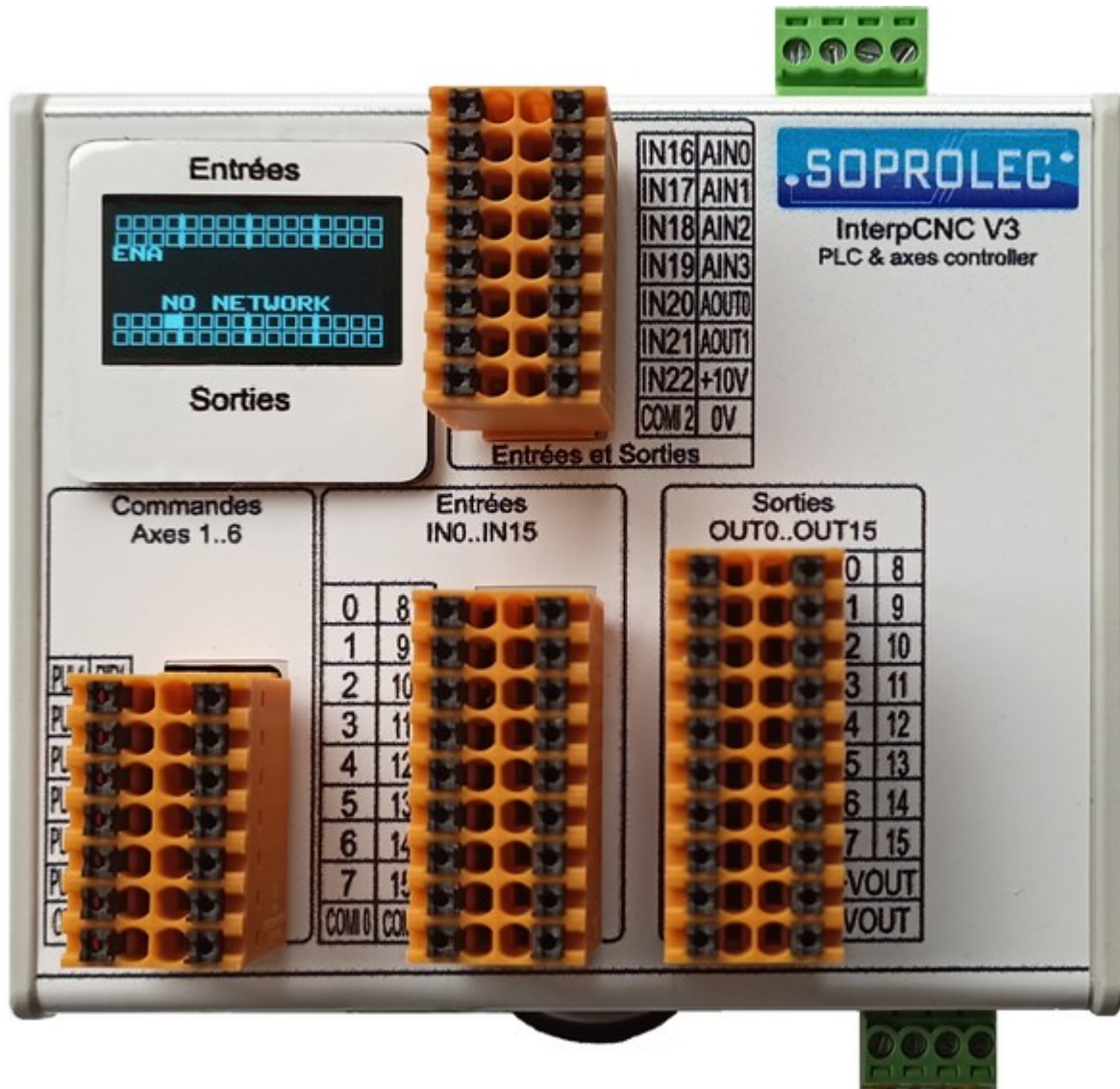
- Added block system.
- Addition of table of constants.

InterpCNC V3 manual

SOPROLEC
ZAC DE L'EPINE
72460 SAVIGNE L'EVEQUE
FRANCE
Tél : +33 (0)2 4376 4476



SOPROLEC InterpCNC V3 Axes card



Setup Manual

Presentation

The InterpCNC V3 card is a PLC card mainly intended for axis control. It has 6 axes command outputs that can be interpolated or independent.

Developed on the basis of a powerful 32-bit 480 mHz processor, the InterpCNC V3 also offers ideal performance for digital control (CNC) applications and also automation applications requiring high-performance axis control/command.

It has the advantage of providing maximum power, connectivity, and functionality, in a very compact case, and directly fixable on a DIN rail.

In addition, the InterpCNC has a powerful Basic language interpreter allowing standalone automation management. (See our PLCBasic Interpreter documentation).

Your project consists on the one hand of a Basic program, but also of the registers and

bits, inputs and outputs, which you will have declared and named in each of the tables. It is also made up of the recipes that you may have defined in the recipe editor. The set will be saved in a single file with the extension ".plc"

The axis control interface in Step/Direction mode is compatible with the entire range of motorizations offered by the SOPROLEC company (stepper motorization, brushless motorization).

Three communication interfaces are available:

USB: Virtual Com port (Modbus RTU protocol), for fast communication in CNC applications

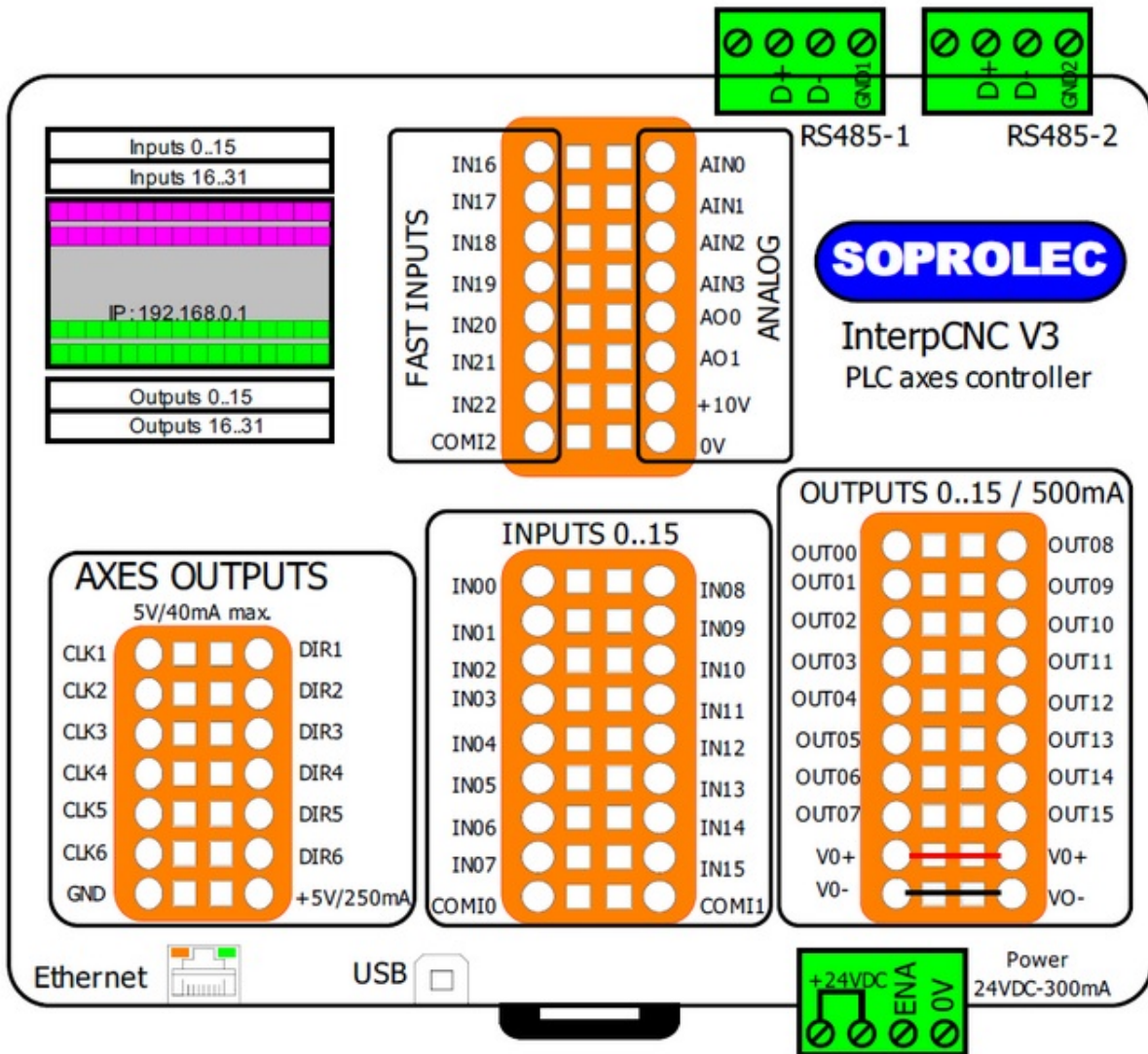
Serial: 2 RS485 ports (COM1 and COM2, Modbus RTU protocol) for industrial applications

Ethernet: Modbus TCP or Modbus UDP protocol.

The COM1 port is also compatible with the DMX512 protocol, widely used in the entertainment world for controlling equipment (see Basic Interpreter Manual).

A mini OLED screen allows you to view the status (0 or 1) of the first 32 inputs and the first 32 outputs of the card in real time, as well as various information (Firmware version on power-up, status of the ENable entry, board IP address, etc.)

Overview of the InterpCNC V3 board



Power :
24VDC/300mA power supply required by the card

Outputs specs :
Outputs 0 to 15: Opto-Isolated Outputs, 500mA max per output.
Caution: Do not consume 500mA on multiple outputs at the same time.
These outputs must be supplied externally on VOut+ and Vout-, voltage < 32V.
Outputs 16 to 96: Virtual outputs, require one or more output expanders via Modbus communication (Example: Kinco KS123). See implementation on page 28.
Outputs PUL1 to PUL6, and DIR1 to DIR6: TTL output 5V/40mA max

Inputs specs :
Digital inputs IN0 to IN15 are Opto-Isolated. They meet the **IEC 61131-2** standard and are type 3, i.e. the minimum current and voltage required to switch them to the high state are respectively between 2.27/2.45 mA, and 7.44/7.98 V for voltage.

NB: Inputs IN0 to IN15 can be filtered, i.e. an anti-bounce filter can be applied to them (4 possible delays in ms, and 3 filtering modes):
Example: case of a dry contactor (button or relay) which could generate rapid pulses by not establishing a frank and clean contact.

See Board Parameters table for settings (Parameters 210 to 213).

Inputs IN0 to IN15 : 0 to 24V

The COMI0 or COMI1 commons must be connected to 0V (PNP inputs) or +24V (NPN inputs).

Inputs IN16 to IN22 : TTL type fast counter inputs

These inputs are said to be fast, because they correspond to physical inputs directly managed by the micro-controller (unlike opto-isolated inputs 0 to 15 which are managed by a component using the SPI bus).

COMI3 must be connected to 0V (PNP) or +24V (NPN).

The **IN21 input** can be configured as an incremental encoder input (2X or 4X).

Inputs IN23 to IN255: Virtual inputs, require one or more input expander via Modbus communication (Example: Kinco KS123).

ENABLE input: Emergency stop function. 0 to 32V max. High level from 3.5V.

Analog Inputs/Outputs:

4 analog inputs AIN0 to AIN3: -10V to +10V.

The resolution is 16 bits (Values read from -32767 to +32767).

-32768: -10V

0: 0V

32767: +10V

NB: parameters 300 to 311 are used to calibrate the Values received on these 4 analog inputs. See parameter table.

2 Analog Outputs AOUT0 and AOUT1: 0 to 10V. 11-bit resolution (from 0 to 2047).

0: 0V

2047: +10V

Setup

Installation under Windows 7, 8, 10 and 11 does not require any particular driver.

In USB, the card is automatically recognized in Plug and Play as a "USB Serial Device" (Virtual COM Port).

In Ethernet, the connection is a classic network connection knowing that the card has its own IP address (re-definable), and that port 502 is generally used in TCP, or 500 in UDP.

It will be necessary to make sure that the firewall of your computer does not block the connection with the card, so that the ICNCStudio software can detect it and connect to it. If you connect the InterpCNC card to your network (via a router or switch), make sure that its IP address is on the same network as the PC on which ICNCStudio will be used (= same beginning of IP address, Example: 192.168 .10.xx)

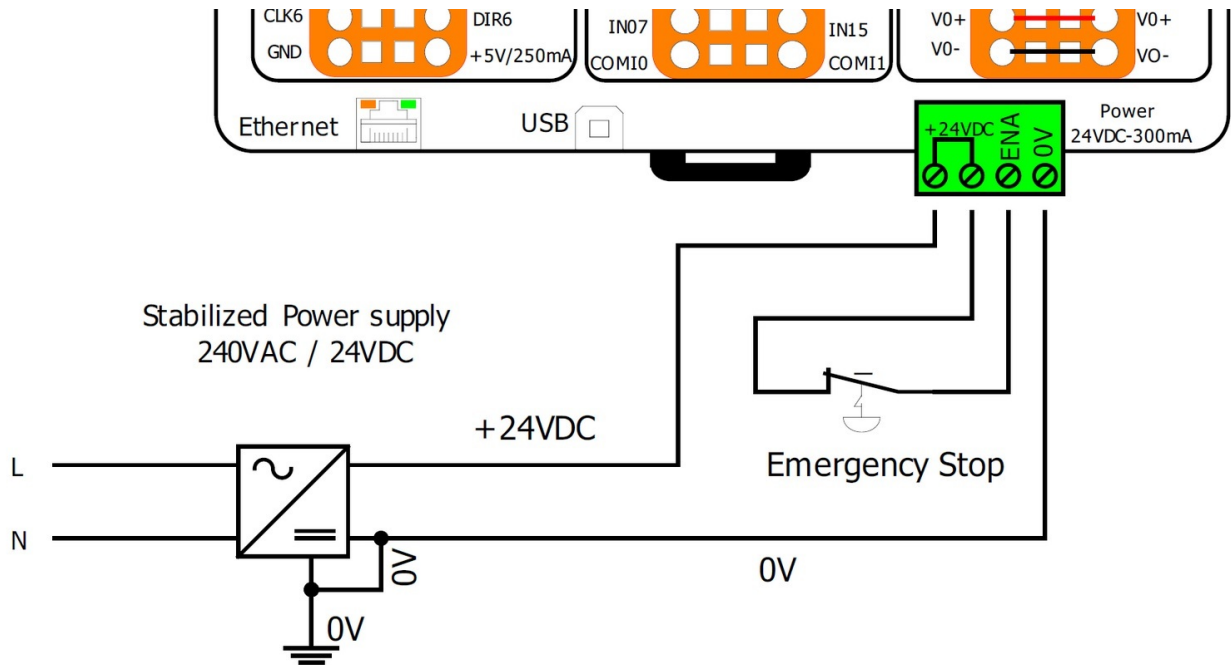
Regardless of the type of interface used, all data exchanges remain based on the Modbus protocol. The card can then be configured either as Master or Slave (see table of parameters).

As a general rule, the HMI always occupies the master position, and the InterpCNC board the slave position.

For simplified implementation: all the connectors are removable and for the orange connectors, the connection of the wires is of the quick type (use a small flat screwdriver to release each central pressure spring, both on insertion and on removal).

Wiring

Power supply, Emergency stop



The connection between +24V and ENA is essential. Therefore, use an NC (normally closed) type contact to ensure this connection.

For safety, an open contact on this link will deactivate the axis command outputs, and will leave outputs OUT0 to 15 as they are, until the next time the emergency stop button is pressed (acknowledgment and recovery).

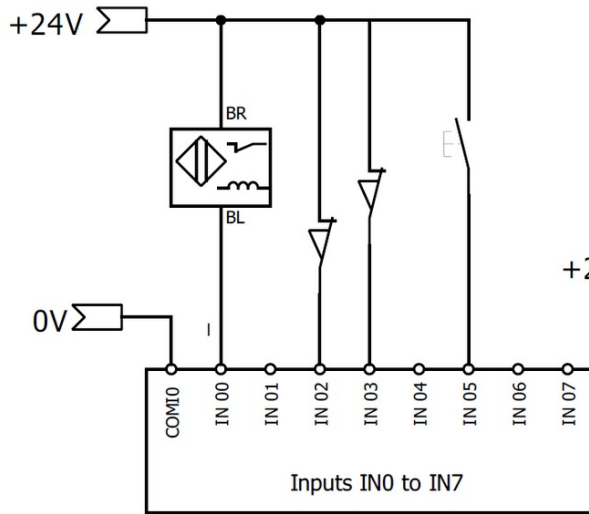
Connection of inputs

The physical inputs of the card can be configured either in PNP or in NPN.

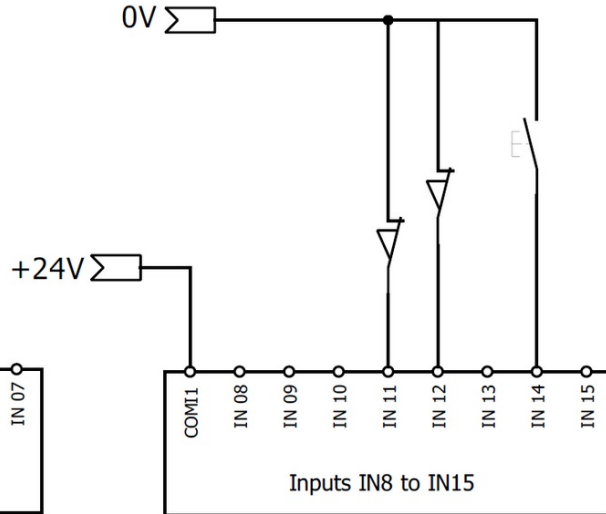
A different configuration is possible on each group of inputs associated with a COMI.

Wiring examples:

P type Inputs
(COMI connected to 0V)



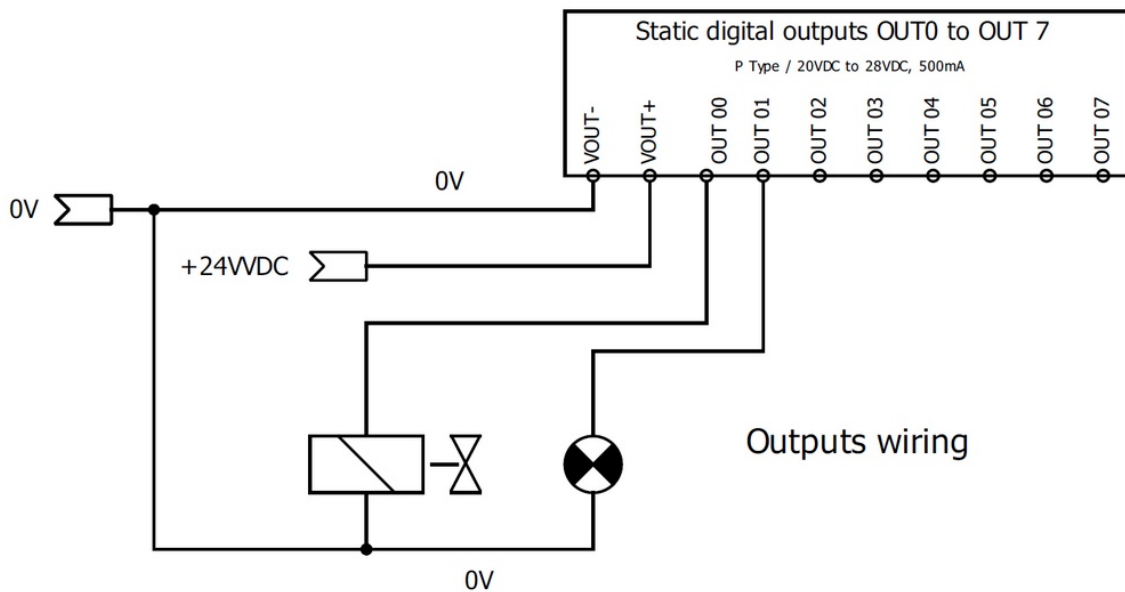
N type Inputs
(COMI connected to +24VDC)



Connection of outputs

The static outputs must be powered externally (connect +24V to VOUT+, and 0V to VOUT-). They are of the PNP type.

Wiring example:

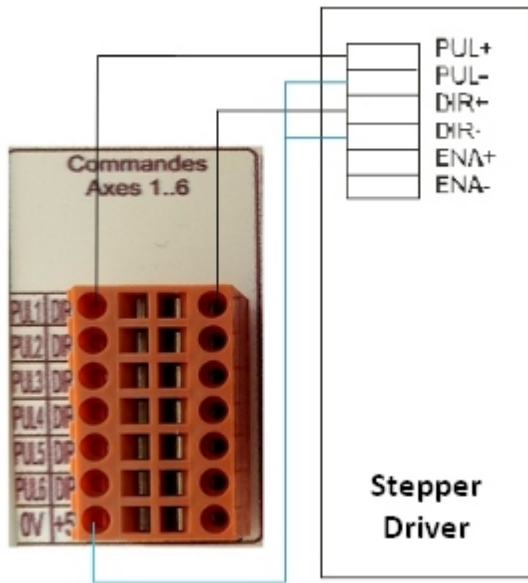


The same for outputs 8 to 15.

NB: the +VOUT and -VOUT are connected internally on the card. You can therefore supply only one side (unless the cumulative intensity of the outputs turns out to be high, you will thus increase the caliber).

Motor driver control

The pulses/direction commands of the card are to be linked directly to the corresponding inputs of the driver.

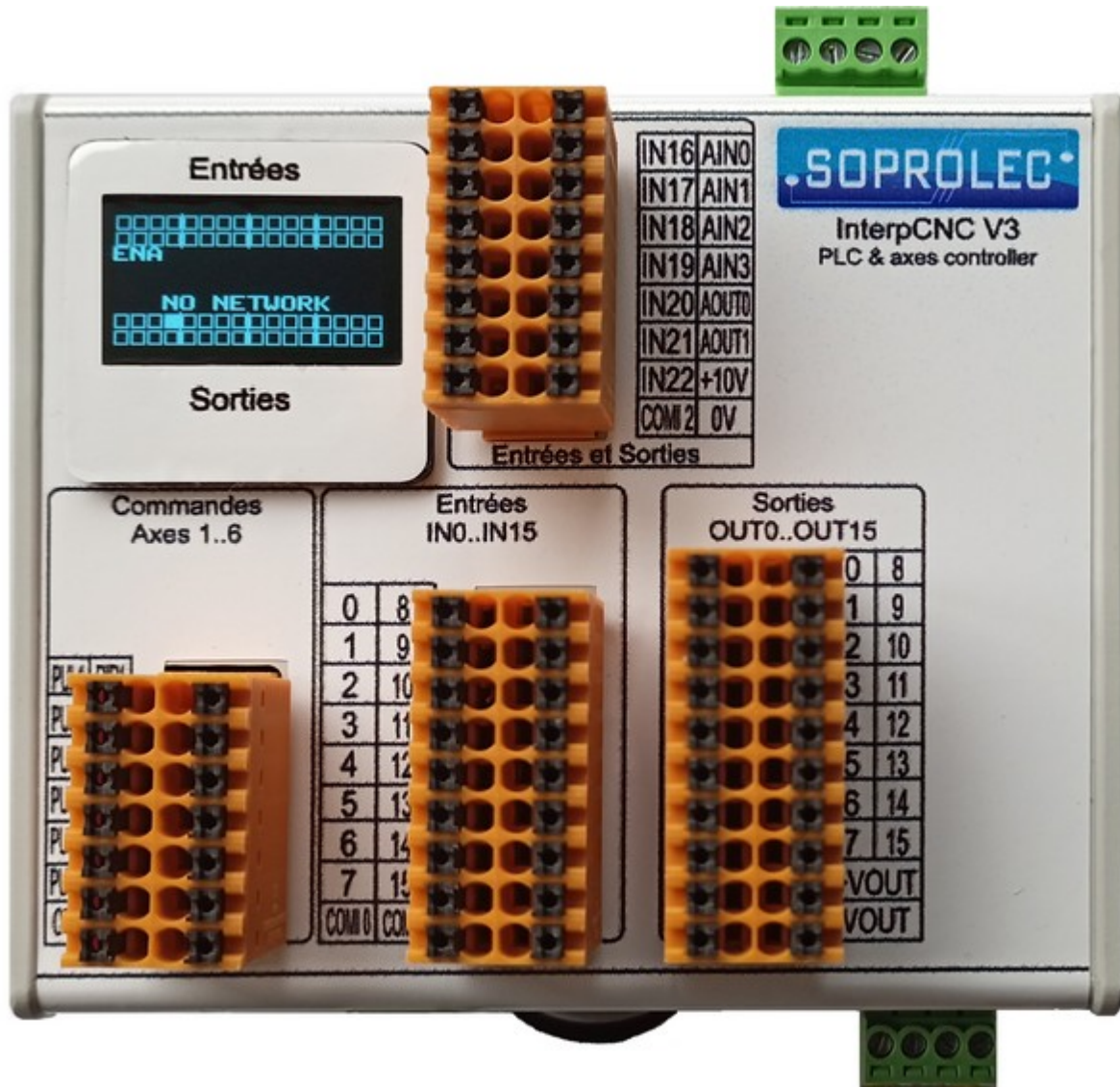


MODBUS_documentation

SOPROLEC
ZAC DE L'EPINE
72460 SAVIGNE L'EVEQUE
FRANCE
Tél : +33 (0)2 4376 4476



SOPROLEC
InterpCNC V3
Axes card



MODBUS communication manual

Introduction

This documentation is intended for advanced programming of the InterpCNC 3 card.

It informs you of a certain number of registers and system bits, as well as their type of addressing.

Thus it becomes possible by reading them, by testing them, to carry out for example your own functions allowing what the already existing functions of the Basic PLC would not have already envisaged for you.

The Modbus commands described allow the card to be controlled without an embedded Basic PLC program.

Indeed, the InterpCNC 3 card is also capable of executing the Modbus commands coming by Example from an external PLC.

The card receives in its buffer the Modbus frames intended for its ID, and executes them.

The RS485 COM1 and COM2 links use the Modbus RTU protocol. These 2 serial ports must be configured in Slave or Master mode for extension management.

The USB link also uses the ModbusRTU protocol with serial port emulation (CDC).

The Ethernet link also uses the Modbus IP protocol in TCP or in UDP.

In server mode, a maximum of 6 clients is allowed.

Identification of InterpCNC PLCs connected to the Ethernet network

The PLC responds to a UDP broadcast type request on port 58080. This default port can be changed via parameter 544.

The broadcast request must contain the following 3 character string: "0 1"

The response from the PLCs will be a string containing the following information:

IP address, TCP port, MAC address, Netbios name,

The response format is as follows: "IP=%s :%d;MAC=%02X:%02X:%02X:%02X:%02X:%02X;NAME=%s"

Example of response :

```
«IP=192.168.10.147:502;MAC=2E:52:B9:1A:03:31;NAME=ICNC3-1A0331»
```

Identification of InterpCNC PLCs connected via USB

Attached cards will be seen as a standard serial port. You can identify the InterpCNCs by the PID and VID information which is as follows:

```
ICNCVID = "0483" et ICNCPID = "5740"
```

The following Code example in C# is used to list the serial ports (partial code)

```
// Search all CDC COM adapter in registry
private void GetCOMPortList(List<ConnexionInfo> USBCOMlist)
{
    try
    {
        ManagementObjectSearcher searcher =
            new ManagementObjectSearcher("root\\CIMV2",
                "SELECT * FROM Win32_PnPEntity WHERE Name LIKE '%(COM[0-9])%'");
        // "SELECT * FROM Win32_PnPEntity");

        try
        {
            foreach (ManagementObject queryObj in searcher.Get())
            {
                if (queryObj["Caption"].ToString().Contains("(COM)")
                {
                    //List<string> DevInfo = new List<string>();

                    string Caption = queryObj["Caption"].ToString();
                    int CaptionIndex = Caption.IndexOf("(COM)");
                    string CaptionInfo = Caption.Substring(CaptionIndex +
1).TrimEnd(')');

                    string deviceId = queryObj["deviceid"].ToString(); //"DeviceID"

                    int vidIndex = deviceId.IndexOf("VID_");
                    int pidIndex = deviceId.IndexOf("PID_");
                    string vid = string.Empty, pid = String.Empty;
```

```

        if (vidIndex != -1 && pidIndex != -1)
        {
            string startingAtVid = deviceId.Substring(vidIndex + 4);

            long vid = startingAtVid.Substring(0, 4); // vid is four characters

            string startingAtPid = deviceId.Substring(pidIndex + 4);

            long pid = startingAtPid.Substring(0, 4); // pid is four characters
        }

        ConnexionInfo DevInfo = new ConnexionInfo();
        if ((vid == "0483") && (pid == "5740"))
            DevInfo.TypeConnexion = ePortType.TYPE_SERIAL_INTERPCNC;
        else
            DevInfo.TypeConnexion = ePortType.TYPE_SERIAL;

        DevInfo.COMPort = CaptionInfo; // Add(CaptionInfo);
        DevInfo.VID = vid;
        DevInfo.PID = pid;

        USBCOMlist.Add(DevInfo);
        mHandlerFormFunctionForAddItem?.Invoke(DevInfo);
    }
}
}
catch (NullReferenceException ex)
{
}
}
catch (ManagementException ex)
{
}
}
}

```

Read/Write Parameters

Access to the saved parameters of the InterpCNC card can be done using the ICNCStudio utility, but also in Modbus:

The parameter table is accessible in Read/Write (Holding registers) from address 32000. All parameters are in 32-bit format (2 registers per parameter).

Adresses of Input Bits (Read only)

- 1X type variables -

The registers represent the state of the discrete inputs of the InterpCNC card as well as the status bits.

All of this information is also available as 16-bit words in read-only registers. In the PLC program, these variables are accessible with the commands:

- IN(0..255), DFM(0..255), DFD(0..255) for the inputs
- STSBit(256..399) for the status bits

0 to 255	digital inputs Also available as Input register 1000 to 1015
256 to 263	Movement bits in progress, 1 bit per axis, Also accessible in Input register 1016 / 8 least significant bits
264 to 271	Movement direction (bit at 0 if negative movement, at 1 if positive movement) Also accessible in Input register 1016 / 8 most significant bits
272 to 279	Current homing sequence Also accessible in Input register 1017 / 8 least significant bits
280 to 287	Error during homing sequence Also accessible in Input register 1017 / 8 most significant bits
288 to 295	Limit switch negative direction active Also accessible in Input register 1018 / 8 least significant bits
296 to 303	Positive direction limit switches active Also accessible in Input register 1018 / 8 most significant bits
304 to 311	Reserved Also accessible in Input register 1019 / 8 least significant bits
312 to 319	Reserved Also accessible in Input register 1019 / 8 most significant bits
320 to 327	COM Status Also accessible in Input register 1020 / 8 least significant bits
	320 One or more saved registers (EEDATA) modified outside the PLC program. Cleared on call to IsEEdataChanged
	321 recipe Changed. Set to 1 when there is a change in the recipes by communication channel or manipulation by the RCP functions. To be tested with PLC command IsRCPChanged. Reset to 0 when calling IsRCPChanged
	322 InterpCNC parameter changed outside the PLC. Cleared on call to IsPrmChanged
	323 USB input in communication with the Grbl protocol and no longer Modbus RTU. Automatic switching to the Grbl protocol on receipt of a frame of length <=4 characters and starting with '\$' or Ctrl-X (chr(0x18))
	324 : DMX Slave Frame received (automatically goes to 0 after reading the bit by stsBit(324)
	325 : The DMX master sends frames to the controller
328 to 335	Miscellaneous Status 1, Also accessible in Input register 1020 / 8 most significant bits
	328 Enable input state
	329 Card locked (same as 328)
	333 Override CNC machining speed <> 100 %
336 to 343	Miscellaneous Status 2, Also accessible in Input register 1021 / 8 least significant bits
344 to 351	Status PLC Basic Also accessible in Input register 1021 / 8 most significant bits
	346 PLCBasic running
352 to 359	Network Status Also accessible in Input register 1022 / 8 least significant bits
	352 Ethernet cable connected
	354 IP Address allocated
	355 Connexion IOT effective
	356 RTC initialized by the SNTP server

	357 Connection to SNTP server ntp.pool.org active
	358 Completed SMTP connection setup
360 to 367	Current probe axis 1 to 8 Also accessible in Input register 1022 / 8 most significant bits
368 to 375	Probe Error Axes 1 to 8 Also accessible in Input register 1023 / 8 least significant bits
376 to 383	Reserved 3
384 to 391	Reserved 4
392 to 399	Reserved 5

Addresses of Input registers (Read only)

- 3X type variables -

These registers represent the internal variables of the InterpCNC.

NB: They can be read in PLC Basic with the same commands as for Holding registers, provided you add 100000 to the register address:

GetMW(1xxxxx), **GetMDW(1xxxxx)**, **GetMI(1xxxxx)**, **GetMDI(1xxxxx)**, **GetMF(1xxxxx)**.

Examples :

- GetMW(101000) to read state of IN0 to IN15 inputs
- GetMDW(101330) to read the result of probing on axis 1

1000 to 1015	Mapping of inputs IN0 to IN255
1016 to 1025	<p>Status Bit Mapping</p> <p>1016 Least significant: Axis bits in motion 1016 Most significant: Movement direction bit in progress (0 if negative, 1 if positive)</p> <p>1017 Least significant: Homing bits in progress 1017 High: Bits Homing Errors</p> <p>1018 Least significant: Negative limit bits active 1018 Least significant: Positive limit bits active.</p> <p>1019: Reserve</p> <p>1020: Miscellaneous status (see details on bits 320 to 335)</p> <p>1021: Miscellaneous status (see details on bits 336 to 351)</p> <p>1022 Least significant: Ethernet network status (see details on bits 352 to 359) 1022 Least Significant: Axis Flags in Probe Sequence</p> <p>1023 Least significant: Probe sequence error flags</p>
1026	Number of characters in Print PLC Basic buffer (command ? Or print)
1027	Number of characters in Trace PLC Basic buffer (command!)
1030 to 1041	Position counters of axes AXE1 to AXE 6 in pulses (signed 32-bit registers also available in Holding register 2400 to 2411)
1042 to 1053	Current axes movement speeds (32 bits signed)

1060 to 1071	Target current position (Last requested target position)
1072 to 1083	Current target speed (Last requested target speed)
1090 to 1097	Analog input AIN0 to AIN7
1100	CPU Load
1101	CPU Load for PLC
1102 to 1107	CPU ID Processor (96 bits)
1108 to 1111	NOR FLASH UID (64 bits)
1112 to 1114	Ethernet MAC address
1115	Firmware version High
1116	Firmware Version Low
1117	Bootloader Version High
1118	Bootloader version Low
1120	NOR FLASH Page size
1121	NOR FLASH Sector Size size
1122	NOR FLASH Bloc Size
1123	NOR FLASH Bloc Count
1124	NOR FLASH Total size Kbyte
1130	<p>Period in μs between two events on fast input IN16 LOW The input must be configured in Interrupt or Counter mode. In Interrupt mode, the measurement corresponds to the period between the last 2 edges (rising or falling without distinction). In counter mode, only the rising edges are taken into account. To take falling edges into account, the input polarity must be reversed (see parameter 200).</p>
1131	Event period on fast input IN16 HIGH
1132	Event period on fast input IN17 LOW
1133	Event period on fast input IN17 HIGH
1134	Event period on fast input IN18 LOW
1135	Event period on fast input IN18 HIGH
1136	Event period on fast input IN19 LOW
1137	Event period on fast input IN19 HIGH
1138	Event period on fast input IN20 LOW
1139	Event period on fast input IN20 HIGH
1140	Event period on fast input IN21 LOW
1141	Event period on fast input IN21 HIGH
1142	Event period on fast input IN22 LOW
1143	Event period on fast input IN22 HIGH
1200	Number of connections on TCP clients (ICNC \rightarrow client)
1201	TCP Client transmission error LOW
1202	TCP Client transmission error HIGH
1203	TCP Client frame counter LOW
1204	TCP Client frame counter HIGH
1220	Number of TCP clients connected (Client \rightarrow ICNC) (10 max)
1221	Server TCP transmission error LOW
1222	Server TCP transmission error HIGH
1223	TCP frame counter Server LOW
1224	TCP frame counter Server HIGH
1230	Modbus RTU master mapping request counter 1 LOW

1231	Modbus RTU master mapping request counter 1 HIGH
1232	Modbus RTU master mapping request counter 2 LOW
1233	Modbus RTU master mapping request counter 2 HIGH
1234	Modbus RTU master mapping request counter 3 LOW
1235	Modbus RTU master mapping request counter 3 HIGH
1236	Modbus RTU master mapping request counter 4 LOW
1237	Modbus RTU master mapping request counter 4 HIGH
1238	Modbus RTU master mapping request counter 5 LOW
1239	Modbus RTU master mapping request counter 5 HIGH
1240	Modbus RTU master mapping request counter 6 LOW
1241	Modbus RTU master mapping request counter 6 HIGH
1242	Modbus RTU master mapping request counter 7 LOW
1243	Modbus RTU master mapping request counter 7 HIGH
1244	Modbus RTU master mapping request counter 8 LOW
1245	Modbus RTU master mapping request counter 8 HIGH
1246	Modbus RTU master mapping request counter 9 LOW
1247	Modbus RTU master mapping request counter 9 HIGH
1248	Modbus RTU master mapping request counter 10 LOW
1249	Modbus RTU master mapping request counter 10 HIGH
1250	Modbus RTU master mapping request counter 11 LOW
1251	Modbus RTU master mapping request counter 11 HIGH
1252	Modbus RTU master mapping request counter 12 LOW
1253	Modbus RTU master mapping request counter 12 HIGH
1254	Modbus RTU master mapping request counter 13 LOW
1255	Modbus RTU master mapping request counter 13 HIGH
1256	Modbus RTU master mapping request counter 14 LOW
1257	Modbus RTU master mapping request counter 14 HIGH
1258	Modbus RTU master mapping request counter 15 LOW
1259	Modbus RTU master mapping request counter 15 HIGH
1260	Modbus RTU master mapping request counter 16 LOW
1261	Modbus RTU master mapping request counter 16 HIGH
1262	Modbus RTU master mapping success counter 1 LOW
1263	Modbus RTU master mapping success counter 1 HIGH
1264	Modbus RTU master mapping success counter 2 LOW
1265	Modbus RTU master mapping success counter 2 HIGH
1266	Modbus RTU master mapping success counter 3 LOW
1267	Modbus RTU master mapping success counter 3 HIGH
1268	Modbus RTU master mapping success counter 4 LOW
1269	Modbus RTU master mapping success counter 4 HIGH
1270	Modbus RTU master mapping success counter 5 LOW
1271	Modbus RTU master mapping success counter 5 HIGH
1272	Modbus RTU master mapping success counter 6 LOW
1273	Modbus RTU master mapping success counter 6 HIGH
1274	Modbus RTU master mapping success counter 7 LOW
1275	Modbus RTU master mapping success counter 7 HIGH
1276	Modbus RTU master mapping success counter 8 LOW
1277	Modbus RTU master mapping success counter 8 HIGH
1278	Modbus RTU master mapping success counter 9 LOW
1279	Modbus RTU master mapping success counter 9 HIGH
1280	Modbus RTU master mapping success counter 10 LOW
1281	Modbus RTU master mapping success counter 10 HIGH
1282	Modbus RTU master mapping success counter 11 LOW

1283	Modbus RTU master mapping success counter 11 HIGH
1284	Modbus RTU master mapping success counter 12 LOW
1285	Modbus RTU master mapping success counter 12 HIGH
1286	Modbus RTU master mapping success counter 13 LOW
1287	Modbus RTU master mapping success counter 13 HIGH
1288	Modbus RTU master mapping success counter 14 LOW
1289	Modbus RTU master mapping success counter 14 HIGH
1290	Modbus RTU master mapping success counter 15 LOW
1291	Modbus RTU master mapping success counter 15 HIGH
1292	Modbus RTU master mapping success counter 16 LOW
1293	Modbus RTU master mapping success counter 16 HIGH
1294	Modbus RTU master mapping error counter 1 LOW
1295	Modbus RTU master mapping error counter 1 HIGH
1296	Modbus RTU master mapping error counter 2 LOW
1297	Modbus RTU master mapping error counter 2 HIGH
1298	Modbus RTU master mapping error counter 3 LOW
1299	Modbus RTU master mapping error counter 3 HIGH
1300	Modbus RTU master mapping error counter 4 LOW
1301	Modbus RTU master mapping error counter 4 HIGH
1302	Modbus RTU master mapping error counter 5 LOW
1303	Modbus RTU master mapping error counter 5 HIGH
1304	Modbus RTU master mapping error counter 6 LOW
1305	Modbus RTU master mapping error counter 6 HIGH
1306	Modbus RTU master mapping error counter 7 LOW
1307	Modbus RTU master mapping error counter 7 HIGH
1308	Modbus RTU master mapping error counter 8 LOW
1309	Modbus RTU master mapping error counter 8 HIGH
1310	Modbus RTU master mapping error counter 9 LOW
1311	Modbus RTU master mapping error counter 9 HIGH
1312	Modbus RTU master mapping error counter 10 LOW
1313	Modbus RTU master mapping error counter 10 HIGH
1314	Modbus RTU master mapping error counter 11 LOW
1315	Modbus RTU master mapping error counter 11 HIGH
1316	Modbus RTU master mapping error counter 12 LOW
1317	Modbus RTU master mapping error counter 12 HIGH
1318	Modbus RTU master mapping error counter 13 LOW
1319	Modbus RTU master mapping error counter 13 HIGH
1320	Modbus RTU master mapping error counter 14 LOW
1321	Modbus RTU master mapping error counter 14 HIGH
1322	Modbus RTU master mapping error counter 15 LOW
1323	Modbus RTU master mapping error counter 15 HIGH
1324	Modbus RTU master mapping error counter 16 LOW
1325	Modbus RTU master mapping error counter 16 HIGH
1350	ProbePosition Axe 1 LOW
1351	ProbePosition Axe 1 HIGH
1352	ProbePosition Axe 2 LOW
1353	ProbePosition Axe 2 HIGH
1354	ProbePosition Axe 3 LOW
1355	ProbePosition Axe 3 HIGH
1356	ProbePosition Axe 4 LOW
1357	ProbePosition Axe 4 HIGH

1358	ProbePosition Axe 5 LOW
1359	ProbePosition Axe 5 HIGH
1360	ProbePosition Axe 6 LOW
1361	ProbePosition Axe 6 HIGH
1998	DMX frame counter received LOW
1999	DMX frame counter received HIGH
2000	Size of DMX frame received
2001 to 2512	DMX channels. Value between 0 and 255
2990	Modbus CNC Buffer Size
2991	GRBL Command Buffer Size
2992	CNC planning buffer size
3000	Space available in the CNC Modbus communication buffer (4096 max)
3001	Space available in the Grbl CNC communication buffer (1024 max)
3002	Space available in the CNC planner buffer (35 max)
3003	CNC status Bit 0: Alarm Bit 1: Gcode in Test mode Bit 2: CNC homing in progress Bit 3: Cycle in progress Bit 4: Pause in progress (Feed hold) Bit 5: Jog in progress Bit 6: Door safety active Bit 7: Sleep mode active Bit 8: Emergency stop active Bit 9: Manual tool change in progress
3004	CNC alarm code: 0: no alarm 1: Limit per limit switch 2: Stroke limit by software 3: Cycle Aborted 4: Initial state of sensor contact faulty, 5: Probe contact detection error 6: Error related to initialization during homing 7: Error related to opening door during homing 8: Homing contact that remains engaged despite clearing 9: Homing contact not found (machine run) 10: Emergency stop activated 11: Homing sequence required 12: Abnormal probe contact detection 13: Triggering of the tool probe during the cycle or in Jog 14: Spindle ready signal timeout error 15: Maximum deviation tolerance between master/slave axes exceeded 16: Internal error 17: Fault detected on motor (Motor Error input) 18: Event detection timeout (modbus command 1012)
3010..3011	Current CNC travel speed (UINT32, mm/min)
3012	THC Status: • b2: THC allowed. Activated by THCon or THConAuto commands. Reset by THCStop command. • b6: Active THC. Activated by THCon, THConAuto or THC Resume. Reset by THCStop or THCPause. Indicates that THC is active.

	<ul style="list-style-type: none"> • b9: THC locked due to underspeed compared to speed cutting requested • b11: Delay before activation in progress (parameter Setting_THC_Delay) • b12: Sampling for THC voltage auto adjustment in progress (if sampling requested, this bit is also at 1 during the initial delay) • b13: Sampling result is out of tolerance. There Value used is limited to the allowed range. The bit goes back to 0 at stop THC
3013,,3014	(int32, mV) Arc voltage measurement in mV. Automatic scaling in depending on the offset and maximum measurement voltage parameters. Indication permanent even if the THC is inactive.
11000 to 11114	Buffer for Print commands (Used by ICNCStudio)
11125 to 11189	Buffer for Trace commands (64 bytes, Used by ICNCStudio)
12000 to 12999	Buffer for MCU memory reading (Used by ICNCStudio)
12000 to 12200	Buffer for NOR FLASH memory reading (Used by ICNCStudio)

Addresses of Holding registers (Read/write)

- 4X type variables -

These registers are used to action the InterpCNC card and to launch axis management commands.

2000 to 2149	Buffer for MODBUS commands
2150 to 2157	Analog outputs AOUT0 to AOUT7
2160 to 2165	Mapping of outputs OUT0 to OUT95
2166 to 2191	Mapping of user bits (coils)
2360 to 2367	Encoder mode counter for 4 QEI channels
2380 to 2395	Discrete input fast counters IN17 to IN24 (unsigned 32-bit registers). The input must be configured in Interrupt or Counter mode. In interrupt mode, all state changes will be counted. In counter mode, rising edges will be counted. To count the falling edges, it is therefore necessary to invert the polarity of the input (see parameters 200).
2400 to 2411	Position counters for axes AXE1 to AXE 6 (signed 32-bit registers)
2412 to 2423	Current speeds (in pulses/s) of the axes during movement (not usable for CNC commands)
2430	Normal Speed Override in CNC Mode
2431	Fast speed override in CNC mode
2444,,2445	(uint32_t, mV) Arc voltage setpoint for THC. Initialized by external control software or by measurement sampling.
2446	(int16_t, mV) Offset applied to the arc voltage setpoint in real time for manual adjustment
2447	(uint16_t, %) : Speed override on THC move (0% to 1000%) This register acts in the same way as the speed gain parameter of the THC settings. It is initialized to 100 on power-up. A Value of 0 will result in a THC shutdown.

	It can be modified at any time with immediate consideration. There modification can also be done using a buffered register write action.
2800 to 2899	Indexes for indexed reads/writes. That is 50 32-bit registers allowing the indexing of the different types of Modbus data. 0 to 65535 for read-only registers (Input registers) 100000 to 165535 for read/write registers (Holding registers) 200000 to 265535 for read-only bits (Input bits) 300000 to 365535 for read/write bits (Coils)
3000 to 3999	User registers in RAM (1000 16-bit registers)
4000 to 4999	User registers in saved RAM (1000 16-bit registers)
5000	DMX Master channel for DMX Master transmission
5001 to 5512	Values of the 512 DMX channels for DMX master transmission
9995	Size of recipes
9996	Index 0 recipe
9997	Index 1 recipe
9998	Index 2 recipe
9999	Index 3 recipe
10000 to 10999	Recipe page 0
11000 to 11999	Recipe page 1
12000 to 12999	Recipe page 2
13000 to 13999	Recipe page 3
32000 to 33999	InterpCNC parameters (index 0 to 1999, 2 registers per parameter)
62000	MB_HOLD_ADDR_MCU_MEMORY_ReadPtr_low (Self-incremented pointer to read CPU memory content)
62001	MB_HOLD_ADDR_MCU_MEMORY_ReadPtr_high
62002	MB_HOLD_ADDR_FLASH_NAND_ReadPtr_low
62003	MB_HOLD_ADDR_FLASH_NAND_ReadPtr_high

About sending Modbus commands

Commands are sent to the InterpCNC through the command buffer located between addresses 2000 and 2149 (Holding registers type variables).

Each command is identified by a command number detailed below.

You can use Modbus functions 06 (write single register) or 16 (write multiple registers) to write to this buffer.

If you are limited to using the Modbus function 06, the event that will start command processing is the writing of the command code located at address 2000. It is therefore necessary to transfer the arguments beforehand of the command.

If you use the Modbus function 16, you can send all the arguments with the command code on a single request.

If several masters or clients are connected to the InterpCNC, we strongly recommend that you use the Modbus function 16 to avoid conflicting access to the command buffer.

Commande100 : Stop an axis

This command allows a single axis to be stopped, using its identifier.

It is equivalent to the Basic interpreter command: **StopAxisID**

Address	2000	2001
----------------	------	------

Parameter	Command ID	Axis ID
Value	100	1,,6

Command 101 : Stop one or several axes

This command is used to Stop one or more axes, identified by their respective bit on a 16-bit word.

It is equivalent to the Basic interpreter command: **StopAxes**

The deceleration used is that indicated when launching the movement command.

You can check the effective stopping of the axes via status bits 256 to 261 "Axis moving"

Address	2000	2001
Parameter	Command ID	Axes bits
Value	101	0x01 à 0x3F

Command 102: Move an Axis at a given Speed

This command allows the movement of an axis until it reaches a given speed.

It is equivalent to the Basic shell command: **MoveSpeed**

The movement can be stopped by a stop command (command 100 or 101) or by indicating a zero movement speed.

The actual stop can then be checked by the "Axis moving" status bits.

After launching a speed movement command, you still have the option of launching a position movement command (command 103 or 104).

Address	2000	2001	2002	2003	2004	2005
Parameter	Command ID	Axis ID	Accel/Decel		Speed	
Value	102	1,,6	LW Accel	HW Accel	LW Frequency	HW Frequency

Command 103: Move an Axis to Target Position

The velocity profile is given by acceleration, velocity and deceleration.

This command moves an axis to a target position.

It is equivalent to the Basic interpreter command: **MoveAxe**

Each axis has its own profile generator. It is therefore possible to launch different movements simultaneously on several axes.

As soon as a movement command is launched, the "Axis in motion" status bit associated with the axis changes to 1. It returns to 0 when the target is reached.

It is also possible to change Target/Speed on the fly. That is, during an ongoing move. If the new target requires a rollback, it will be executed automatically. The "axis in motion" status bit does not go to 0 during the reversal of the direction of movement of the axis.

Address	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009
Param	Command ID	Axis ID	Acceleration		Speed		Deceleration		Position cible	
Value	103	1,,6	LW Accel	HW Accel	LW Frequency	HW Frequency	LW Frequency	HW Frequency	LW Target	HW Target

Command 104: Move an axis specified number of steps from current position

The velocity profile is given by acceleration, velocity and deceleration.

This command allows an axis to be moved over a defined number of steps. It is equivalent to the Basic interpreter command: **MoveAxeRelatif**

If an axis is moving in speed mode (command 102), you can issue a move command relative to the current position. This allows for Example to trigger the continuous rotation of a motor (function 102) until a cell is detected. When this information arrives, move the axis a given distance.

Address	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009
Param	Command ID	Axis ID	Acceleration		Speed		Deceleration		Number of steps	
Value	104	1,,6	LW Accel	HW Accel	LW Frequency	HW Frequency	LW Frequency	HW Frequency	LW Target	HW Target

Command 105: Write Current Position Counter (same as writing to position registers)

This command is equivalent to the Basic interpreter command: **SetPos(axis ID)**

It is imperative never to change the position registers during a movement.

Address	2000	2001	2002	2003
Parameter	Command ID	Axis ID	Position	
Value	105	1,,6	LW Position	HW Position

Command 106: Launch homing of an axis

The homing command is used to initialize the position of an axis after powering up using a limit switch.

The homing procedure takes place in 3 stages:

- Rapid movement until a limit switch is detected
- Slow backspace until the limit signal is lost, The position register then takes the Value of the argument "Home position to set"
- Complementary clearance movement relative to the position of loss of the Homing signal (according to argument Clearance).

If the input used for the homing sequence is already assigned to the limit switch management function, it is temporarily deactivated. So you can use a common sensor for

the homing or limit switch function.

This command is equivalent to the Basic interpreter Home command:

2002: **Homing Mode** is still 0

2003: *Input Number* is the number of the input receiving the sensor.

2004: **Expected Input state**: Input state triggering the end of the procedure.

2014 and 2015: **Max stroke** = Max stroke (in steps)

2016: **Tempo Reverse Direction**, pause time (in ms), before returning to the sensor

2017 and 2018: **Home Position** to set, Value at which the front position counter is initialized

clearance (most often set to 0)

2019 and 2020: **Clearance** = Clearance relative to the original position (in steps)

You can launch homing sequences simultaneously on several axes.

When the homing procedure is launched, the "Homing in progress" status bit changes to 1.

It automatically returns to 0 when the homing is finished (with or without error).

If the homing did not take place normally or was interrupted by a Stop axis command, the "Homing Error" status bit will be set to 1. Errors can be linked to incorrect parameters in the command or to the fact that the input has not been activated within the maximum travel distance allowed in the control.

The error bit automatically returns to 0 when launching a new homing command.

In a program using the homing function, it is therefore necessary to test the "Homing in progress" status bit (bits 272 to 277) then, to check that it is working correctly using the "Homing Error" bit (bits 280 to 285)

The argument "Home position to set" allows to indicate the Value that the position register takes at the moment of the loss of the homing sensor during the reverse slow movement. This argument can be positive or negative.

Let's take the Example where you want a homing position at 0 but 500 steps away from the sensor. You must then indicate a homing position of -500 and a clearance of 500.

Address	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009
Parameter	Command and ID	Axis ID	Homing Mode	Input Number	Expected Input State	Direction	High Speed		Low Speed	
Value	106	1,6	0	0,255	0 or 1	0 : Negative 1 : Positive	LW High speed	HW High speed	LW Low speed	LW Low speed

2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
Acceleration		Deceleration		Max Stroke		Tempo Reverse Direction	Home Position to set		Clearance	
LW Acceleration	HW Acceleration	LW Deceleration	HW Deceleration	LW Max Stroke	HW Max Stroke	Time in ms	LW Home Position	HW Home Position	LW Clearance	HW Clearance

Command 107: Launch a Probe on an input

This command is equivalent to the Probe command of the Basic interpreter:

2002: **Input Number** is the number of the input receiving the sensor.

2003: **Expected Input state** is the expected state on the input for detection.

2010 and 2011: **Max stroke** corresponds to the Max stroke (in steps). It is a signed Value whose ± sign indicates the direction.

The progress of the sequence is materialized by the status bits "Probe in progress" (status bits 360 to 365) and "Probe Error". (status bits 368 to 373)

When launching the command, the "Probe in progress" bit changes to 1 and the "Probe Error" bit changes to 0.

At the end of the sequence, the "Probe in progress" bit changes to 0. If an error has occurred during the sequence or this sequence has been stopped by a stop axis command, the "Probe error" bit will be activated . It is therefore appropriate to test the "Probe Error" bit after the change to 0 of the "Probe in progress" bit.

The result of this command is the position of the axis when the indicated input is activated. These positions are available in Inputs registers 1330 to 1341.

NB: Several probing sequences can be launched simultaneously on different axes.

Address	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
Param	Command ID	Axe ID	Input Number	Expected Input state	Acceleration		Vitesse		Deceleration		Max stroke (signed value → Direction)	
Value	107	1,,6	0,,255	0 or 1	LW Acceleration	HW Acceleration	LW Velocity	HW Velocity	LW Deceleration	HW Deceleration	LW Max Stroke	HW Max Stroke

Fonction C associée :

```
int ICNC3_ProbeAxe(modbus_t *ctx,
    uint8_t AxeID,
    uint16_t InputNumber,
    uint8_t ExceptedInputState, /* 0 if DIN=0 when switch is pressed (NC contact type) */
    uint32_t Acceleration,
    uint32_t Speed,
    uint32_t Deceleration,
    int32_t StrokeLimitStep)
```

Command 108: Start Probing on Multiple Inputs

This command is used to move an axis until an end of movement condition is reached on inputs DIN0 to DIN31

The sequence ends with the acquisition of the capture position when the following condition is met:

$$(((ActualInputStates0_31 \text{ AND } ANDMask) \text{ XOR } XORMask) \lt; \gt; 0)$$

2012 and 2013: **Max stroke** corresponds to the Max stroke (in steps). It is a signed Value whose ± sign indicates the direction.

The progress of the sequence is materialized by the status bits "Probe in progress" (status bits 360 to 365) and "Probe Error". (status bits 368 to 373)

When launching the command, the "Probe in progress" bit changes to 1 and the "Probe Error" bit changes to 0.

At the end of the sequence, the "Probe in progress" bit changes to 0. If an error has occurred during the sequence or this sequence has been stopped by a stop axis command, the "Probe error" bit will be activated . It is therefore appropriate to test the "Probe Error" bit after the change to 0 of the "Probe in progress" bit.

The result of this command is the position of the axis when the result of the lmagic test is <> 0. These positions are available in the Inputs registers 1330 to 1341.

NB: Several probing sequences can be launched simultaneously on different axes.

Address	2000	2001	2002	2004	2005	2006	2004	2007	2008	2009	2010	2011	2012	2013
Param	Command ID	Axe ID	AND mask	XOR mask			Acceleration		Speed		Deceleration		Max stroke (signed value → Direction)	
Value	108	1,,6	LW AND mask	HW AND mask	LW XOR mask	HW XOR mask	LW Acceleration	HW Acceleration	LW Velocity	HW Velocity	LW Deceleration	HW Deceleration	LW Max Stroke	HW Max Stroke

Associated C function :

```
int ICNC3_ProbeAxewithMask(modbus_t *ctx,
    uint8_t AxeID,
    uint32_t ANDMask,
    uint32_t XORMask,
    uint32_t Acceleration,
    uint32_t Speed,
    uint32_t Deceleration,
    int32_t StrokeLimitStep) ;
```

Command 109: Launch of a Probing on analog input threshold

This command allows you to move an axis until reaching a threshold on an analog input

The sequence ends with the acquisition of the capture position when the threshold is reached by applying the following test:

If operator = 0: probing as long as the analog input is <= to the indicated threshold

If operator = 1: probing as long as the analog input is >= to the indicated threshold

2012 and 2013: **Max stroke** is in steps. It is a signed value whose ± sign indicates the direction. The progress of the sequence is indicated by the status bits "Probe in progress" (status bits 360 to 365) and "Probe Error". (status bits 368 to 373)

When the command is issued, the "Probe in progress" bit changes to 1 and the "Probe Error" bit changes to 0.

At the end of the sequence, the "Probe in Progress" bit goes to 0.

If an error occurred during the sequence or this sequence was stopped by a stop axis

command, the “Probe error” bit will be activated and the captured position register will not be updated. It is therefore appropriate to test the “Probe Error” bit after the “Probe in progress” bit has gone to 0.

The result of this command is the position of the axis at the moment when the trigger threshold is reached. These positions are available in Input registers 1330 to 1341.

NB: Several probing sequences can be launched simultaneously on different axes.

Address	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012
Parameter	Command ID	Axis ID	AIN number	Operator	Threshold mV	Acceleration		Speed		Deceleration		Max stroke (signed value → Direction)	
Value	109	1,,6	0,,7	0,,1	0,,1000	LW Acceleration	HW Acceleration	LW Velocity	HW Velocity	LW Deceleration	HW Deceleration	LW Max Stroke	HW Max Stroke

Associated C function :

```
int ICNC3_ProbeAxeOnAnalogInput(modbus_t* ctx,
    uint8_t AxeID,
    uint32_t AnalogInputNumber,
    uint32_t Operator,
    int32_t Threshold_mV,
    uint32_t Acceleration,
    uint32_t Speed,
    uint32_t Deceleration,
    int32_t StrokeLimitStep);
```

Exemple :

```
#define PROBE_UNTIL_AIN_LOWER_THAN_THRESHOLD 0
#define PROBE_UNTIL_AIN_GREATER_THAN_THRESHOLD 1
#define STATUS_BIT_Z_PROBING_IN_PROGRESS 362 // Input bit #362
#define STATUS_BIT_Z_PROBING_ERROR 370 // Input bit #370
#define Z_PROBE_POSITION 354 // Input register #354, INT32

// Probe Z axis
// Analog input channel 0
// Move until AIN0 greater than 5000mV
// Accel = 10000Hz/s, Velocity = 1000Hz, Decel = 10000Hz/s
// Negative direction and maximum stroke of 2500 steps
//
// Return -1 in case of communication error
// Return 0 in case of probe error (ie, AIN0 level is greater than threshold
when probe start or threshold can't be reach before 2500 steps
// Return 1 with Probe position in case of success
//
int ProbeWithAnalogChannel(modbus_t* ctx, int* ProbePosition)
{
    int res;
    uint8_t inStatusBit;
    int ProbePositionResult;

    res = ICNC3_ProbeAxeOnAnalogInput(ctx, // Modbus handler context
        (uint8_t)3, // Axe Z ID
        0, // Analog input #0
        PROBE_UNTIL_AIN_GREATER_THAN_THRESHOLD, // Operator
        5000, // 5000mV threshold
```

```

        10000,      // Acceleration (Hz/s)
        1000,     // Velocity (Hz)
        10000,   // Deceleration Hz/s
        -2500);  // 2500 steps in negative direction as limited stroke
if (res <= 0)
    return -1; // Communication error

// Wait for en of probe or communication Error
do {
    Sleep(100);

    // Read Z probe in progress status bit
    res = modbus_read_input_bits(ctx,
STATUS_BIT_Z_PROBING_IN_PROGRESS, 1, &inStatusBit);

    } while ((inStatusBit == 1) || (res<=0));
if (res <= 0)
    return -1; // Communication error

// Check for probe error
res = modbus_read_input_bits(ctx, STATUS_BIT_Z_PROBING_ERROR, 1,
&inStatusBit);
if (res <= 0)
    return -1; // Communication error

if (inStatusBit != 0)
    return 0; // Probe error

// Read probe position result
res = ICNC3_Get32bitsInputRegister(ctx, Z_PROBE_POSITION,
&ProbePositionResult);
if (res <= 0)
    return -1; // Communication error

*ProbePosition = ProbePositionResult;
return 1; // Succes

```

Command 110: Force Inputs

This command is used to define the forcing of the state of an input. It is equivalent to the Basic interpreter command: SetIn InputNumber, State. When a force is active, all the functions of the PLC using the inputs will only see the forced state of this input.

Address	2000	2001	2002
Parameter	Command ID	Input Number	Forcing
Value	110	0,,255	-1, 0 ou 1

-1 => No forcing
0 => Forcing to 0
1 => Forcing to 1

Command 200 : PLCBasic command

This command allows the Direct Launch of a PLCBasic command (exactly as if it were sent from the 'Command:' line field from ICNCStudio). It is therefore executed immediately.

Address	2000	2001	2002	2003	---	2001+n/2
Parameter	Command ID	Command Length	Cmd	Cmd	Cmd	Cmd
Value	200	0,,255	Cmd[1] Cmd[0]	Cmd[3] Cmd[2]	---	10 Cmd[n-1]

A command must end with a line break (CHR(10)).

The length of the command is expressed in the number of characters in the command (including line break).

If the command has an odd number of characters, the last command register must be line feed CHR(10).

A command can be launched even if the PLC program is running.

If a PLC program is in RUN mode, sending an incorrect command will cause the PLC program to stop.

Using indexed Reads/Writes

This feature allows you to group the information you are interested in into a continuous Modbus address range. It is thus possible to optimize the flow of communication.

You have 2 address ranges for this.

- the index area located at addresses 2800 to 2899, i.e. 50 32-bit registers.
- the Values zone indexed at addresses 2900 to 2950, i.e. 50 16-bit registers.

The index area must be initialized when your application is launched to point to the Modbus information that you wish to group together. These indexes can point to Modbus variables of different types (Holding registers, Input registers, Input bits or Coils).

After initializing this area according to your needs, you can read/write your data in the indexed Values area. If indexes correspond to read-only variables, write operations will be ignored.

Let's take an example of an application where you want to regularly read the following information:

Input registers

1000: Status of inputs IN0 to IN15
 1016: Moving Axis Flag Bits
 1090: Analog input 0 status
 3000: Space available in the CNC command buffer
 3003: CNC status bits

Holding registers

2160: Status of outputs OUT0 to OUT15
 2400-2401: Position Axis 1

2402-2403: Position Axis 2

This information being of various types (Holding and Input registers), it is interesting to use indexing. Otherwise, reading all of this information requires a multitude of requests.

NB: To access the **Holding registers** through the Index table, add **100000** to the required Modbus addresses, add **200000** to access the addresses of the **Input bits**, and add **300000** to the addresses of the **Coils**.

In this particular case, you must initialize the index table with the following values (coded in 32 bits):

[1000, 1016, 1090, 3000, 3003, 102160, 102400, 102401, 102402, 102403]

Consider the following Modbus array of indexes with correspondence in the Indexed Values array:

Index table			
Modbus Address	16 bits Registers	32 bit value Of the Index	Pointed value
2800	0x03E8	1000	State of inputs IN0 to IN15
2801	0x0000	0x000003E8	
2802	0x03F8		
2803	0x0000	1016	Moving axes indicator bits
2804	0x0442	0x00000442	
2805	0x0000		Analog input 0 status
2806	0x0BB8	3000	Space available in the CNC command buffer
2807	0x0000	0x00000BB8	
2808	0x0BBB	3003	CNC status bits
2809	0x0000	0x00000BBB	
2810	0x8F10	102160	Status of outputs OUT0 to OUT15
2811	0x0001	0x00018F10	
2812	0x9000	102400	Position Axis 1 (LOW)
2813	0x0001	0x00019000	
2814	0x9001	102401	Position Axis 1 (HIGH)
2815	0x0001	0x00019001	
2816	0x9002	102402	Position Axis 2 (LOW)
2817	0x0001	0x00019002	
2818	0x9003	102403	Position Axis 2 (HIGH)
2819	0x0001	0x00019003	

Table of indexed values	
Modbus Address	Read value
2900	State of inputs IN0 to IN15
2901	Moving axes indicator bits
2902	Analog input 0 status
2903	Space available in the CNC command buffer
2904	CNC status bits
2905	Status of outputs OUT0 to OUT15
2906	Position Axis 1 (LOW)
2907	Position Axis 1 (HIGH)
2908	Position Axis 2 (LOW)
2909	Position Axis 2 (HIGH)

Reading registers 2900 to 2909 in one request therefore allows us to obtain all the required information.

CNC control functions

The InterpCNC has a buffered operating mode dedicated to the sequence of commands independently of the communication times between the PC and the CNC.

Whatever the communication channel (USB, Ethernet, RS485), the protocol used is the Modbus protocol.

We will distinguish 2 types of CNC controls:

- Buffered commands
- Immediate commands (= unbuffered)

Buffered CNC commands are numbered between 1000 and 1999.

Immediate effect CNC commands are numbered between 2000 and 2999

It is also possible to switch the communication protocol of the USB link to use a Grbl compatible mode. This switch occurs automatically when receiving a frame of length ≤ 4 characters and starting with '\$' or Ctrl-X ($\text{chr}(0x18)$). It remains active until the power is turned off.

Part 1: Buffered commands

There are 3 buffers used for the management of CNC functions:

- 1 buffer for communication with a Grbl compatible protocol on the USB link.
- 1 buffer for communication via Modbus commands.
- 1 buffer linked to commands processed by the motion planner.

The Grbl buffer makes it possible to exploit the embedded Gcode interpreter using many applications available in open source. We will first detail the control of the CNC via the Modbus protocol knowing that via this protocol and the Modbus 1000 command detailed below, it is also possible to work with Gcode commands.

It should also be noted that the on-board PLC program can operate in parallel with the processing of CNC commands. You can therefore have autonomous automation functions that do not interfere with the management carried out by the control PC.

Buffered commands will be placed in the dedicated CNC communication buffer via Modbus. They will then be processed by the planner to ensure the fluidity of the movements and possibly the actions synchronized with these same movements. Unbuffered commands that have an immediate effect. These are mainly functions for global modification of operation such as gear changes, break requests, etc.

The space available in the CNC Modbus command buffer can be read in the register (Input register 3000). Its size can be obtained by reading register 2990 (16-bit registers). The space available in the buffer of orders processed by the scheduler can be read in the fill level register of scheduled orders (Available size in Input register 2992).

You will find below the details of the commands necessary to operate the CNC functions.

Command 1000 : Executing a Gcode instruction

Sending a Gcode type instruction. The character string is broken down and placed in the Modbus command sending buffer. Be careful to take into account the inversion per byte in the 16-bit registers.

Length is the number of ASCII characters of the Gcode command.

The buffer used for this command is the Grbl command buffer and not the Modbus command buffer.

It is therefore advisable to check the space available in this buffer before sending a new command.

The space available in this buffer is accessible through the Input register 3001.

Address	2000	2001	2002	2003	---	2001+n/2
----------------	------	------	------	------	-----	----------

s										
Parameter	Command ID	Command Length	Cmd	Cmd	Cmd	Cmd	---	---	Cmd[n]	Cmd[n-1]
Value	1000	0,,255	Cmd[1]	Cmd[0]	Cmd[3]	Cmd[2]				

Example : to send a Gcode command : G01 X12.2 F3000

Grbl buffer occupancy: 2 + Command Length / 2

Command 1001 : Definition of the machining speed in mm/min

Used to indicate the machining speed for the movement commands to follow. The Speed Value is expressed in mm/min and of 16-bit unsigned integer type. The initial Value of the machining speed at power-up is 1mm/min.

Address	2000	2001	2002
Parameter	Command ID	float	
Value	1001	Speed mm/mn	

Occupancy in the Modbus buffer: 3 registers

Command 1002: Interpolated Linear Move of Axes to Target Positions (Absolute Positions)

This command is used to define the machining paths.

The first argument is used to define the speed to be taken into account (machining speed or rapid speed) and also to indicate the axes concerned by the command. The length of the frame will therefore depend on the number of axes to be moved. The machining speed must first be set using command 1001.

The positions are given in mm and of float type. The axis resolutions must therefore be correctly configured (see parameters 1100 to 1105).

Only the target positions of the axes to be moved and indicated in the first argument must be sent in the frame.

Address	2000	2001	2002	2003	2004	2005	...
Parameter	Command ID	Indicator	Float		Float		...
Value	1002	0x01 à 0x7F	Position 1 (mm)		Position 2 (mm)		...

Flag parameter detail:

- Bit 0 => The target of the X axis is indicated in the frame
- Bit 1 => The Y axis target is indicated in the frame
- Bit 2 => The Z axis target is indicated in the frame
- Bit 3 => The target of axis A is indicated in the frame
- Bit 4 => The B axis target is indicated in the frame

- Bit 5 => The C axis target is indicated in the frame
- Bit 6 => The movement speed is the maximum speed if the bit is at 1. Otherwise, the machining speed is used

Occupancy in the Modbus buffer: $2 + 2 * \text{Number of axes to move}$

Command 1003 : Circular interpolation

Allows you to create a circular interpolation type trajectory (Circle or arc of circle). The movement is made at the speed indicated by the last ICNC3_PushSetFeedRate command.

Address	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009
Parameter	Command ID	Direction	X		Y		I			J
Value	1003	0 : CW 1 : CCW	Position X (mm)		Position Y (mm)		Position I (mm)			Position J (mm)

Command 1010: Synchronized Action (Digital Output, Analog Output, Register)

Execution of a command synchronized with motions.

Commands can be of different types:

- Writing the state of a discrete output
- Writing an analog output
- Writing a Modbus register

You can place multiple synchronized actions in the buffer. They will all be associated with the next movement command and executed before the movement concerned. If no command is currently present in the buffer, the action is processed immediately.

Address	2000	2001	2002	2003
Parameter	Command ID	Action Type	Address	Value
Value	1010	1, 2 or 3	According to type	According to type

Action type 1: Definition of the state of an all or nothing output,

Action type 2: Set the state of an analog output

Action type 3: Writing to a Holding register.

Details of the different types of synchronized commands:

Address	2000	2001	2002	2003
Parameter	Command ID	Action Type	Digital Output number	Value
Value	1010	1	0 to 255	0 to 1

Action type 1 : Write a discrete output

Address	2000	2001	2002	2003
----------------	------	------	------	------

Parameter	Command ID	Action type	Analog. Output number	Value
Value	1010	2	0 to 7	0 to 10000mV

Action Type 2: Write Analog Output

Address	2000	2001	2002	2003
Parameter	Command ID	Action type	Register Address	Value
Value	1010	3	2000 to 65535	0 to 65535

Action type 3 : Write a Modbus register (Holding register)

Address	2000	2001	2002	2003
Parameter	Command ID	Action type	Not used	Not used
Value	1010	4	0 to 65535	0 to 65535

Action type 4 : THC Activation. Voltage setpoint fixed by control software in holding register 2444,,2445

Address	2000	2001	2002	2003
Parameter	Command ID	Action type	Duration measurement (ms)	Not used
Value	1010	5	0 to 65535	0 to 65535

Action type 5: Activation. Voltage set point set automatically by measuring the arc voltage at the start of the cut. The result of the measurement is stored in the holding registers 2444,,2445

Address	2000	2001	2002	2003
Parameter	Command ID	Action type	Not used	Not used
Value	1010	6	0 to 65535	0 to 65535

Action type 6: Stop THC function, the Z axis is stopped at its current position. The treatment of this action differs from the others. In particular, to allow resynchronization with the next axis movement commands, this command will freeze the processing of commands during the deceleration of Z if it is moving. Therefore, this action should not be used in the middle of a trajectory.

If you need to freeze the THC in a cut, use the THCPause action (Type 7)

Address	2000	2001	2002	2003
Parameter	Command ID	Action type	Not used	Not used
Value	1010	7	0 to 65535	0 to 65535

Type 7 action: THC pause to suspend torch height control. If there is a movement in progress, it is stopped.

Address	2000	2001	2002	2003
----------------	------	------	------	------

Parameter	Command ID	Action type	Not used	Not used
Value	1010	8	0 to 65535	0 to 65535

Type 8 action: THC resume to reactivate the THC that has been suspended by a THCPause action (type 7).

All synchronized actions occupy 4 registers in the Modbus buffer.

Command 1011: Buffered Timeout

Execution of a timeout in the sequence of commands placed in the buffer.

Address	2000	2001
Parameter	Command ID	Duration of delay in ms
Value	1011	0,,65535

Occupation in the Modbus buffer: 2 registers

Command 1012: Wait for State or Event

Put the machine on hold for a state (DIN, AIN or register) or an event. This command is used to stop the execution of buffered commands until a condition is met.

This command supports a timeout which can have the effect of stopping the machine and putting it in an alarm state or letting the process run. The alarm code is 18,

You can for example use this command to control the spindle drive output which indicates that the spindle has reached speed (wait for the state of an input). Another possible use on a plasma cutting machine for detecting sheet metal contact (waiting for a rising edge of the sheet metal probe).

Waiting for the status of a DIN input

Address	2000	2001	2002	2003	2004	2005
Parameter	Command ID	Action type	Timeout (ms)	Action Timeout	Hold Type	DIN number
Value	1012	1	0..65535	0 or 1	1 to 4	0 to 255

Type of action = 1 for waiting for the status of a DIN input

Timeout: Maximum event wait time (in milliseconds)

Action Timeout:

= 0 => continue normally in case of timeout,

= 1 => Stop and alarm in case of timeout, The timeout alarm code is 18

Hold type:

- 1: Waiting for a rising edge on the input
- 2: Waiting for a falling edge on the input
- 3: Waiting for high state
- 4: Waiting for low state

DIN number: Entry number concerned by the wait (0 to 255)

Waiting for the state of an analog input

Address	2000	2001	2002	2003	2004	2005	2006
Parameter	Command ID	Action type	Timeout (ms)	Timeout Action	Hold type	Voltage threshold	AIN number
Value	1012	2	0..65535	0 or 1	1 to 3	0,,10000	0,,7

Type Action = 2 for waiting for the level of an analog input (AIN0 to AIN7)

Timeout: Maximum event wait time (in milliseconds)

Action Timeout:

= 0 => continue normally in case of timeout,

= 1 => Stop and alarm in case of timeout, The timeout alarm code is 18

Hold type:

1: Wait for analog input to fall below a threshold

2: Wait for analog input to be above a threshold

3: Waiting for analog input to equal a Value

Threshold: Expected Threshold or Value expressed in mv

AIN number: Analog input number affected by the wait

Waiting for the Value of modbus information (register or bit)

Address	2000	2001	2002	2003	2004	2005	2006 (LW)	2007 (HW)
Parameter	Command ID	Action type	Timeout (ms)	Action Timeout	Hold type	Treshold	Extended modbus address	
Value	1012	3	0..65535	0 or 1	1 to 4	0,,65535	0,,365535	

Type Action = 3 for Value waiting for modbus information

Timeout: Maximum event wait time (in milliseconds)

Action Timeout:

= 0 => continue normally in case of timeout,

= 1 => Stop and alarm in case of timeout, The timeout alarm code is 18

Hold type:

1: Wait until Value modbus is below a threshold

2: Wait for modbus Value to be above a threshold

3: Waiting for modbus Value to equal a Value

Modbus address to monitor:

The address is indicated in a 32-bit format to allow access to the different types of modbus variables.

For a register of type Input register: Address between 0 and 65535

For a Holding register: Address between 100000 and 165535

For an Input bit type bit: Address between 200000 and 265535

For a Coil bit type bit: Address between 300000 and 265535

Threshold: Expected Threshold or Value. For Values of type Bit, Value of 0 or 1

Part 2: Unbuffered Commands

Command 1100: Edit Override Machining and Rapid Traverse

Immediate action to modify the machining speed (feed rate) and out of material (Rapid Move).

The machining speed override depends on the current machining speed.

You can also read/write these Values directly in Holding registers 2430 and 2431.

Address	2000	2001	2002
Parameter	Command ID	Override machining in %	Override Out of material in %
Value	1005	0..65535	0..65535

Command 1101: Pause machining in progress

Preferably use command 1200 with function code 130

Immediate stop of movements in progress. The buffers are not emptied.

It is also possible to assign a discrete input for pausing machining via parameter 951. You must then indicate an input number from 0 to 255 assigned to this function. To inhibit it, indicate a number of -1.

Address	2000
Parameter	Command ID
Value	1101

Command 1102: Resume interrupted machining

Preferably use command 1200 with function code 129

Resuming the machining in progress

It is also possible to assign a discrete input for resuming machining via parameter 952. You must then indicate an input number from 0 to 255 assigned to this function. To inhibit it, indicate a number of -1.

Address	2000
Parameter	Command ID
Value	1102

Command 1110: Execute a homing machine sequence

Launch of a homing command according to the pre-configured internal parameters.

Address	2000	2001
Parameter	Command ID	Concerned axes
Value	1110	0..31

Axes concerned:

0: launch the entire homing sequence specified in the parameters

1,,6: Specify axis to initialize

Command 1111: Execution of a manual move (Jog)

This command is used to move the axes in relative or absolute mode. It will only be taken into account if the current CNC status is "Idle" or "Jog", Jogs in progress can be interrupted by the direct command "Stop Jog" (command 120 0, function 133).

You can combine the simultaneous movements of several axes. The indicated speed will be the combined speed of the different movements.

As with linear movement commands, the length of the frame depends on the number of axes to be moved.

The axis positions must be given in ascending order of axis indexes.

Example, to move the X and Z axes, Indicator will be 5, the positions will be given with Position X then Position Z.

If bit 6 of the indicator of the axes to be moved is at 0, the positions will be absolute positions. If the bit is 1, it will be a relative position.

For manual movements, it is recommended to use this Jog command in incremental mode. In the event of a communication breakdown, movements will be automatically limited.

Address	2000	2001	2002	2003	2004	2005	2006	2007	...
Parameter	Command ID	Indicator	Float	Float	Float	Float	Float	Float	...
Value	1111	0x01 to 0x7F	Speed (m/mn)	Position 1 (mm)	Position 2 (mm)	Position 3 (mm)	Position 4 (mm)	Position 5 (mm)	...

Detail of the Indicator parameter:

- Bit 0 => The target of the X axis is indicated in the frame
- Bit 1 => The Y axis target is indicated in the frame

- Bit 2 => The Z axis target is indicated in the frame
- Bit 3 => The target of axis A is indicated in the frame
- Bit 4 => The B axis target is indicated in the frame
- Bit 5 => The C axis target is indicated in the frame
- Bit 6 => 0 → Absolute positions; 1->Relative positions

Command 1200: Direct execution of a command

This command allows you to act immediately on the operation of the CNC. It is used for real-time control over various parameters and the operating status of the machine.

Address	2000	2001
Parameter	Command ID	Function code
Value	1200	0..255

Possible subcommand values:

Function code	Action	Details
88	Alarm acknowledgment	
129	Machining recovery	Resuming machining after pausing
130	Machining pause	Pausing machining, spindle stop The buffers are not emptied.
133	Stop Jog	Stopping a running jog motion
138	Cancel overspeed machining	Machining overspeed returns to 100%
143	Cancel fast overspeed	Out of material overspeed returns to 100%
153	Annuler survitesse broche	Spindle overspeed returns to 100%
255	Immediate stop	CNC motion stop with ramp deceleration. If THC is active, it is stopped, Command and planning buffers are also purged.

Using the internal clock (RTC)

The InterpCNC has an internal clock to manage the date and time. However, this clock is not saved when the power is turned off. It should therefore be initialized before using its functions.

Initialization can be done by:

- Modbus commands 112, 113 and 114,
- The PLC program using the RTC command,
- Automatically via an SNTP server if the InterpCNC has internet access.

For automatic update by STNP, parameters 546 and 547 must be set correctly. The SNTP server used is “sntp.pool.org”.

The clock will then be initialized taking into account the time zone indicated in parameter 547 and summer/winter time if bit b1 of parameter 547 is active.

You have 2 status bits that allow you to determine the synchronization status:

stsBit(STS_RTC_SYNCHRONIZED) which indicates that the clock has been set,

stsBit(STS_SNTP_CONNECTED) which indicates that an SNTP connection is established.

SNTP server synchronization, if enabled, is automatically renewed every hour.

It is possible to read the different clock registers (time, date) in the Input Registers 1987 to 1995.

In the PLC program, in the case of read-only registers, 100000 should be added to the address for reading with the GetMW command. To read these registers, the commands will therefore be GetMW(101987) to GetMW(101995)

Note for reading through modbus registers:

To obtain consistent data, reading the Input Register 1987 (RTC Time) causes the creation of a buffer storing the current date and time. This buffer remains valid for at least 100ms. The ideal is therefore to read all the information required in this period of time.

Numerous PLCBasic functions are also available to exploit the RTC clock. They are detailed in the specific PLCBasic documentation.

Command 112: Set Date on RTC Clock

This command is used to set the date of the PLC's internal clock.

Address	2000	2001	2002	2003
Parameter	Command ID	Day	Month	Year
Value	112	1,,31	1,,12	0,,99

Command 113: Set Time to RTC Clock

This command is used to set the time of the PLC's internal clock.

Address	2000	2001	2002	2003
Parameter	Command ID	hours	minutes	seconds
Value	113	0,,23	0,,59	0,,59

Command 114: Simultaneous setting of date and time on the RTC clock

This command is used to simultaneously set the date and time of the PLC's internal clock.

Address	2000	2001	2002	2003	2004	2005	2006
Parameter	Command ID	Day	Month	Year	Hours	Minutes	Seconds
Value	114	1,,31	1,,12	0,,99	0,,23	0,,59	0,,59

Miscellaneous settings

This chapter deals with advanced features, offered by both hardware and firmware of the card, and easily configurable from ICNCStudio.

Password protection

In some cases, and especially for industrial applications, password protection is necessary.


This allows you to protect your PLC Basic program against reading, against writing, or both.

This allows for Example to prevent the use of a machine with a program other than the one with which it was delivered, or even to recover the embedded program to use it on another machine of which it would have been made an unauthorized clone.

Paramètres Automate

Mot de Passe PLC program Auto RUN

Lecture protégée

Écriture protégée 

Horloge temps réel

Synchro sur serveur SNTP Fuseau horaire GMT-1

Heure été/hiver automatique

Configuration afficheur OLED

Luminosité Rotation 180°

Sortie PUL/DIR en TTL

Rediriger OUT16 à OUT27 sur des sorties TTL

Card lock

- Choose and check the **Protected Read** ("Lecture protégée") and/or **Protected Write** ("Écriture protégée") option
 - Choose a password, and click on Validate
 - > a new window asks you to confirm the chosen password.
- Then click on **Send to PLC**. The closed padlock symbol appears.
For more details, see [Password Protection](#), in the General Configuration chapter.

Unlocking the card

Click on the padlock, and enter your password.

- Uncheck the lock options, and click on **Send to PLC**

NB: the password chosen previously remains in automatic entry not readable in its entry field, even if the card is unlocked.

Warning: If your password has been permanently forgotten, in this case you will have to contact the SOPROLEC company to obtain an unlocking code which will be generated specifically for your card.

Trick

The **Load from PLC button** (or type **reboot** in the command line of the Basic editor) can allow you to refresh the state if you no longer know where you are.

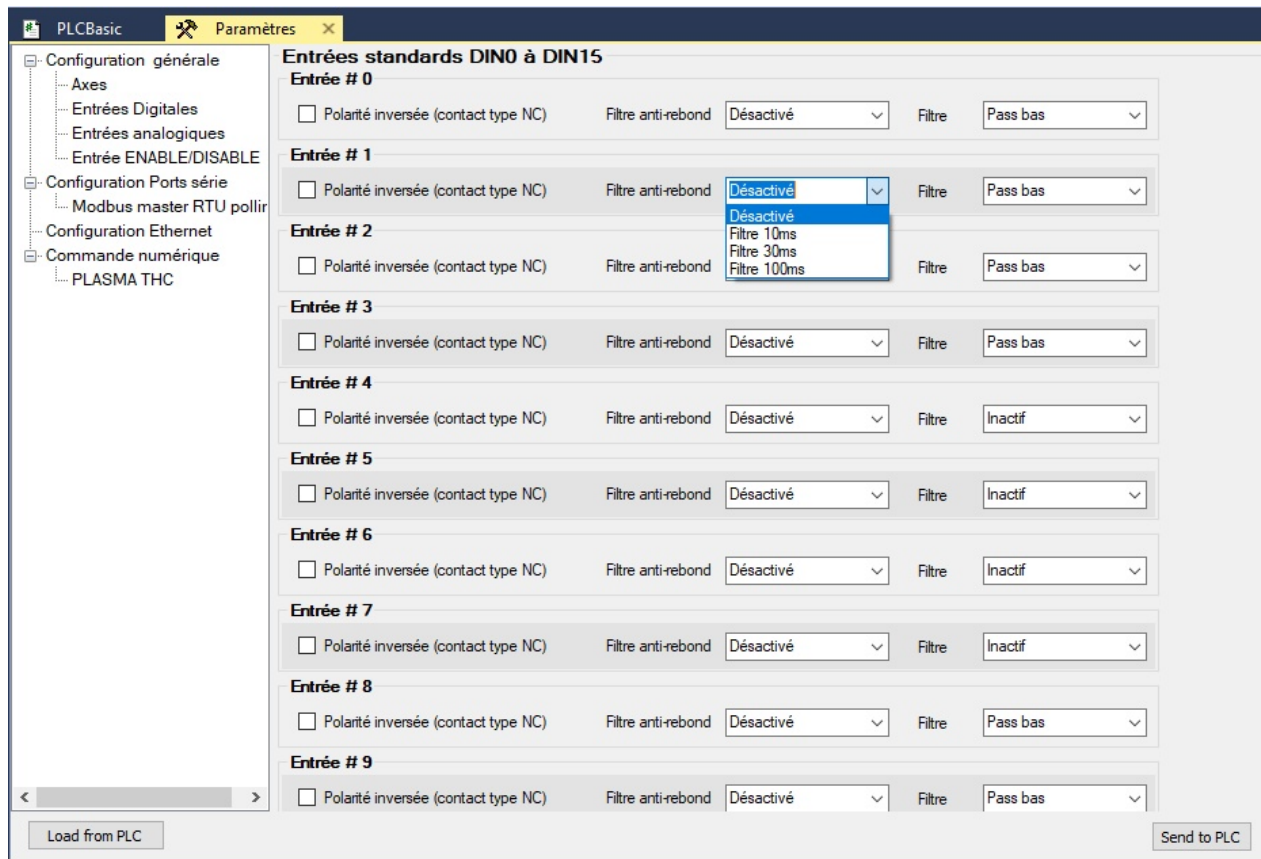
Standard inputs

For each input from DIN0 to DIN15, it is possible to reverse its polarity.
It is also possible to adjust its sensitivity, using 3 levels of Debounce filter ('Filtre anti-rebond') : 10, 30, or 100ms

This filter can use the filtering level selected according to 3 modes:

Off, Low Pass, or Sampling.

For the choice of the **Debounce** mode see the explanation given in the [DIN section of the table of parameters](#).



Fast inputs

1 Use of inputs in Encoder Mode (17 to 22)

We can have 3 encoders because 2 fast inputs are required for each of the encoders.
 Channel 0: inputs 21 and 22. Frequency up to 1Mhz
 Channel 1: inputs 19 and 20. Frequency up to 50 khz
 Channel 2: inputs 17 and 18. Frequency up to 50 khz
 Card parameters 221 to 226 must be configured in Mode 4 (4X, all edges are taken into account), or in Mode 3 (2X, 1 edge out of 2 is taken into account).

GetEncoder(: Returns the position of an encoder input

Syntax: **GetEncoder**(Channel) 'Channel: 1 to 3

Example: SetMDW 3018, **GetEncoder**(3)

SetEncoder: Assignment of a Value to an encoder input

Syntax: **SetEncoder** Channel, Value 'Channel: 1 to 3.

Example: **SetEncoder** 3, 0' Set encoder input #3 to 0

2 Using Inputs in Counter Mode (16 to 22)

We can therefore have 7 counters. Card parameters 220 to 226 are to be configured:

In mode 0 (standard), the refreshing of the table storing the state of the inputs takes place every millisecond.

In mode 1 (on "IT" interrupt), the refresh of the array storing the state of the inputs takes place on each interrupt accessing it.

In mode 2 (Counter mode), the refresh of the table storing the state of the inputs takes place on each edge (rising or falling, depending on the configuration of the polarity of the inputs (card parameter 200)).

GetCnt(: Read one of the counters associated with fast inputs 16-22.

The result is an unsigned integer.

Syntax: **GetCnt**(CounterNo)

CounterNo = 1 to 7

Example: SetMDW 3018, **GetCnt**(4) 'Writes to address 3018 the Value read from counter #4

SetCnt: Write to the counters of fast inputs 16 to 22. The InterpCNC has 7 fast inputs which can be used as count inputs.

Syntax: **SetCnt** CounterNo, Value

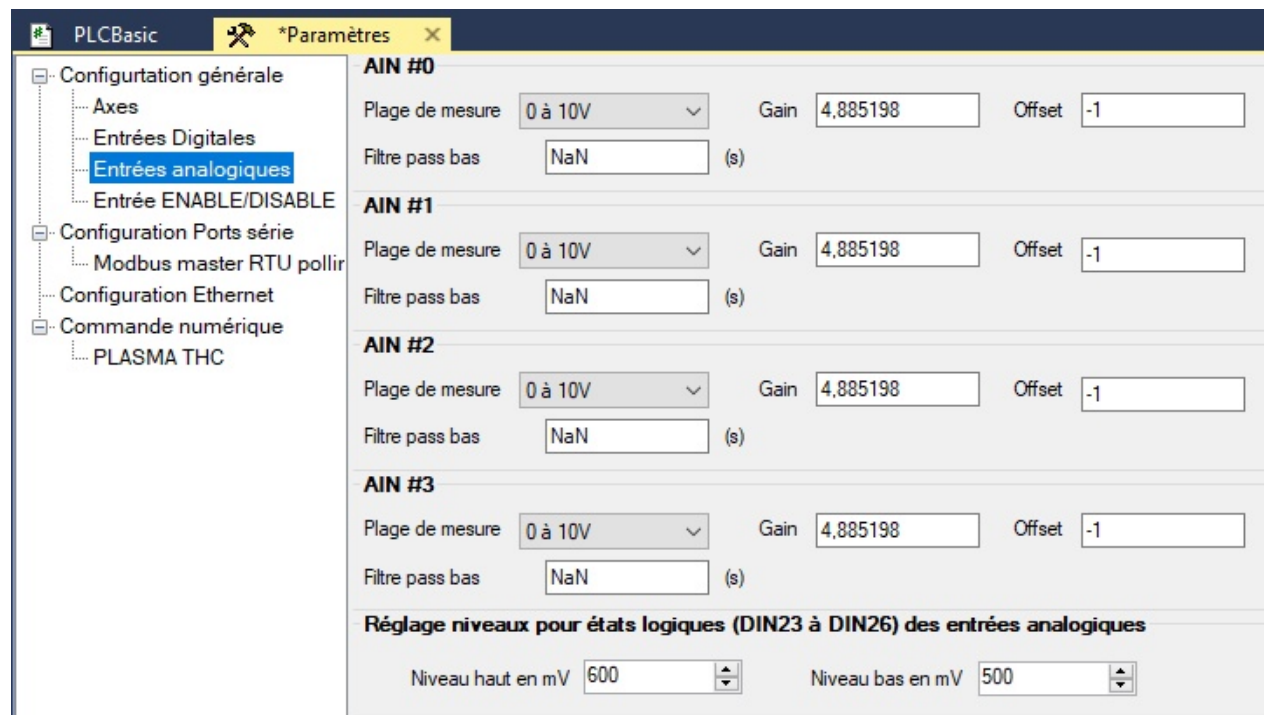
Counter No = 1 to 7

Value = Counter value

Example: **SetCnt** 4, 0 'Resets counter 4 to 0

Using Analog inputs as Digital inputs

If more "on/off" digital inputs than the 24 available are required, it is also possible to use the analog inputs. These will be mapped as DIN23 to DN26.



A parameter (**312**) is used to define the level from which a high level is considered (600 mV by default)

A parameter (**313**) is used to define the level below which a low level is considered (500 mV by default)

The Values are entered directly in mV (from 0 to 10000).

The polarity of these inputs can also be reversed, for example for the use of Normally Closed type contacts.

To do this, check the corresponding option on the [Digital Inputs](#) ("Entrée Digitales") page, from the above "**Parameters**" menu.

I/O expansion modules

Card configuration for use of a Kinco I/O expansion module:

The mapping system (also called polling) makes it possible to transmit with adjustable refresh rate, **Modbus** frames to control and exchange data with all types of peripherals (variators, drivers, HMI, PLC, etc.)

Example with a Kinco I/O expansion module:
the **KS123-14DR** -> 6 outputs/relays and 8 inputs, connected to COM2



RS485 Modbus connection:

D+ on A
D- on B

1) In the card parameters table, configure COM2:
420 -> 2 for Master (InterpCNC V3 card is Master)

- 421 -> Baud Rate (ex: 38400)
- 422 -> 8 (data bits)
- 423 -> 0 (Parity)
- 424 -> 1 (Stop Bits)
- 425 -> 1 (Modbus Slave ID)

Configuration COM 2		
420	COM2 Mode : 0=None, 1=Slave, 2=Master	2
421	COM2 baud rate	38400
422	COM2 Bits de données	8
423	COM2 Parité	0
424	COM2 Bits de stop	1
425	COM2 Modbus Esclave ID	1

2) Redefine a mapping for the inputs, and a mapping for the outputs:

Here, the state of the 6 module outputs will be controlled by the state of the 6 Coils (Modbus bits) n° 110 to 115.

The 8 inputs are mapped to virtual inputs 32 to 39 (DIN).

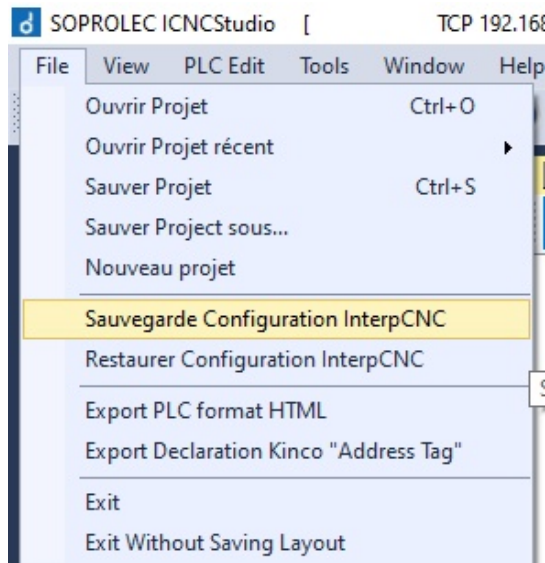
This state will be refreshed by a reading every 50ms.

NB: The Slave ID is the modbus address of the module (it can be redefined using a small Kinco utility).

Of course, this type of module does not make it possible to benefit from inputs and outputs as fast as those of the InterpCNC V3 (especially since the outputs of these modules are on relays), but in certain applications this solution can be interesting because you are no longer limited to 16 digital inputs and 16 digital outputs.

Exports and Imports of files

On the **File** tab of the Menu bar:



→ The option « **Sauvegarde Configuration InterpCNC** ». This saves the following data in the form of an .ini file:

- Map Settings export.
- Export of Saved Memory (EEprom)
- Export Recipes

The option « **Restaurer Configuration InterpCNC** » allows, as its name suggests, to reload from the .ini file, the parameters and registers from a previous backup.

All or part of these 3 sections can be saved or restored (options to be checked).

→ The **Export Declaration Kinco Address Tag**, is a very useful function that generates a .csv file associating the name (Tag) assigned to each Bit or Register used in our PLCBasic program, with its Modbus address and its type of access (read, read/write: 0X, 1X, 3X, 4X, etc.).

This file is then extremely practical for the design of HMI screens associated with your PLCBasic program, with DTools or HMIWare software from the Kinco brand. Then just import your Address Tag .csv file with the "Import Address Tag" command.

In your design project, you then have the pages of your HMI, the same naming of the bits and registers, and the Modbus addresses used by the card. The development of HMI Kinco screens is thus really facilitated.

On the **PLC Edit** tab:

→ The option **Export Programme Analysé**. This function is used to save in .txt format, your PLCBasic program in raw format, ie without comments and where each constant name is replaced by its numerical Value. This .txt file can then be opened, and a 'Copy/Paste' of its contents to the PLC program window gives an overview of the program as it is executed by the card's Basic interpreter.