



VTB Software Resources



The contained information in this handbook are only informative and they can being change without warning and they must not being understandings with some engagement from Promax srl. Promax srl does not assume responsibility or obligates for errors or inaccuracies that can be found in this handbook. Except how much granted from the license, no part of this publication can be reproduced, saved in a recording system or transmitted in whatever form or with any means, electronic, mechanical or recording system or otherwise without Promax srl authorization. Any reference to names of society or products have only demonstrative scope and it does not allude to some real organization.

Rev. 1.0.0

© Promax s.r.l. – Via Newton, 5/G – Z.I. Malacoda – CastelFiorentino (Fi) ITALY
email:info@promax.it - internet:www.promax.it

1 Preface

This document is referred to NGQ Board HARDWARE resources usable with VTB language
For more details to VTB language, see the following links:

Programming Guide
Objects Guide

The following examples, are not referred to real applications

2 RS232/RS485 Port

The NGQ allows to use 1 RS232/485 port, with a custom or standard (MODBUS RTU) protocols.

2.1 SER_SETBAUD

Programming the BaudRate of the second SERIALE PORT - SER2.

Syntax

SER_SETBAUD (Baud **as long**) **as void**

Parameters

Baud Value of Baud Rate. The standard value are:
1200-2400-4800-9600-19200-38400-57600-115200

2.2 SER_MODE

Programming the mode of the second SERIAL PORT. If this function is never called, by default the port is programmed with:

No parity
8 bit per character
1 bit stop.

Syntax

SER_MODE(par **as char**, nbit **as char**, nstop **as char**) **as void**

Parameters

par Parity (0=no parity, 1=odd parity, 2=even parity)
nbit Number of bits per character (7 or 8)
nstop Number of stop bits (1 or 2)

2.3 SER_GETCHAR

Reads the receive buffer of the serial port. It doesn't wait for the presence of a character. This function, must be calling, in POLLING by VTB application
The operating System, manages the INTERRUPT BUFFER

Syntax

SER_GETCHAR () **as int**

Return Value

int **-1** No character is in the buffer
>=0 Code (0 to 255) of the character read from the buffer

2.4 SER_PUTCHAR

ISends a character to the serial port.

Syntax

SER_PUTCHAR (Car **as int**) **as void**

Parameters

Car Code (0 to 255) of the character to send

2.5 SER_PUTS

Sends a string of characters to the serial port. The string must be ended with the character 0 (NULL).

Syntax

SER_PUTS (str **as *char**) **as void**

Parameters

***str** String Pointer

2.6 SER_PRINTL

SFormatting print of an INTEGER value.

Syntax

SER_PRINTL (format as **char*, val as *long*) as *void*

Parameters

Format String corresponding to the format to be printed

Val Any integer value or expression

Available formats

#####	Print a fixed number of characters	23456
###.###	Force the print of decimal point	123.456
+####	Force the print of the sign	+1234
#0.##	Force the print of a ZERO	0.12
X####	Print in HEXADECIMAL format	F1A3
B####	Print in BINARY format	1011

2.7 SER_PRINTF

Formatting print of a FLOAT value. It is the same as *ser_printl* but use a float value

Syntax

SER_PRINTF (**const char** *format, val as *float*) as *void*

Parameters

Format String corresponding to the format to be printed

Val Any integer value or expression

2.8 SER_PUTBLK

Sends a precise number of characters to the serial port. Unlike the function *ser_puts* it allows to send also the character with 0 code enabling the managing of binary protocols, furthermore it starts the background transmission setting in appropriate mode the RTS signal useful to work with RS485 lines.



Syntax

SER_PUTBLK (Buffer as **char*, Len as *int*) as *void*

Parameters

***Buffer** Pointer to the data buffer to send

Len Number of bytes to send

2.9 SER_PUTST

Reads the state of background transmission started by *ser_putblk*.

Syntax

SER_PUTST () as *int*

Return Value

int **-1** Transmit error

>=0 Number of characters to be transmitted

2.10 Example

In the following example, is call the Read_Data() function, in polling in the Task Main SER2 Setting:

Baud rate → 115,200
Nr. bit dati → 8
Nr. bit Stop → 1
Parità → NO

Response value:

Character received =1 → Echo charatcer reveived (1) with *ser_putchar*
Character received =2 → Send Text "Test String" with *Ser_puts*
Character received =3 → Formatted Print Variable **Num** (number of characters received)
Character received =4 → Formatted Print Variable **NumFloat** (Float random)
Character received =5 → Send in Binary mode Nr. **789488** with *Ser_putblk*
Character received =6 → Test state *Ser_putblk* - reply:
 255 send data error
 Nr characters in the transmission buffer
Character received=Others → Response **254** - Error unknown command

Variables used

Internal VAR	Bit VAR	Define	Static VAR	VSD VAR	Fixed VAR
			No	EXP	<input type="checkbox"/>
Variable	Type	Shared	Export in Class		
String(20)	CHAR	No			
Num	LONG	No			
NumFloat	FLOAT	No			
Ret_fn	CHAR	No			

Code in Init Main

```

Page Init | Master Event | Master Cycle | Page Functions
1  ser_setbaud(115200) ' set baud 115200

```

`ser_setbaud(115200) ' set baud 115200`

Code in Master Ciclo Main

```

Page Init | Master Event | Master Cycle | Page Functions
1  Read_Data() 'Read data from RS232

```

`Read_Data() 'Read data from RS232`

Code in Page Functions Main

```

Page Init | Master Event | Master Cycle | Page Functions
1  *****
2  'Read Data From RS232
3  *****
4  function Read_Data() as void
5  dim Ret as int
6  Ret=Ser_getchar() ' Reda one char from RS232 buffer
7  if Ret=-1 ' none
8      endfunction ' return
9  endif

```

```
'*****  
'Read Data From RS232  
'*****  
function Read_Data() as void  
  
Ret_fn=Ser_getchar() ' Read one char from RS232 buffer  
if Ret_fn=-1 ' none  
    endfunction ' return  
endif  
inc Num ' increases the received chars  
NumFloat=Num*2.13 'random number  
'process data received  
select Ret_fn  
    case 1 ' ----- echo char with send_putchar  
        Ser_putchar(Ret_fn) ' send reply echo char  
    case 2 ' ----- send string with ser_puts  
        strcpy(String(),"Test String") ' Copy in array string text  
        ser_puts(String()) ' put data  
    case 3 ' ----- print a long formatted with ser_printl  
        ser_printl("###.##",Num) ' print ex: 123.45 format  
    case 4 ' ----- print a float formatted with ser_printf  
        ser_printf("###.###",NumFloat) ' print NumFloat  
    case 5 ' ----- put a block with ser_putblk  
        'Send a number 789488  
        String(0)=0xF0 'LSB  
        String(0)=0x0B  
        String(0)=0x0C  
        String(0)=0 'MSB  
        Ser_putblk(String(),4) ' Data len 4 byte  
    case 6 ' ----- test if ser_putblk is busy  
        Ret_fn= Ser_putst() ' check if function ser_putblk is busy  
        if Ret_fn=-1  
            Ser_putchar(255) ' send error  
        else  
            Ser_putchar(Ret_fn) ' send number of chars  
        endif  
    case else  
        Ser_putchar(254) ' send error no char  
endselect  
  
endfunction
```

[Example Download](#)

3 Modbus RTU

The SER2 port, is able to manage the RTU MODBUS protocol.
The protocol MODBUS RTU is available in two configuration:

Master

Slave

3.1 Modbus RTU Slave Object

This Object, manage, the RTU MODBUS SLAVE protocol.

Property

Nodo	Node slave
BaudRate	baud rate
PtData()	Array Data Register in the NGQ memory
Max Len Data	Data Register dimension
TimeOut	Master Time Out (millisecond) This must be smallest by a MASTER TimeOut

Methods

No

The following requests are handled MODBUS RTU:

Function Code 3	Read Multiple Registers
Function Code 6	Preset Single Registers
Function Code 16	Preset Multiple Registers

Events

No

3.2 Example ModBus slave

In the next example, are read and written, some registers 16 bit declared in the NGQ memory.
The registers array is named **Data**, and the maximum dimension, is in the **DEFINE MAX_DATA**

Where :

Read/Written from Modbus register Nr.1 → **Data(0)**

Read/Written from Modbus register Nr.2 → **Data(1)**

etc.

The example, Read the data register Nr. 2 - Data(1) and written the Data register Nr1 - Data(0)

Objects used:



Modbus → **Cmodbus** → **ModBus Protocol**

Project Explorer				
Project	Objects	Functions	Properties	Tables
Modbus1				
Property Events				
Property	Value			
Name	Modbus1			
Left	15			
Top	15			
Modbus Node	1			
Baudrate	19200			
Pt Data	data()			
MAX Len Data	MAX_DATA			
TIME OUT	100			

Variables used

Internal VAR	Bit VAR	Define	Static VAR	VSD VAR	Fixed V
				No	EXP <input type="checkbox"/>
Variable	Type	Shared	Export in Class		
Data(MAX_DATA)	CHAR	No			

DEFINE Used

Internal VAR	Bit VAR	Define	Static VAR
Variable	Type		
MAX_DATA		100	

Code in Master Ciclo Main

```

Page Init | Master Event | Master Cycle | Page Functions
1 | '*****
2 | ' Sample code
3 | '*****
4 | select Data(1)
5 |     case 100
6 |         Data(0)=1
7 |     case 200
8 |         Data(0)=2
9 | endselect
    
```

```

'*****
' Sample code
'*****
select Data(1)
    case 100
        Data(0)=1
    case 200
        Data(0)=2
endselect
    
```

Example Download

3.3 Modbus RTU Master Object

This Object, manage, the RTU MODBUS MASTER protocol.

Property

BaudRate	Comm Baud rate
TimeOut	Time Out for SLAVE response (millisecond). This must be more great by a slaves TimeOut
Parita	0 none - 1 odd - 2 even
N. bit car	Number bit for char
N. bit stop	Number stop bit

Methods

function .write_regn(nodo as char, addr as uint, value as *int) as char

Preset single register func 16 ModBus RTU

Parameters

nodo	Node slave modbus
addr	Start Address register to write (Slave)
Value	Unsigned integer (values to write)

Return

0	Write OK
1	Error respons
2	Time Out
3	Data len > 127

function .read_regn(nodo as char, addr as uint, value as *int) as char

Read single register func 3 ModBus RTU

Parameters

nodo	Node slave modbus
addr	Start Address register to read (Slave)
Value	Pointer to unsigned integer (value to read)

Return

0	Read OK
1	Error respons
2	Time Out
3	IData len > 127
4	Checksum error

3.4 Example ModBus Master

In the next example, are read and written, some registers 16 bit in a slave

Objects used:



Modbus → **CmodbusMaster** → **ModBus Master Protocol**

Project Explorer				
Project	Objects	Functions	Properties	Tables
ModbusMaster1				
Property Events				
Property	Value			
Nome	ModbusMaster1			
Left	25			
Top	15			
Baudrate	19200			
TIME OUT	100			
Parità	0			
n° bit Car	8			
n° bit Stop	1			

Variables used

Internal VAR	BIT VAR	Define	Static VAR	VSD VAR	Fixed VAR
			No	EXP	
Variable	Type	Shared	Export in Class		
RegModbus	UINT	No			
Valret	CHAR	No			

Code in Main Page Functions

```

Page Init | Master Event | Master Cycle | Page Functions
1 .....
2 ' Raed data from node 1
3 ' register 10 in RegModbus variable
4 .....
5 function Read_Data_Node_1() as void
6 Valret=modbusmaster1.read_reg(1, 10, regmodbus())
7 .....

```

```

' *****
' Raed data from node 1
' register 10 in RegModbus variable
' *****
function Read_Data_Node_1() as void
Valret=modbusmaster1.read_regn(1, 10, regmodbus())
if Valret>0
    ' read error
endif
endfunction
' *****
' Write data to node 1
' register 10 RegModbus variable
' *****
function Write_Data_Node_1() as void
RegModbus=100
Valret=modbusmaster1.write_regn(1, 10, RegModbus)
if valret>0
    ' write error
endif
endfunction

```

[Example Download](#)

4 Analog Inputs Read

The NGQ board, has 4 analog inputs managed by VTB functions
In this Board revision, the analog inputs, have a 12 bit resolution (value from 0 to 4095)

4.1 Inputs Read

Syntax

NG_ADC(Channel as Char) as uint

Parameters

Channel Channel number (from 0 to 3)

Return Value

Returns the analog value (from 0 to 4095)

Where 0 is the minimum voltage level (0 volt) , 4095 is the maximum voltage level configured in the input (normally 10 Volt)

4.2 Example Analog inputs read

In the following example, are read the analog inputs from 0 to 3. the values are written in the array AnalogValues

The channels are read in TaskPlc

Variables used

Internal VAR	Bit VAR	Define	Static VAR	VSD VAR	Fixed VAR
			No	EXP	<input type="checkbox"/>
Variable	Type	Shared	Export in Class		
AnalogValues(8)	UINT	No			
NumCh	INT	No			

Code in Init TaskPlc

```
TASK PLC Code
Init Task PLC Task PLC
1 NumCh=0 ' reset number channel to read
```

NumCh=0 ' reset number channel to read

Code in Task Plc

```
TASK PLC Code
Init Task PLC Task PLC
1 '*****
2 ' Read The channel
3 '*****
4 AnalogValues (NumCh)=ng_adc (NumCh)
```

```
'*****
' Read The channel
'*****
AnalogValues (NumCh)=ng_adc (NumCh)
inc NumCh 'increase channel number
if NumCh=4 ' limit
    NumCh=0
endif
```

Example Download

5 CanOpen Management

The NGQ board, can use, one Canopen line Master/Slave

The MASTER port, can use the PDO programmable by CanOpen configurator.(see Canopen Configurator - [Link Chapr. 14](#))

5.1 PXCO_SDO DL

This function allows to send data to a node of the canopen net using the protocol SDO. It is supported only the SDO EXPEDITED mode allowing to send up to 4byte of data length.

Syntax

PXCO_SDO DL(node **as char**, index **as int**,subidx **as uchar**,len **as long**,data **as *char**) **as char**

Parameters

Node	Node ID of the SLAVE to which send data
Index, subindex	Address in the Object-Dictionary of the data to be written
Len	Number of bytes to send
*data	Pointer to the data to send

Return value

char	0	No error
	<>0	Communication error
	=2	The node responded with a SDO ABORT CODE, calling the function read_sdoac in the system variables _SYSTEM_SDOAC0 e _SYSTEM_SDOAC0 will be available the relative error code.



5.2 PXCO_SDO UL

This function allows to read data from a node of the canopen net using the protocol SDO. It is supported only the SDO EXPEDITED mode allowing to read up to 4byte of data length.

Syntax

PXCO_SDO UL(node **as char**, index **as uint**,subidx **as uchar**,dati **as *char**) **as char**

Parameters

Node	Node ID of the SLAVE to which send data
Index, subindex	Address in the Object-Dictionary of the data to be written
*data	Pointer to the data to send

Return value

char	0	No error
	<>0	Communication error
	=2	The node responded with a SDO ABORT CODE, calling the function read_sdoac in the system variables _SYSTEM_SDOAC0 e _SYSTEM_SDOAC0 will be available the relative error code.



5.3 READ_SDOAC

Reading of the SDO ABORT CODE sent by a node in the canopen net as answer to a request done with the function PXC0_SDODL or PXC0_SDOUL. The read code will be written in the system variables _SYSTEM_SDOAC0 e _SYSTEM_SDOAC1.
Refer to the DS301 specific of the CAN OPEN for the code error values.

Syntax

READ_SDOAC() as void

5.4 PXC0_SEND

Sending of a CAN frame at low level. This function allows to send in the net a CAN frame with a desired COB-ID and DATS. For example it's possible to send manually PDO frames, HEART-BEAT frames, etc.

Should be specified the manage of PDO is managed AUTOMATICALLY by the CANOPEN CONFIGURATOR.

Syntax

PXC0_SEND(id as int, Len as char, Dati as *char) as char

Parameters

id COB-ID value
Len Number of data to send
***Data** Pointer to the data buffer

Return value

char 0 No error
<>0 Communication error

5.5 PXC0_NMT

Sending of a NMT frame of the CAN OPEN. NMT protocol allows to set the state of the nodes in the net. Remind that all the nodes correctly configured (canopen configurator) are automatically set in START state.

Syntax

PXC0_NMT(state as char, node as char) as char

Parameters

state State to set:
1 = START NODE
2 = STOP NODE
128 = PRE-OPERATIONAL
129 = RESET NODE
130 = RESET COMUNICATION
node Number of the node

Return value

char 0 No error
<>0 Communication error



5.6 READ_EMICY

Reads the last EMERGENCY OBJECT frame sended by a CAN OPEN node.

The emergency code is written in the system array `_SYSTEM_EMICY(8)` and it will contain all the 8 bytes of the EMERGENCY OBJECT frame as from the DS301 specific of the CAN OPEN. Usually it is called cyclically. The emergency code depends by type of connected device, therefore refer to its manual.

Syntax

`READ_EMICY()` as char

Return Value

char 0 No error
 <>0 Node that generated the emergency object.

_SYSTEM_EMICY							
0	1	2	3	4	5	6	7
Emergency Error Code		Error Register		Manufacturer specific Error Code			



WARNING

THE SYSTEM DOESN'T BUFFER MORE THAN ONE MESSAGE, THEN IF MORE EMERGENCY OBJECT ARE SENDED ALONG A SINGLE TASK PLC, ONLY THE LAST WILL BE READ.

AN EMERGENCY OBJECT DOES NOT MEAN THAT THERE IS ACTUALLY A NODE IN AN EMERGENCY. THE DS301 SPECIFIC PROVIDE THAT AN EMERGENCY OBJECT ARE SEND ALSO ON ALARM RESET. FURTHERMORE SOME DEVICES CAN BE SEND THIS FRAME AT START UP.

5.7 Example CanOpen Functions

In the following example, are used the Canopen Functions.

Variables used

Internal VAR	Bit VAR	Define	Static VAR	VSD VAR	Fixed VAR
				No	EXP <input type="checkbox"/>
Variable	Type	Shared	Export in Class		
Value	INT	No			
Ret	CHAR	No			
Restart	CHAR	No			

Code in Master Ciclo Main

Page Init	Master Event	Master Cycle	Page Functions
1	Sdo_Dl()	' Sdo Download	
2	Sdo_Ul()	' Sdo Upload	
3	Send_Pdo()	' send pdo	
4		'check if restart node 1	
5		if Restart=1	

```

Sdo_Dl() ' Sdo Download
Sdo_Ul() ' Sdo Upload
Send_Pdo() ' send pdo
'check if restart node 1
if Restart=1
    Restart=0 ' reset flag restart
    Ret=pxco_nmt(1,1) ' Start Node
    if Ret<>0 'test error
        '...
    endif
endif

'polling emergency object
Ret=Read_emcy()
if Ret<>0
    ' in Ret node error
    ' in _SYSTEM_EMCY code error
endif
    
```

Code in Main Page Functions

Page Init	Master Event	Master Cycle	Page Functions
-----------	--------------	--------------	----------------

```

1  '*****
2  ' Sdo Download function
3  ' send the value 100 at:
4  ' Node 1
5  ' Index 0x2000
6  ' Subindex 0
7  '*****
8  function Sdo_Dl() as void
9  Value=100
10 Ret=pxco_sdodl(1,0x2000,0,2,Value()) 'node=
11                               'len=2 byte, value=100
'*****
' Sdo Download function
' send the value 100 at:
' Node 1
' Index 0x2000
' Subindex 0
'*****
function Sdo_Dl() as void
Value=100
Ret=pxco_sdodl(1,0x2000,0,2,Value()) 'node=1, index=0x2000, subidx=0,
                               'len=2 byte, value=100

if Ret<>0 'test error
    if Ret=2
        read_sdoac() 'Read SDO ABORT CODE
        'in _SYSTEM_SDOAC0 code error
        'in _SYSTEM_SDOAC1 code error
    endif
    '...
endif
endfunction

'*****
' Sdo Upload function
' read the value at:
' Node 1
' Index 0x2000
' Subindex 0
'*****
function Sdo_Ul() as void
Ret=pxco_sdoul(1,0x2000,0,Value()) 'node=1, index=0x2000, subidx=0,
                               'read in value

if Ret<>0 'test error
    if Ret=2
        read_sdoac() 'Read SDO ABORT CODE
        'in _SYSTEM_SDOAC0 code error
        'in _SYSTEM_SDOAC1 code error
    endif
    '...
endif
endfunction

'*****
' Send PDO
' COB - ID = 0x201
' 2 Bytes
    
```


VTB Software Resources NG QUARK

```
' SVariable in Value
'*****
function Send_Pdo() as void
Value=100
Ret=pxco_send(0x201,2,Value()) 'cob-id=0x201) 2 bytes
if Ret<>0 'test error
'....
endif
endfunction
```

[Example Download](#)

5.8 Example CanOpen Axes interpolation mode

In the following example, are managed 3 CanOpen Axes in linear interpolation.

ATTENTION:

All speed are managed in mm/min if setted the following parameters

RapX,RapY,RapZ

All axes target positions are managed in micron (0.001 mm) if setted the following parameters

RapX,RapY,RapZ

Objects used



Motor Control → **CobjInterpola** → **Interpolatore**

Project Explorer	
Project	Objects
Interp	
Property	
Property	Value
Nome	Interp
Left	15
Top	10
N.assi	3
N.tratti	16
Vper	1024
Div. Vper	1024
Abilita arcto	1

Motor Control → **CstdCanOpen** → **Ds402 x 3**

Project Explorer	
Project	Objects
AxisX	
Property	
Property	Value
Name	AxisX
Left	10
Top	85
Node	1
Mode	0
Speed	0
Position	0
Abs	True
State	False
home_delay	1000

Project Explorer	
Project	Objects
AxisY	
Property	
Property	Value
Name	AxisY
Left	55
Top	85
Node	2
Mode	0
Speed	0
Position	0
Abs	True
State	False
home_delay	1000

Project Explorer	
Project	Objects
AxisZ	
Property	
Property	Value
Name	AxisZ
Left	100
Top	85
Node	3
Mode	0
Speed	0
Position	0
Abs	True
State	False
home_delay	1000

VTB Software Resources NG QUARK

Are managed the following functions:

Wait_Move – Axes state movement

Parameters No
Return 1 Axes in movement
0 Axes stop

Move_Axes – Move the Axes in linear interpolation

Parameters Vel → Feed Axes in mm/min
Flg → Set to 1 for disable the movements buffer
(Stop axes at end trajectory)
Set to 0 for enable the movements buffer
(Stop Axes only if edge > SGLP)
Px,Py,Pz → Axes target values in 0.001 mm
Return 0 Movement inserted in the buffer – buffer empty
1 Buffer full (you must repeat Move_Axes up to when buffer empty)

Acc_Axes – Set interpolation Acceleration

Parameters Value → Value in count per TAU
Return No

Stop_Axes – Stop Axes

Parameters No
Return No

Enable_Axis_X_Y_Z – Enable the Axes control and preset at value 0

Parameters No
Return No

Disable_Axis_X_Y_Z – Disable the Axes control

Parameters No
Return No

cancfgerr – CanOpen Custom Error. This function is called at Canopen Node init (node setted in configuration by Canopen configurator) when the node, reply error

Parameters Node → Node number in error
Err → Error code
Return No

Close_cancfgerr - CanOpen Custom Error. This function is called at end Canopen nodes configuration

Parameters No
Return No

Open_cancfgerr - CanOpen Custom Error. This function is called at start Canopen nodes configuration

Parameters Nodes → Nodes number in configuration
Return No

Variables used

Internal VAR	Bit VAR	Define	Static VAR	VSD VAR	Fixed VAR
			No	EXP <input type="checkbox"/>	
Variable	Type	Shared	Export in Class		
Vect(3)	LONG	No			
RapX	FLOAT	No			
RapY	FLOAT	No			
RapZ	FLOAT	No			
ActualX	LONG	No			
ActualY	LONG	No			
ActualZ	LONG	No			
Node_Error(3)	CHAR	No			

Code in Main Page Functions

```

Page Init | Master Event | Master Cycle | Page Functions
1 | *****
2 | ' Return 1 if axes move
3 | ' 0 Axes stop
4 | *****
5 | function Wait_Move() as char
6 |     Wait_Move=interp.move()
7 | endfunction
8 | *****

```

```

' *****
' Return 1 if axes move
' 0 Axes stop
' *****
function Wait_Move() as char
    Wait_Move=interp.move()
endfunction

```

```

' *****
' Move Axes
' Vel= interp vel Axes in mm/min
' Flg if 1 move without buffer
' 0 move in buffer mode
' Px,Py,Pz Axes value in 0.001 mm
'Return 1 if movement is inserted in the buffer
' 0 The movement is not inserted in the buffer
' in this case, is necessary reload the movement
' *****
function Move_Axes(Vel as long, Flg as char, Px as long, Py as long,Pz as
long) as char
    Vel=Vel*TAU/60 ' Transform in mm/min
    Vect(0)=Px
    Vect(1)=Py
    Vect(2)=Pz
    Move_Axes=interp.moveto(Vel, Flg, Vect())
endfunction

```

```

' *****
' Set ACC
' Value Acc value in count
' *****
function Acc_Axes(Value as long) as void
    interp.acc=Value

```

VTB Software Resources NG QUARK

```
endfunction
'*****
' Stop Axes
'*****
function Stop_Axes() as void
    interp.stop()
endfunction
'*****
' Axis X enable
'*****
function Enable_X() as void
AxisX.mod0=0 ' remove interpolation mode
AxisX.start=0 ' stop PDO Qx
'Preset Axis X 0, not change y,z
Vect(0)=0
Vect(1)=interp.pc(1)
Vect(2)=interp.pc(2)
interp.preset(Vect())
AxisX.home=0 'preset driver
'enable axis
AxisX.enable=1
AxisX.start=1 ' start PDO Qx
AxisX.mod0=2 ' set interpolation mode
endfunction
'*****
' Axis X Disable
'*****
function Disable_X() as void
AxisX.enable=0
endfunction

'*****
' Axis Y enable
'*****
function Enable_Y() as void
AxisY.mod0=0 ' remove interpolation mode
AxisY.start=0 ' stop PDO Qx
'Preset Axis Y 0, not change x,z
Vect(0)=interp.pc(0)
Vect(1)=0
Vect(2)=interp.pc(2)
interp.preset(Vect())
AxisY.home=0 'preset driver
'enable axis
AxisY.enable=1
AxisY.start=1 ' start PDO Qx
AxisY.mod0=2 ' set interpolation mode
endfunction

'*****
' Axis Y Disable
'*****
function Disable_Y() as void
AxisY.enable=0
endfunction

'*****
' Axis Z enable
'*****
function Enable_Z() as void
```

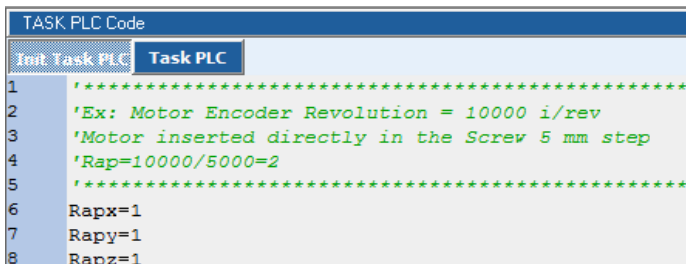
VTB Software Resources NG QUARK

```
AxisZ.mode=0 ' remove interpolation mode
AxisZ.start=0 ' stop PDO Qx
'Preset Axis Z 0, not change x,y
Vect(0)=interp.pc(0)
Vect(1)=interp.pc(1)
Vect(2)=0
interp.preset(Vect())
AxisZ.home=0 'preset driver
'enable axis
AxisZ.enable=1
AxisZ.start=1 ' start PDO Qx
AxisZ.mode=2 ' set interpolation mode
endfunction

'*****
' Axis Z Disable
'*****
function Disable_Z() as void
AxisZ.enable=0
endfunction

'*****
' Error check
' CanOpen node
'*****
function cancfgerr(node as int,err as uchar) as void
Node_Error(node)=err ' copy the code error
endfunction
'*****
' Close init CanOpen
'*****
function close_cancfgerr() as void
endfunction
'*****
' Custom error init
' CanOpen node
'*****
function open_cancfgerr(nodes as int) as void
' Reset nodes status error
Node_Error(0)=0
Node_Error(1)=0
Node_Error(2)=0
endfunction
```

Code in Init Task PLC

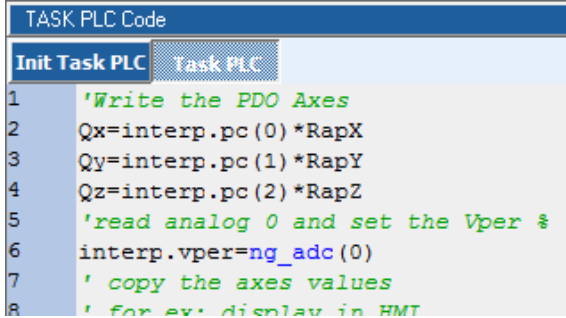


```
TASK PLC Code
Init Task PLC Task PLC
1 '*****
2 'Ex: Motor Encoder Revolution = 10000 i/rev
3 'Motor inserted directly in the Screw 5 mm step
4 'Rap=10000/5000=2
5 '*****
6 Rapx=1
7 Rapy=1
8 Rapz=1
```

```
'*****
'Ex: Motor Encoder Revolution = 10000 i/rev
'Motor inserted directly in the Screw 5 mm step
'Rap=10000/5000=2
```

```
'*****'  
Rapx=1  
Rapy=1  
Rapz=1
```

Code in Task PLC



```
TASK PLC Code  
Init Task PLC Task PLC  
1 'Write the PDO Axes  
2 Qx=interp.pc(0)*RapX  
3 Qy=interp.pc(1)*RapY  
4 Qz=interp.pc(2)*RapZ  
5 'read analog 0 and set the Vper %  
6 interp.vper=ng_adc(0)  
7 ' copy the axes values  
8 ' for ex: display in HMI
```

```
'Write the PDO Axes  
Qx=interp.pc(0)*RapX  
Qy=interp.pc(1)*RapY  
Qz=interp.pc(2)*RapZ  
'read analog 0 and set the Vper %  
interp.vper=ng_adc(0)  
' copy the axes values  
' for ex: display in HMI  
' value in 0.001 mm  
ActualX=interp.pc(0)  
ActualY=interp.pc(1)  
ActualZ=interp.pc(2)
```

Example Download

5.9 Example CanOpen Axes position mode

In the following example, are management, a CanOpen Axis by VTB OBJECT
See doc Vtb Object Guide for more informations.

WARNING:

All speed are managed in mm/min if setted the following parameters:

MSOF e DSOF

All axes target positions are managed in micron (0.001 mm) if setted the following parameters:

MSOF e DSOF

Objects used:



Motor Control Plus → **CobjPos** → **Posizionatore**

Project Explorer	
Project Objects Functions Properties	
Pos1	
Property Events	
Property	Value
Vper	1024
Div. Vper	1024
AccQstop	10
Acc	5
RzeroMode	1
RzeroOffset	0
RzeroPreset	0
RzeroVel	10
RzeroVelf	5
RzeroAcc	10
Msof	10000
Dsof	5000
LimitN	-99999999
LimitP	99999999
Gioco	0
Vgioco	1
MsofV	1
DsofV	1
RZERO ENABLE	True
AXIS TYPE	1
VTB AXIS OBJECT	CanPos1
PDO NAME	qx
STEP CHANNEL	0
STEP NODE	1

Motor Control → **CstdCanOpen** → **Ds402**

Project Explorer	
Project Objects Functions Properties	
CanPos1	
Property Events	
Property	Value
Name	CanPos1
Left	75
Top	30
Node	1
Mode	0
Speed	0
Position	0
Abs	True
State	False
home_delay	0

VTB Software Resources NG QUARK

Are managed the following functions:

Wait_Move – Axis state movement

Parameters No
Return 1 Axis in movement
0 Axes stop

Move_Axis – Move the Axis

Parameters Vel → Feed Axes in mm/min
Flg → Set to 1 for disable the movements buffer
(Stop axes at end trajectory)
Set to 0 for enable the movements buffer
Px, → Axes target values in 0.001 mm
Return 0 Movement inserted in the buffer – buffer empty
1 Buffer full (you must repeat Move_Axes up to when buffer empty)

Acc_Axis – Set Acceleration

Parameters Value → Value in count per TAU
Return No

Stop_Axis – Stop Axes

Parameters No
Return No

Enable – Enable the Axis control and preset at value 0

Parameters No
Return No

Disable – Disable the Axes control

Parameters No
Return No

StartHome – Start homing - Vel in pos1.rzerovel and pos1.rzerovelf

Parameters No
Return No

CheckHome – Check homing state

Parameters No
Return 1 homing finished

StopHome – Stop homing

Parameters No
Return No

cancfgerr – CanOpen Custom Error. This function is called at Canopen Node init (node setted in configuration by Canopen configurator) when the node, reply error

Parameters Node → Node number in error
Err → Error code
Return No

Close_cancfgerr - CanOpen Custom Error. This function is called at end Canopen nodes configuration

Parameters No
Return No

Open_cancfgerr - CanOpen Custom Error. This function is called at start Canopen nodes configuration

Parameters Nodes → Nodes number in configuration
Return No

Variables used

Internal VAR	Bit VAR	Define	Static VAR	VSD VAR	Fixed VAR
			No	EXP	
Variable	Type	Shared	Export in Class		
DigitalInputs	UINT	No			
Node_1_Error	CHAR	No			

Code in Main Page Functions

```

Page Init | Master Event | Master Cycle | Page Functions
1 | | | |
2 | | | | ' Enable Axis
3 | | | | '*****
4 | | | | function Enable() as void
5 | | | |     pos1.Enable()
6 | | | | endfunction
'*****
' Enable Axis
'*****
function Enable() as void
    pos1.Enable()
endfunction
'*****
' Disable Axis
'*****
function Disable() as void
    pos1.Disable()
endfunction
'*****
' Preset Axis
'*****
function Preset(Val as long) as void
    pos1.Preset(Val)
endfunction
'*****
' Return 1 if axis move
'     0 Axis stop
'*****
function Wait_Move() as char
    Wait_Move=pos1.move()
endfunction
'*****
' Axis Stop Move
'*****
function Stop() as void
    pos1.Stop()
endfunction
'*****
' Start Homing
' Homing input see in task plc
'*****
function StartHome() as void
    pos1.StartHome()
endfunction
'*****
' Check if homing finished
' Return 1 if finished
'*****
function CheckHome() as char
    CheckHome=pos1.status_home

```

VTB Software Resources NG QUARK

```
endfunction
'*****
' Stop home function
'*****
function StopHome () as void
    pos1.StopHome ()
endfunction
'*****
' Move Axis
' Vel= vel Axis in mm/min
' Flg if 1 move without buffer
'     0 move in buffer mode
' Px Axis value in 0.001 mm
'Return 1 if movement is inserted in the buffer
'     0 The movement is not inserted in the buffer
'     in this case, is necessary reload the movement
'*****
function Move_Axis (Vel as long, Flg as char, Px as long) as char
    Vel=Vel*TAU/60 ' Transform in mm/min
    Move_Axis=pos1.moveto (Vel, Flg, Px)
endfunction
'*****
' Set ACC
' Value Acc value in count
'*****
function Acc_Axis (Value as long) as void
    pos1.acc=Value
endfunction
'*****
' Error check
' CanOpen node
'*****
function cancfgerr (node as int, err as uchar) as void
Node_1_Error=err ' copy the code error
endfunction
'*****
' Close init CanOpen
'*****
function close_cancfgerr () as void
endfunction
'*****
' Custom error init
' CanOpen node
'*****
function open_cancfgerr (nodes as int) as void
' Reset node 1 status error
Node_1_Error=0
endfunction
```

Code in Init Task PLC

TASK PLC Code	
Init Task PLC	Task PLC
1	pos1.msosf=10000 ' motor 10000 i/rev
2	pos1.ext_fcZ=Fc_Home ' home input

```
pos1.msosf=10000 ' motor 10000 i/rev
pos1.dsosf=5000 ' 5 mm per revolution motor
```

Code in ask PLC

```
TASK PLC Code
Init Task PLC Task PLC
1 DigitalInputs=ng_di(0) ' read digital inputs
2 pos1.ext_fcz=Fc_Home ' home input
```

```
DigitalInputs=ng_di(0) ' read digital inputs
pos1.ext_fcz=Fc_Home ' home input
```

[Example Download](#)

Code in ask PLC

```
TASK PLC Code
Init Task PLC Task PLC
1 DigitalInputs=ng_di(0) ' read digital inputs
2 pos1.ext_fcz=Fc_Home ' home input
```

```
DigitalInputs=ng_di(0) ' read digital inputs
pos1.ext_fcz=Fc_Home ' home input
```

[Example Download](#)

6 Digital I/O

The NGQ boards, allows to use 11 digital inputs and 8 digital outputs, management by VTB functions.

6.1 NG_DI – Read Digital Inputs

Read the Digital Inputs state.

The Digital Inputs are bit mapped – from 0 to 10

Input	11	10	9	8	7	6	5	4	3	2	1
Bit	10	9	8	7	6	5	4	3	2	1	0

Syntax

```
NG_DI(CardNumber as Char) as uint
```

Parameters

CardNumber 0

Return Value

Uint Value - 11 inputs bit mapped

bit = 1 → **Input ON**

bit = 0 → **Input OFF**

6.2 NG_DO – Writ Digital Outputs

Writes the digital outputs state

The Digital Outputs are bit mapped – from 0 to 7

Output	8	7	6	5	4	3	2	1
Bit	7	6	5	4	3	2	1	0

Syntax

```
NG_DO(CardNumber as Char, StatoOutputs as Uint) as void
```

Parameters

CardNumber 0

StatoOutputs Output state

bit = 1 → **Output ON**

bit = 0 → **Output OFF**

6.3 Example Digital I/O

In the next example, are managed the Digital I/O in the following mode:

UPDATING I/O IN TASK PLC

Management I/O in bit mode. The first 3 inputs are copied in the first 3 outputs

Variables used

Internal VAR	Bit VAR	Define	Static VAR	VSD VAR	Fixed VAR
<div style="border: 1px solid gray; padding: 2px;"> ▼ No ▼ EXP <input type="checkbox"/> </div>					
Variable	Type	Shared	Export in Class		
DigOutputs	UINT	No			
DigInputs	UINT	No			

BIT Used

Internal VAR	Bit VAR	Define	Static VAR	VSD VAR
<div style="border: 1px solid gray; padding: 2px;"> ▼ No ▼ </div>				
Name	Main Variable	NBit	Shared	
INP0	DigInputs	0	No	
INP1	DigInputs	1	No	
INP2	DigInputs	2	No	
INP3	DigInputs	3	No	
INP4	DigInputs	4	No	
INP5	DigInputs	5	No	
INP6	DigInputs	6	No	
INP7	DigInputs	7	No	
INP8	DigInputs	8	No	
INP9	DigInputs	9	No	
INP10	DigInputs	10	No	
OUT0	DigOutputs	0	No	
OUT1	DigOutputs	1	No	
OUT2	DigOutputs	2	No	
OUT3	DigOutputs	3	No	
OUT4	DigOutputs	4	No	
OUT5	DigOutputs	5	No	
OUT6	DigOutputs	6	No	
OUT7	DigOutputs	7	No	

Code in Task PLC

```
TASK PLC Code
Init Task PLC  Task PLC
1  OUT0=INP0 ' copy input 0 on outpus 0
2  OUT1=INP1 ' copy input 1 on outpus 1
3  OUT2=INP2 ' copy input 2 on outpus 2

OUT0=INP0 ' copy input 0 on outpus 0
OUT1=INP1 ' copy input 1 on outpus 1
OUT2=INP2 ' copy input 2 on outpus 2
DigInputs=ng_di(0) ' udpate digital inputs
ng_do(0,DigOutputs) ' update digital outputs
```

[Example Download](#)

7 Analog Outputs

The NGQ, manage 2 Analog Outputs +/- 10 V 12 bit

7.1 NG_DAC – Write Analog Outputs


This function allows to update the analog outputs of each channel equipped in the NGQ. These expansions have a digital to analog converter at 12 bit, with a range of +/-10V. Therefore a value of +2047 corresponds to 10V in output, a value of -2047 corresponds to -10V. The selection of the channel is made by an index from 0 to 7, each expansion manages two channels:

Syntax

NG_DAC(Channel as Char, Val as Long) as void

Parameters

Channel Number of channel (from 0 to 1)
val Analog output value (from -2047 to +2048)

 **WARNING**
THE ANALOG OUTPUTS ARE IN THE LITTLE EXPANSION BOARD. TO USE THESE ANALOG OUTPUTS MUST BE ENABLED THE PROPERTY "ENCODER ENABLE" FROM NGQ INIT OBJECT

NGQ INIT

→

P-P Interp. Mask	7
STEP Level	0
ENCODER Enable	True

7.2 NG_DAC_CAL - CALIBRATION OF THE ANALOG OUTPUT OFFSET

This function allows to calibrate the OFFSET of the analog outputs. Usually it can be occur that the analog output has a little value of voltage (OFFSET) in the order of mV also if zero has been set. With *ng_dac_cal* we can null this voltage setting a value opposite to the offset one. Remind that for each unit the output value will be about 4mV.

Syntax

NG_DAC_CAL(Channel as Char, Offset as Long) as void

Parameters

Channel Numero Canale (from 0 to 1)
Offset OFFSET value (from -2047 to +2048)

 **WARNING**
THE OFFSET VALUE ISN'T SAVED AND IT MUST BE SET AT EACH TURN-ON. SAVE THIS VALUE IN NGQ FRAM MEMORY

7.3 Example Analog Outputs

In the next example, are managed the analog outputs

Variables used

Internal VAR	Bit VAR	Define	Static VAR	VS
				No
Variable	Type	Shared		
AnalogInput	UINT	No		

Code in Task PLC

```

TASK PLC Code
Init Task PLC Task PLC
1 AnalogInput=ng_adc(0) ' read analog input 1 0 to 1023 0-10V
2 ' 0 to 512 output -10V to 0 v
3 ' 512 to 1023 output 0 V to 10V
4 AnalogInput=AnalogInput<<2
5 Ng_dac(0,AnalogInput) ' copy in the analog output 0
    
```

```

AnalogInput=ng_adc(0) ' read analog input 1 0 to 1024 0-10V
' -2047 0 output -10V to 0 v
' 0 2048 output 0 V to 10V
AnalogInput=AnalogInput-2048
Ng_dac(0,AnalogInput) ' copy in the analog output 0
    
```

Example Download

8 PULSE/DIR channels on NGQ

The NGQ can use 4 channels PULSE/DIR on CPU

NGQ *clock position mode* *125 KHz total*
NGQ *clock interpolation mode* *30 KHz total*

8.1 PP_STEP – Generating STEP/DIR signals

This function, is the primitive that allows the generation STEP and DIR signal on the specified channel. Generally it is used, by objects that allows to “*Ramp and Position*” generator.

Syntax

PP_STEP(Channel **as Char**, Value **as Long**) **as void**

Parameters

Channel Number of the STEP/DIR channel
 from 0 to 3
Value Absolute value of the position of the step/dir axis



8.2 PP_PRESET – PRESET OF STEP/DIR POSITION

This function updates the current position of a step/dir channel.

Syntax

PP_PRESET(Channel **as Char**, Value **as Long**) **as void**

Parameters

Channel Numero del canale STEP/DIR
From 0 to 3

Value Valore della posizione che assumerà il l'asse step/dir



8.3 PP_GETPOS – READING OF ACTUAL POSITION

This function reads the actual position of a step/dir channel. The value will correspond to the DOUBLE of the real position.

Syntax

PP_GETPOS(Channel **as Char**) **as long**

Parameters

Channel Number of the STEP/DIR channel

Return Value

Long Actual position x 2

8.4 Example STEP/DIR Axes in Interpolation Mode

In the following example, are managed, 3 STEP/DIR Axes In linear interpolation.

WARNING:

ATTENTION:

All speed are managed in mm/min if setted the following parameters

RapX,RapY,RapZ

All axes target positions are managed in micron (0.001 mm) if setted the following parameters

RapX,RapY,RapZ

Objects used:



Motor Control → **CobjInterpola** → **Interpolatore**

Project Explorer				
Project	Objects	Functions	Properties	Tables
Interp				
Property Events				
Property	Value			
Nome	Interp			
Left	15			
Top	10			
N.assi	3			
N.tratti	16			
Vper	1024			
Div. Vper	1024			
Abilita arcto	1			

Are managed the following functions:

Wait_Move – Axes state movement

Parameters No
Return 1 Axes in movement
 0 Axes stop

Move_Axes – Move the Axes in linear interpolation

Parameters Vel → Feed Axes in mm/min
 Flg → Set to 1 for disable the movements buffer
 (Stop axes at end trajectory)
 Set to 0 for enable the movements buffer
 (Stop Axes only if edge > SGLP)
 Px,Py,Pz → Axes target values in 0.001 mm
Return 0 Movement inserted in the buffer – buffer empty
 1 Buffer full (you must repeat Move_Axes up to when buffer empty)

Acc_Axes – Set interpolation Acceleration

Parameters Value → Value in count per TAU
Return No

Stop_Axes – Stop Axes

Parameters No
Return No

Enable_Axis_X_Y_Z – Enable the Axes control and preset at value 0

Parameters No
Return No

Variables used

Internal VAR	Bit VAR	Define	Static VAR	VSD VAR	Fixed V
			No	EXP	
Variable	Type	Shared	Export in Class		
Vect(3)	LONG	No			
RapX	FLOAT	No			
RapY	FLOAT	No			
RapZ	FLOAT	No			
ActualX	LONG	No			
ActualY	LONG	No			
ActualZ	LONG	No			
DisableStep	CHAR	No			

Code in Main Page Functions

```

Page Init  Master Event  Master Cycle  Page Functions
1  '*****
2  ' Return 1 if axes move
3  '      0 Axes stop
4  '*****
5  function Wait_Move() as char
6      Wait_Move=interp.move()
7  endfunction
8  '*****

```

```

'*****
' Return 1 if axes move
'      0 Axes stop
'*****
function Wait_Move() as char
    Wait_Move=interp.move()
endfunction

```

```

'*****
' Move Axes
' Vel= interp vel Axes in mm/min
' Flg if 1 move without buffer
'      0 move in buffer mode
' Px,Py,Pz Axes value in 0.001 mm
'Return 1 if movement is inserted in the buffer
'      0 The movement is not inserted in the buffer
'      in this case, is necessary reload the movement
'*****
function Move_Axes(Vel as long, Flg as char, Px as long, Py as long,Pz as
long) as char
    Vel=Vel*TAU/60 ' Transform in mm/min
    Vect(0)=Px
    Vect(1)=Py
    Vect(2)=Pz
    Move_Axes=interp.moveto(Vel, Flg, Vect())
endfunction

```

```

'*****
' Set ACC
' Value Acc value in count
'*****
function Acc_Axes(Value as long) as void
    interp.acc=Value
endfunction

```

```
'*****  
' Stop Axes  
'*****  
function Stop_Axes() as void  
    interp.stop()  
endfunction  
'*****  
' Axis X enable  
'*****  
function Enable_X() as void  
    DisableStep=1  
'Preset Axis X 0, not change y,z  
    Vect(0)=0  
    Vect(1)=interp.pc(1)  
    Vect(2)=interp.pc(2)  
    interp.preset(Vect())  
'enable axis  
    DisableStep=0  
endfunction  
'*****  
' Axis Y enable  
'*****  
function Enable_Y() as void  
    DisableStep=1  
'Preset Axis Y 0, not change X,z  
    Vect(0)=interp.pc(0)  
    Vect(1)=0  
    Vect(2)=interp.pc(2)  
    interp.preset(Vect())  
'enable axis  
    DisableStep=0  
endfunction  
'*****  
' Axis Z enable  
'*****  
function Enable_Z() as void  
    DisableStep=1  
'Preset Axis Z 0, not change X,Y  
    Vect(0)=interp.pc(0)  
    Vect(1)=interp.pc(1)  
    Vect(2)=0  
    interp.preset(Vect())  
    PidZ.posr=0  
'enable axis  
    DisableStep=0  
endfunction
```

Code in Init Task PLC

```
TASK PLC Code
Init Task PLC Task PLC
1  '*****
2  'Ex: Motor Encoder Revolution = 10000 i/rev
3  'Motor inserted directly in the Screw 5 mm step
4  'Rap=10000/5000=2
5  '*****
6  Rapx=1
7  Rapy=1
8  Rapz=1
```

```
'*****
'Ex: Motor Encoder Revolution = 10000 i/rev
'Motor inserted directly in the Screw 5 mm step
'Rap=10000/5000=2
'*****
Rapx=1
Rapy=1
Rapz=1
```

Code in Task PLC

```
TASK PLC Code
Init Task PLC Task PLC
1  if DisableStep=0 ' disable output step
2     pp_step(0, interp.pc(0)*RapX) 'Update the X Axis
3     pp_step(1, interp.pc(1)*RapY) 'Update the Y Axis
4     pp_step(2, interp.pc(2)*RapZ) 'Update the Z Axis
5  endif
```

```
if DisableStep=0 ' disable output step
    pp_step(0, interp.pc(0)*RapX) 'Update the X Axis
    pp_step(1, interp.pc(1)*RapY) 'Update the Y Axis
    pp_step(2, interp.pc(2)*RapZ) 'Update the Z Axis
endif
'read analog 0 and set the Vper %
interp.vper=ng_adc(0)
' copy the axes values
' for ex: display in HMI
' value in 0.001 mm
ActualX=interp.pc(0) ' read actual position X
ActualY=interp.pc(1) ' read actual position Y
ActualZ=interp.pc(2) ' read actual position Z
```

Example Download

8.5 Example STEP/DIR Axis in Position Mode

In the following example, are management, a CanOpen Axis by VTB OBJECT
See doc Vtb Object Guide for more informations.

WARNING:

All speed are managed in mm/min if setted the following parameters:

MSOF e DSOF

All axes target positions are managed in micron (0.001 mm) if setted the following parameters:

MSOF e DSOF

Objects used:



Motor Control Plus → **CobjPos** → **Posizionatore**

Property	Value
Nome	Pos1
Left	25
Top	30
N.TRATTI	8
Vper	1024
Div. Vper	1024
AccQstop	10
Acc	5
RzeroMode	1
RzeroOffset	0
RzeroPreset	0
RzeroVel	10
RzeroVelf	5
RzeroAcc	10
Msof	10000
Dsof	5000
LimitN	-99999999
LimitP	99999999
Gioco	0
Vgioco	1
MsofV	1
DsofV	1
RZERO ENABLE	True
AXIS TYPE	2
VTB AXIS OBJECT	0
PDO NAME	0
STEP CHANNEL	0
STEP NODE	0

VTB Software Resources NG QUARK

Are managed the following functions:

Wait_Move – Axis state movement

Parameters No
Return 1 Axis in movement
0 Axes stop

Move_Axis – Move the Axis

Parameters Vel → Feed Axes in mm/min
Flg → Set to 1 for disable the movements buffer
(Stop axes at end trajectory)
Set to 0 for enable the movements buffer
Px, → Axes target values in 0.001 mm
Return 0 Movement inserted in the buffer – buffer empty
1 Buffer full (you must repeat Move_Axes up to when buffer empty)

Acc_Axis – Set Acceleration

Parameters Value → Value in count per TAU
Return No

Stop_Axis – Stop Axes

Parameters No
Return No

Enable – Enable the Axis control and preset at value 0

Parameters No
Return No

Disable – Disable the Axes control

Parameters No
Return No

StartHome – Start homing - Vel in pos1.rzerovel and pos1.rzerovelf

Parameters No
Return No

CheckHome – Check homing state

Parameters No
Return 1 homing finished

StopHome – Stop homing

Parameters No
Return No

Variables used

Internal VAR	Bit VAR	Define	Static VAR	VSD VAR	Fixed VA
				No	EXP <input type="checkbox"/>
Variable	Type	Shared	Export in Class		
DigitalInputs	UINT	No			

Code in Main Page Functions

Page Init	Master Event	Master Cycle	Page Functions
1			'*****
2			' Enable Axis
3			'*****
4			function Enable() as void
5			pos1.Enable()
6			endfunction

```
'*****
' Enable Axis
'*****
function Enable() as void
    pos1.Enable()
endfunction
'*****
' Disable Axis
'*****
function Disable() as void
    pos1.Disable()
endfunction
'*****
' Preset Axis
'*****
function Preset(Val as long) as void
    pos1.Preset(Val)
endfunction
'*****
' Return 1 if axis move
'    0 Axis stop
'*****
function Wait_Move() as char
    Wait_Move=pos1.move()
endfunction
'*****
' Axis Stop Move
'*****
function Stop() as void
    pos1.Stop()
endfunction
'*****
' Start Homing
' Homing input see in task plc
'*****
function StartHome() as void
    pos1.StartHome()
endfunction
'*****
' Check if homing finished
' Return 1 if finished
```

VTB Software Resources NG QUARK

```
'*****  
function CheckHome () as char  
    CheckHome=pos1.status_home  
endfunction  
'*****  
' Stop home function  
'*****  
function StopHome () as void  
    pos1.StopHome ()  
endfunction  
'*****  
' Move Axis  
' Vel= vel Axis in mm/min  
' Flg if 1 move without buffer  
'     0 move in buffer mode  
' Px Axis value in 0.001 mm  
'Return 1 if movement is inserted in the buffer  
'     0 The movement is not inserted in the buffer  
'     in this case, is necessary reload the movement  
'*****  
function Move_Axis (Vel as long, Flg as char, Px as long) as char  
    Vel=Vel*TAU/60 ' Transform in mm/min  
    Move_Axis=pos1.moveto (Vel, Flg, Px)  
endfunction  
'*****  
' Set ACC  
' Value Acc value in count  
'*****  
function Acc_Axis (Value as long) as void  
    pos1.acc=Value  
endfunction
```

Code in Init Task PLC

```
TASK PLC Code  
Init Task PLC Task PLC  
1 pos1.msosf=10000 ' motor 10000 i/rev  
2 pos1.ext_fcZ=Fc_Home ' home input
```

```
pos1.msosf=10000 ' motor 10000 i/rev  
pos1.dsosf=5000 ' 5 mm per revolution motor
```

Code in Task PLC

```
TASK PLC Code  
Init Task PLC Task PLC  
1 DigitalInputs=ng_di(0) ' read digital inputs  
2 pos1.ext_fcZ=Fc_Home ' home input
```

```
DigitalInputs=ng_di(0) ' read digital inputs  
pos1.ext_fcZ=Fc_Home ' home input
```

Example Download

9 Permanent Memory

The NGQ, uses 2 tipology permanent memory:

Internal type FLASH 1024 Bytes
External type FRAM 16 Kb (optional)

The difference is the following:

The internal FLASH memory, must be writing with only block to 1024 Bytes
The external FRAM memory, can be writing byte per byte

The memory selection, is by software, using the addr value:

Addr from 0 to 1023 **Internal FLASH memory**
Addr from 1024 to 17407 **External FRAM memory**

9.1 Permanent memory

9.1.1 IMS_READ – Read memory

Reads from the memory at address ADDR a number of byte as in NBYTE and writes them in the array pointed by Punt..

Syntax

IMS_READ(Punt **as *Char**, Addr **as Long**, Nbyte **as Long**) **as Char**

Parameters

Punt Pointer to data buffer where read data will be saved
Addr Start address in the reserved area of the device
Nbyte Number of bytes to be read

Return Value

Char 0 No error
 <>0 Writing error

9.1.2 IMS_WRITE – Write memory

Writes in the memory at the address contained in ADDR, the data pointed by Punt for a total of NBYTE of data.

Syntax

IMS_WRITE(Punt **as *Char**, Addr **as Long**, Nbyte **as Long**) **as Char**

Parameters

Punt Pointer to data buffer to be written
Addr Start address in the reserved area of the device
Nbyte Number of bytes to be written

Return value:

Char 0 No error
 <>0 Writing error



WARNING

THE EXTERNAL FRAM MEMORY, CAN BE WRITING BYTE per BYTE
THE INTERNAL FLASH MEMORY, MUST BE WRITING WITH BLOCK TO 1024 BYTES

9.2 Example save/load in external FRAM

In the following example, are saved and loaded by FRAM the values in a Long Vector. This example can be used for a machines parameters management .

Is used a Checksum (parameters values sum) and saved in the LAST position of array. The Checksum is used to ensure, the parameters integrity

Are managed the following functions:

LoadPar – Load from FRAM the values

Parameters No
Return 0 OK
 1 Error FRAM
 2 Error checksum

SavePar – Save in FRAM the values

Parameters No
Return 0 OK
 1 Error FRAM

Variables used

Internal VAR	Bit VAR	Define	Static VAR	VSD VAR	Fixed V
			No	EXP	<input type="checkbox"/>
Variable	Type		Shared	Export in Class	
val_par(PAR_NUMBER)	LONG		No		

DEFINE used

Internal VAR	Bit VAR	Define	Static VAR
Variable	Type		
PAR_NUMBER	100		

Code in Main Page Functions

```

Page Init | Master Event | Master Cycle | Page Functions
1 .....
2 'Load parameters from FLASH in RAM
3 'Calculates the checksum
4 'return >0 ERROR
5 .....
6 function LoadPar() as char
7 dim n as long

```

```

'*****
'Load parameters from FRAM in RAM
'Calculates the checksum
'return >0 ERROR
'*****
function LoadPar() as char
dim n as long
dim ckl as long
dim ck as long
dim Ret as char
'PAR_NUMBER is number of parameters
'all parameters are in long

```

VTB Software Resources NG QUARK

Ret=**ims_read**(val_par(),1024,PAR_NUMBER*4) ' reads parameters from FRAM
and 'puts in val_par vector

```
if Ret<>0
    'LOAD ERROR !!!!
    LoadPar=1 'return ERROR 1
    return
endif
ck=val_par(PAR_NUMBER) 'gets the check sum in last position
ckl=0
for n=0 to n<(PAR_NUMBER-1) 'calculates the checksum
    ckl=ckl+val_par(n)
next n
if ckl=0          'if all parameters are ZERO - chekcsun error
    ckl=ck+1
endif
if ckl<>ck
    'Checksum ERROR
    LoadPar=2 'return ERROR 2
else
    LoadPar=0 'return OK
endif
endfunction

'*****
'Save the parameters in FRAM
'Return >0 ERROR
'*****
function SavePar() as char
dim ck as long
dim n as long
dim Ret as char
ck=0
for n=0 to n<(PAR_NUMBER)-1 'calculates the checksum
    ck=ck+val par(n)
next n
val_par(PAR_NUMBER-1)=ck 'put the checksum
Ret=ims_write(val_par(),1024,PAR_NUMBER*4) 'save the parameters
if Ret<>0
    'SAVE ERROR !!!!
    SavePar=1 'return ERROR 1
else
    SavePar=0 'return OK
endif
endfunction
```

Example Download



Index

1	Preface	2
2	RS232/RS485 Port.....	3
2.1	SER_SETBAUD.....	3
2.2	SER_MODE	3
2.3	SER_GETCHAR	3
2.4	SER_PUTCHAR	3
2.5	SER_PUTS	3
2.6	SER_PRINTL	4
2.7	SER_PRINTF	4
2.8	SER_PUTBLK.....	4
2.9	SER_PUTST.....	4
2.10	Example.....	5
3	Modbus RTU	7
3.1	Modbus RTU Slave Object	7
3.2	Example ModBus slave	7
3.3	Modbus RTU Master Object.....	9
3.4	Example ModBus Master	10
4	Analog Inputs Read	11
4.1	Inputs Read	11
4.2	Example Analog inputs read	11
5	CanOpen Management	12
5.1	PXCO_SDODL	12
5.2	PXCO_SDOUL	12
5.3	READ_SDOAC	13
5.4	PXCO_SEND	13
5.5	PXCO_NMT	13
5.6	READ_EMCY	14
5.7	Example CanOpen Functions	15
5.8	Example CanOpen Axes interpolation mode.....	18
5.9	Example CanOpen Axes position mode	24
6	Digital I/O.....	30
6.1	NG_DI – Read Digital Inputs	30
6.2	NG_DO – Writ Digital Outputs	30
6.3	Example Digital I/O.....	31
7	Analog Outputs.....	33

VTB Software Resources NG QUARK

7.1	NG_DAC – Write Analog Outputs	33
7.2	NG_DAC_CAL - CALIBRATION OF THE ANALOG OUTPUT OFFSET.....	33
7.3	Example Analog Outputs	34
8	PULSE/DIR channels on NGQ	35
8.1	PP_STEP – Generating STEP/DIR signals.....	35
8.2	PP_PRESET – PRESET OF STEP/DIR POSITION.....	36
8.3	PP_GETPOS – READING OF ACTUAL POSITION.....	36
8.4	Example STEP/DIR Axes in Interpolation Mode.....	37
8.5	Example STEP/DIR Axis in Position Mode	41
9	Permanent Memory	45
9.1	Permanent memory	45
9.1.1	IMS_READ – Read memory.....	45
9.1.2	IMS_WRITE – Write memory	45
9.2	Example save/load in external FRAM.....	46

