VTBII Visual Tool Basic

www.promax.it

User Guide

PROMAX Motion & Control

Rev. 1.00.0

1 INTRODUCTION

VTBII is an integrated development environment for OBJECT oriented programming on PROMAX platforms. This environment contains inside all tools needed to development of application in simple and intuitive way. The VTBII philosophy is based on latest technologies R.A.D. (RAPID APPLICATION DEVELOPMENT) which allow a fast development of application writing a reduced amount of source code. A large library of OBJECTS and TECNHOLOGIC FUNCTIONS allow to create applications for all sector area of industrial automation. VTBII integrates a high level language like enhanced BASIC MOTION. It's also possible to manage in clear and simple way FIELD BUS such as:

CAN OPEN

ETHERCAT

MODBUS

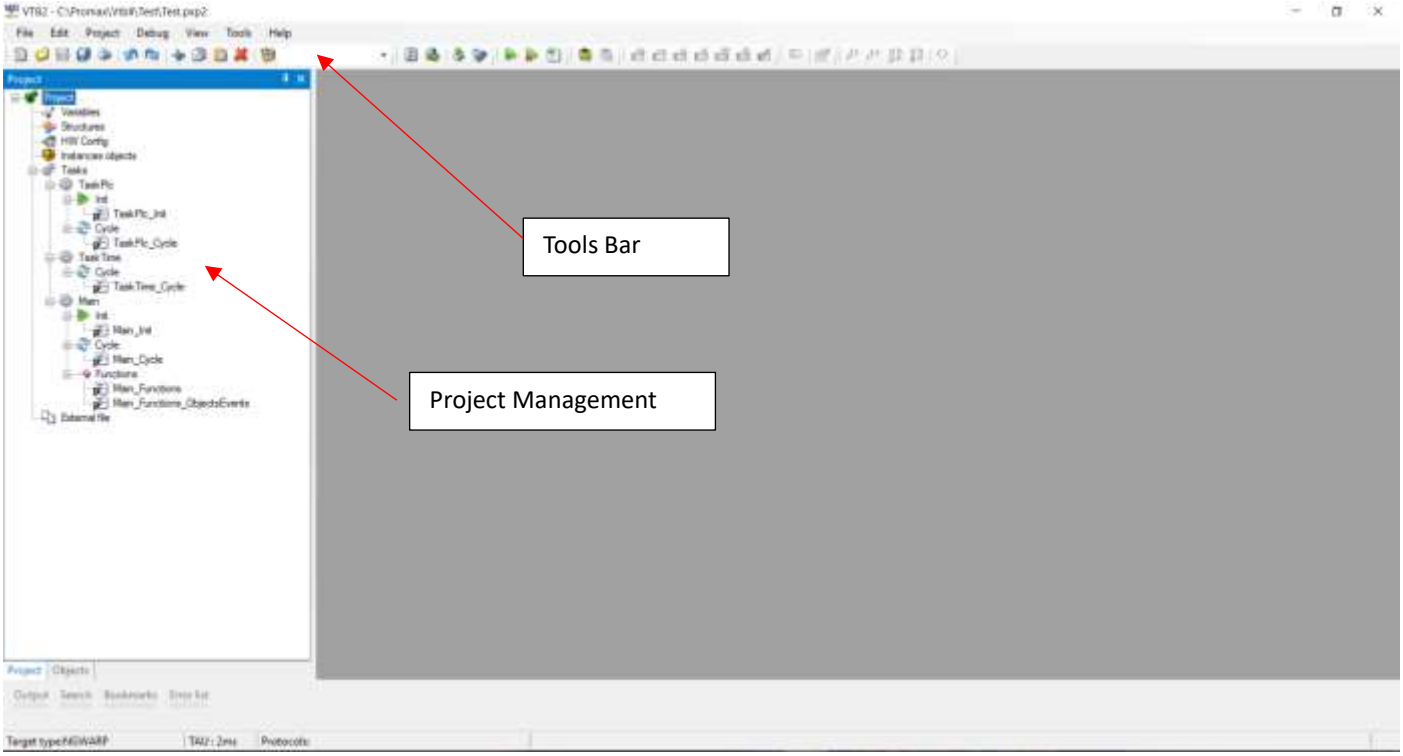
Powerful functions of AXIS MOVING allow to manage any type of machine using LINEAR, CIRCULAR, FAST LINEAR INTERPOLATION or ELECTRIC GEAR, CAM PROFILES, etc.

2 NOTES ON PROGRAMMING LANGUAGE

VTBII programming language is defined as **BASIC MOTION**.
Its syntax is very similar as enhanced BASIC with some terminologies derived from **C** language. Management of the functions is very similar as **VISUAL BASIC** also for **DATA STRUCTURES**.
Some **INSTRUCTIONS** are **VTB PROPRIETARY** but following the same philosophy.
VTBII is a language **CASE INSENSITIVE** that is it make no differences between **UPPER CASE** and **LOWER CASE** regarding instructions, functions, variables etc. **VTB** converts internally all characters in **UPPER CASE**. The only one exception is the management of **DEFINE** where characters are not converted in upper case but they remain so in all compilation passes.
Because VTB is a language addressed to MOTION, some features, considered of secondary importance, remained at PRIMITIVE level. For example, the STRING management is made like C language using function such as STRCPY, STRCAT, STRCMP etc.

3 DEVELOPMENT ENVIRONMENT

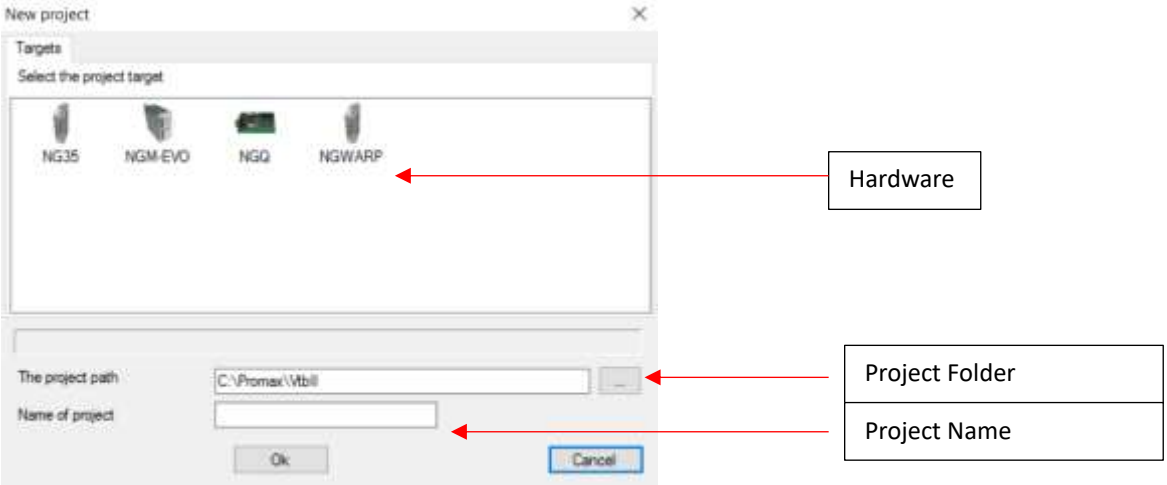
The development environment of VTBII has a common intuitive interface like all Windows applications. It isn't necessary to have a great experience of programming. In the environment is included an EDITOR with Intellisense optimized for VTBII programming.



3.1 Tools Bar

New Project - From menù ***File → New Project***

It creates a new application. The previous one will be closed requesting a confirm for saving.
Following will be open the Window Hardware Selector:



Open Project - From menù ***File → Open Project***
Opens an existing project

Document Save - From menù ***File → Document Save***
Saves the current document

Project Save - From menù ***File*** **→ Project Save**
Save the current Project

Undo - From menù ***Edit → Undo***
Undo last operation

Repat - From menù ***Edit*** **→ Repat**
Repat last operation Undo

Cut - From menù ***Edit*** **→ Cut**
Cut the selected element

Copy - From menù ***Edit → Copy***
Copy the selected element

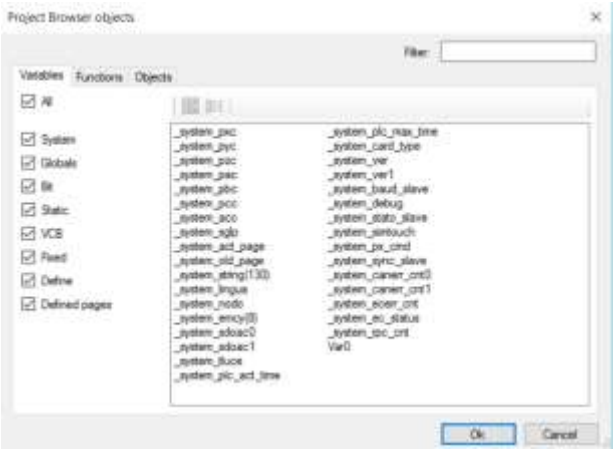
Paste - From menù ***Edit*** **→ Paste**
Paste from clipboard

Delete - From menù ***Edit*** **→ Delete**
Delete the selected element

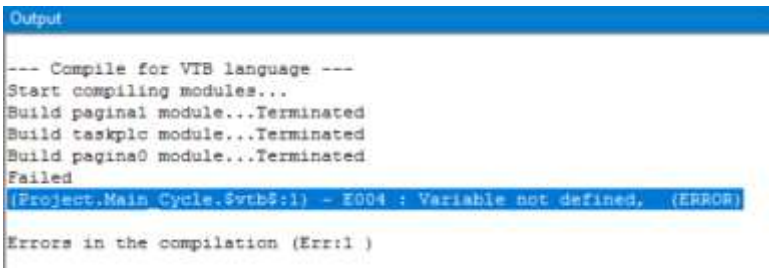
Find and Replace - From menù ***Edit*** **→ Find and Replace**
Find and Replace the Text

Project Settings - From menù ***Project*** **→ Project Settings**
Settings of Project (See [Application Configuration](#))

Project Browser - From menù ***Project*** → **Project Browser**
Allows to show the variables, Functions and Objects declared in the current project



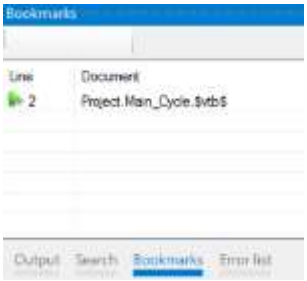
Build - From menù ***Project*** → **Build**
Build the current project
The results are showed in the output window.



Double click on error line for show the source code

Upload Application - From menù ***Project*** → **Upload Application**
Upload Application in the target by the select channel (Ethernet or RS232)

Add/Remove Bookmark
Add or Remove Bookmark from the current line
For return to bookmark, open "Bookmarks" window and select the desired Bookmark with doubleclick



Remove All Bookmarks
Remove All Bookmarks .

Go To Line
Moves the Cursor to line

Start Debug - From menù ***Debug*** → **Start**
Start the Debug session (see **Debug Application**)

Stop Debug - From menù ***Debug*** → **Stop**
Stop the Debug session

3.2 Project Manager

PROJECT MANAGER allows a rapid selection and navigation inseide the Application Project.
With PROJECT MANAGER all functions are easy to reach.

Opens the variables manager
(See **Type of Variables**)

Opens the structures manager
(See **Structure**)

Hardware Configuration
The Hardware Configuration is based the Board used
Therefore the parameters are different for each Hardware type
(See **Hardware Configuration**)

Show the Objects Instances
(See **Object Manager**)

Show all VTBII TasksI
(See **VTBII Tasks**)

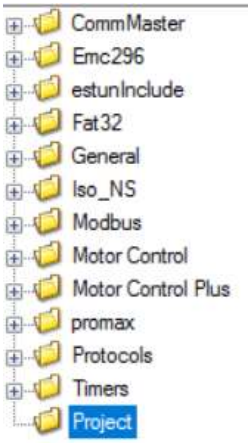
3.3 Object Manager

OBJECT MANAGER allows a rapid selection of Objects to insert in the project

3.3.1 Insert A Object

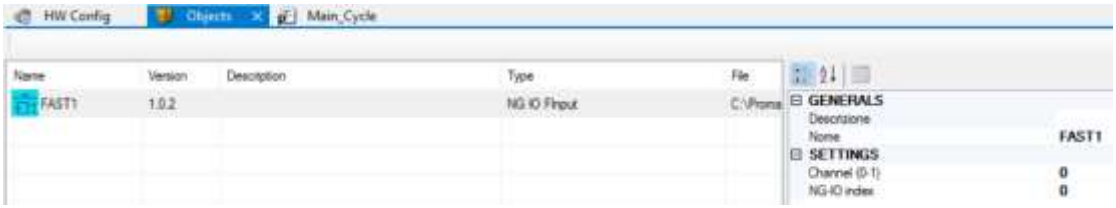
Select **TAB** Objects:

Following all Objects will be shown



Open the folder and add the object with double click or button 

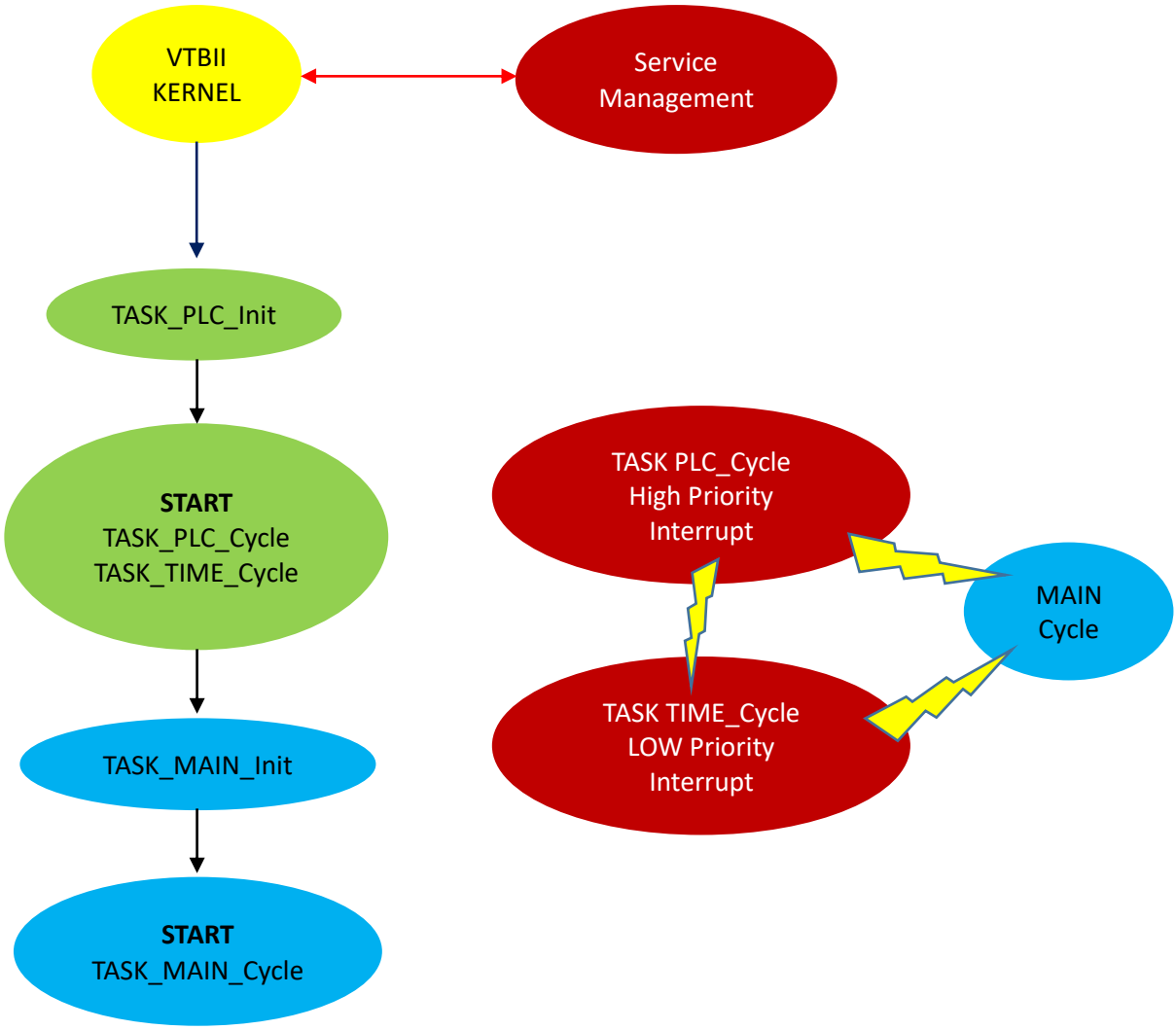
The object will be inserted in the Objects Window and the Properties will be shown



4 VTBII TASKS

VTBII uses Three differents TASKS.
Two tasks are ***Interrupt Time Type***, these can interrupt the other processes.
The third Task is managed in cooperative mode, that is executed when the Interrupt Tasks are not in execution.

Task Plc	Deterministic Interrupt Time
Task Time	Not Deterministic Interrupt Time
Task Main	Cooperative Task



4.1 Task Plc

The Task Plc has two section. **TASK PLC_Init** and **TASK PLC_Cycle**

4.1.1 TASK PLC_INIT

It is the **FIRST** task that the S.O. executes. Normally it contains all Initializations for Application (All Tasks).

4.1.2 TASK PLC_Cycle

This task is the highest priority one: it is deterministic and run at fixed time making it suitable to manage situation that need a fast and precise response time. This task can not be interrupted by no other tasks but it can instead interrupt any other. Normally it is used by AXIS CONTROL OBJECTS or fast PLC cycles, but it can contain every type of code sequence excluding some IFS functions like:

AXIS INTERPOLATION (xxx.MOVETO, xxx.LINE_TO)
MANAGE OF CANOPEN SDO.
STATIC CYCLES
(see the single functions for details)
The typical sample time is 2 milliseconds that is an enough time to manage a lot of application (for example 6 AXIS interpolation), however it can go down also under 1 millisecond when the charge of work is less stressful and for CPU with high computing power. In this task is also managed the CAN OPEN and ETHERCAT protocol in DETERMINISTIC mode. However it is advisable that its elapsed time doesn't exceed 90% of sample time, else we risk to slow or even to stop the other tasks.
IF THE CODE INSIDE TASK PLC BLOCKS IT ALL SYSTEM GO IN CRASH.
To verify the elapsed time of TASK PLC there are two field in DEBUG.NET application:
PLC TP and **PLC TM** never must exceed the sample time.

4.1.3 Note on Concurrent Programming

The use of CONCURRENT programming requires particular ATTENTION as in all MULTITASK systems. To avoid unexpected operation it's recommended do not call the same function from INTERRUPT TASKS and COOPERATIVE TASK in the same application. In other words the functions managed by MAIN TASK can be called without problems from PAGE TASK, but NOT ALSO from TASK TIME e TASK PLC and vice versa.
That is because if an INTERRUPT TASK using a function occurs exactly while a COOPERATIVE TASK is running in the same function, that could lead to abnormal operations in the application.

SHARING OF VARIABLES
Again in CONCURRENT programming can also occur some problem when variables are shared between INTERRUPT TASKS and COOPERATIVE TASK. Practically if managing of the variable don't provide an ATOMIC ASSEMBLER INSTRUCTION, this can cause false reading value when it is written by a TASK and read by another. According to the CPU type of the system these problems can occur in the following type of variables:

Harwdare	Type
ALL	FLOAT

To overcome this problem VTB offers the possibility of a SECURE SHARING OF VARIABLES. Indeed in the variables declaration dialog there is an apposite field to enable the secure sharing. However, because a lot of use of this facility can generate jitter problem we recommend to ***use the enable of secure sharing of variables only when ABSOLUTELY NECESSARY.***
The same problem could also occur when using data array shared by more process. A simple example can be the use of array to data exchange in MODBUS protocol. These problems can arise when, for example, the writing process of data and the reading one are asynchronous. It can happen indeed that a reading process starts when the writing one has filled the array only partially. In this case the reading process will read a lot of new data and some from the old scan. It's evident in this situation false value readings can occur. System isn't able to understand these situations therefore to solve it there is the needs of ***semaphores*** at application level.
Task plc has also an INIT section. All code insert here will run only one time at system reset.

4.2 Task Time

The TASK Time is started after the TASK PLC. It has ONE Section

4.2.1 TASK TIME_Cycle

TASK TIME, like TASK PLC, works at fixed time. It differs from that for two features:

a) it has a lower priority and it can be INTERRUPTED by TASK PLC;

b) it hasn't limit to managing of the IFS functions of VTB.

The scan time of this task is programmable at multiple of the sampling time of TASK PLC. TASK TIME is useful for the managing of timed cycles and with medium response time, furthermore the possibility of calling all IFS functions makes it of great utility, ensuring constant time to software. Typical sample time can be about 5 or 10 milliseconds, with witch it's possible to manage a complex PLC cycle with a lot of I/O channels. If the elapsed time of this task overcomes its sample time the system will continue to work stopping the cooperative tasks but task plc will continue to run.

4.3 Task Main

The Task Plc has two section**. TASK MAIN_Init** and **TASK MAIN_Cycle**

4.3.1 TASK MAIN_INIT

It is the **BEFORE** the Task_Main Cycle. Normally it contains all Initializations for Task Main_Cycle.

4.3.2 TASK MAIN_Cycle

TASK MAIN is called continuously by VTB cycle running. Therefore a static cycle on TASK MAIN will stop only this TASK and the Task Time and Task Plc will continue to Run. Its scanning time depends by the code contained in all the other TASKS. Usually this TASK manages repetitive cycles as control of emergency or alarm states, graphic control etc. where there isn't the need for constant time. However its scanning time can be very fast, also in the order of few ***microseconds,*** when the code inside the task is very short.

4.4 Functions

There are two sections, **Main_Functions** and **Main_Functions_ObjectsEvents**

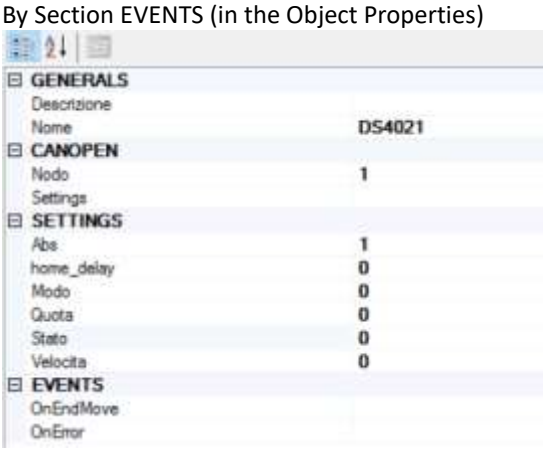
4.4.1 Main_Functions

This section allows to decleare the Functions visible to all tasks.

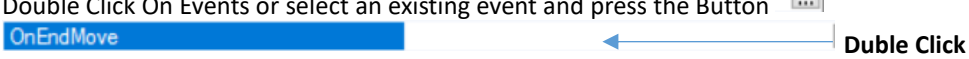
4.4.2 Main_Functions_ObjectsEvents

In this section are inserted automatically the Events instances declared by some objects
For activate an event in an Object see below:

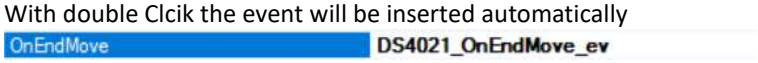
1) Insert the Object (ex Motor Control->CSTDCANOPEN->DS402 – 1.0.2)
2) By Section EVENTS (in the Object Properties)



3) Double Click On Events or select an existing event and press the Button ...



With double Clcik the event will be inserted automatically



5 HARDWARE CONFIGURATION

In this chapter is explained all parameters for hardware configuration

5.1 NGQ/NGQx Configuration

CanOpen

Enable **0** Disabled
1 Enabled

BaudRate Baud Rate

Sync **0** Sync Message Disabled
1 Sync Message Enabled

SlowPx **Set to 0.Reserved**

CheckError **0** Check Errors CanOpen Disabled
1 Check Errori CanOpen CUSTOM
With this option the system doesn't perform any action but it calls some functions to allow the customization of the managing of CanOpen configuration errors.
The functions called by the system are three and they have to be defined by the application:

function open_cancfgerr(nodes as char) as void
nodes = Total number of nodes in the CanOpen configuration.
This function is called by the system before starting the CanOpen configuration. The total number of the nodes in the configuration is written in the parameter ***nodes***

function cancfgerr(node as int, err as uchar) as void
node=Number of node.
err=Result of configuration.
0 = Node correctly configured.
<>0 = Error code. See relative chapter of CanOpen functions.
This is called at the end of configuration of each node writing the result in the parameter ***err***.

function close_cancfgerr() as void
This function is called after the end of the last node configured.

General

EnableEncoder **0** Encoders and Analog outputs Disable on NGQx (This means a NGQ)
1 Encoders and Analog outputs Enabled on NGQx (This means a NGQx)

LinkRPC

LinkType RS232 COM by managed RPC protocol for HOST PC.
0 - None RPC Link
1 - **SER1/PROG** this means that the DEBUG will be disabled
For Application Upload is necessary manual **BOOT/RESET** operation
2 - SER2

BaudRate Baud rate for RPC RS232 (default 115.200)

StepDir

Enable Mask Enable PULSE/DIR channels (Only for NGQ) Bit mapped
Values:
Bit 0 Channel 0 Enabled **Bit 1** Channel 1 Enabled
Bit 2 Channel 2 Enabled **Bit 3** Channel 3 Enabled

Interpolation Mask Enable P PULSE/DIR channels in interpolation mode (Only for NGQ) Bit mapped
Values:
Bit 0 Channel 0 Enabled **Bit 1** Channel 1 Enabled
Bit 2 Channel 2 Enabled **Bit 3** Channel 3 Enabled

5.2 NGMEVO Configuration

Analog Inputs

Enable Mask Enable the analog Inputs Bit Mapped
Values:
Bit 0 Channel 0 Enabled – Digital Input 9 removed
Bit 1 Channel 1 Enabled – Digital Input 10 removed
Bit 2 Channel 2 Enabled – Digital Input 11 removed
Bit 3 Channel 3 Enabled – Digital Input 12 removed
Bit 4 Channel 4 Enabled – Digital Input 13 removed
Bit 5 Channel 5 Enabled – Digital Input 14 removed
Bit 6 Channel 6 Enabled – Digital Input 15 removed
Bit 7 Channel 7 Enabled – Digital Input 16 removed

CanOpen

Enable **0** Disabled
1 Enabled
BaudRate Baud Rate
Sync **0** Sync Message Disabled
1 Sync Message Enabled
SlowPx Set to 0.Rserverd
CheckError **0** Check Errors CanOpen Disabled
1 Check Errori CanOpen CUSTOM
(See **CheckError NGQ**)

General

Modality **Set to 1 Reserved**
Number of Expansions Number of **NGMIO** in the NGMEVO

LinkRPC

LinkType RS232 COM by managed RPC protocol for HOST PC.
0 - RPC Link Enabled on ETHERNET
1 - **SER1/PROG** this means that the DEBUG RS232 will be disabled (Only Ethernet)
For Application Upload is necessary manual **BOOT/RESET** operation
2 - SER2
BaudRate Baud rate for RPC RS232 (default 115.200)
Ethernet IP IP Addres of NGMEVO (default 10.0.0.80)
For all Ethernet connections (RPC,Debug,Modbus TCP/IP)
Ethernet Mask Ethernet subnet mask (default 255.255.255.0)
Gateway Not used

LSYNC

Enable Mask **Set to 0 Reserved**
Prescaler **Set to 6 Reserved**

StepDir

Enable Mask Enable PULSE/DIR channels. Bit mapped
Values:
Bit 0 Channel 0 Enabled **Bit 1** Channel 1 Enabled
Bit 2 Channel 2 Enabled **Bit 3** Channel 3 Enabled
Interpolation Mask Enable P PULSE/DIR channels in interpolation mode. Bit mapped
Values:
Bit 0 Channel 0 Enabled **Bit 1** Channel 1 Enabled
Bit 2 Channel 2 Enabled **Bit 3** Channel 3 Enabled

5.3 NGWARP Configuration

CanOpen

Enable	**0** Disabled
	1 Enabled
BaudRate	Baud Rate
Sync	**0** Sync Message Disabled
	1 Sync Message Enabled
SlowPx	Set to 0.Rserverd
CheckError	**0** Check Errors CanOpen Disabled
	1 Check Errori CanOpen CUSTOM
	(See **CheckError NGQ**)

Ethercat

Enable	**0** Disabled
	1 Enabled

LinkRPC

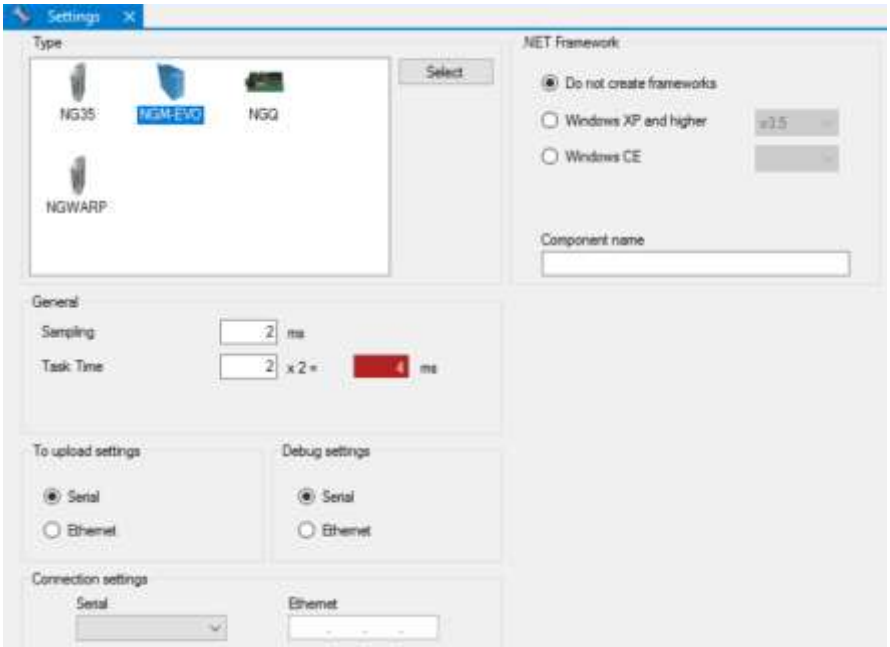
LinkType	RS232 COM by managed RPC protocol for HOST PC.
	0 - RPC Link Enabled on ETHERNET
	1 - **SER1/PROG** this means that the DEBUG RS232 will be disabled (Only Ethernet)
	For Application Upload is necessary manual **BOOT/RESET** operation
	2 - SER2
BaudRate	Baud rate for RPC RS232 (default 115.200)
Ethernet IP	IP Addres of NGMEVO (default 10.0.0.80)
	For all Ethernet connections (RPC,Debug,Modbus TCP/IP)
Ethernet Mask	Ethernet subnet mask (default 255.255.255.0)
Gateway	Not used

6 APPLICATION CONFIGURATION

First to start a new application, is necessary to configure it.

ATIVATION VTBII CONFIGURATION

From Menù ***Project → Settings***



6.1 Type

Hardware selection. This selection is already proposed when the new project is started.
If the Hardware is changed, the current project will be saved.
Select the Hardware type and press **SELECT.**

6.2 General

Project General settings

6.2.1 Sampling

Execution time Tempo of l TASK PLC, EXES CONTROL,CANOPEN or ETHERCAT. Low values, can crashed the application.
Typical 2 Ms for 8 Interpolated Axes.
The TASK PLC TIME (see its time in the DEBUG APPLICATION) must not be over the 60-70% of Sampling time selected.

6.2.2 Task Time

Execution time of TASK TIME. It is referred on Multiple of TASK_PLC.

6.3 Upload Settings

Allows to settings the UPLOAD channel parameters.

6.3.1 Serial

From RS232

6.3.2 Ethernet

From Ethernet (Only NGWARP)

6.4 Debug Settings

Allows to settings the DEBUG channel parameters

6.4.1 Serial

From RS232

6.4.2 Ethernet

From Ethernet (Only NGWARP,NGMEVO)

6.5 Connection Settings

PC COM RS232(for RS232 connection) and IP Adddres (for ETHERNET Coonection)

6.5.1 Serial

PC RS232 port number

6.5.2 Ethernet

Ethernet IP Address

WARING

The IP Address must be appropriate to Hardware IP.
Therefore, if is present a DHCP (from Router) this must distribute a different address to IP selected in the Hardware, but anyway the address must be the some family.
If the DHCP is not present, select the static IP on PC

Ex: IP BOARD=10.0.0.80
IP PC=10.0.0.90

6.6 .NET Framework

VTBII compiler can create a DLL COMPONENT MODEL which can be imported in .NET (dot net) projects. That allows the full control of hardware resource directly by a PC: READ/WRITE VARIABLES, CALL FUNCTION IN REMOTE PROCEDURE CALL.
For details refer to the **NG Framework manual**

6.6.1 Do not create Frameworks

None .DLL framework will be create

6.6.2 Windows XP and Higher

Generate a DLL for Windows XP or higher versions (Windows 7,8,10)
Select the Framework version.
Normally for **XP V 3.5**, for **7,8,10 v4.5**

6.6.3 Windows CE

Generate a DLL for Windows CE
Select the Framework version.

7 TYPE OF VARIABLES

VTB can manage several types of variables which can be used in programming phase.
Commonly all VARIABLES will be allocated in the VOLATILE MEMORY (RAM) of the system and they are zeroed at system boot. In systems equipped with NON-VOLATILE RAM (NGWARP) it's also possible to allocate them in this area, they are defined as STATIC VAR and they will retain its value also after turn-off. VARIABLES follow the STANDARD terminology like to common programming languages.
Furthermore, it can be declared VARIABLES referred to external component like to CANOPEN or ETHERCAT. These are managed automatically from the system in transparent mode.

7.1 Valori Numerici

VTB manages numeric values in conventional mode as other compilators. A numeric value can be written in **DECIMAL NOTATION** as well as in **HEXADECIMAL NOTATION** by preceding the number with the prefix ***0x*** (ZERO X). For example the decimal number 65535 is translated with the hexadecimal 0xFFFF.
FLOATING-POINT values must be written with decimal point and it can not written in hexadecimal format.

Example:

```
A=1236      ' assigning 1236 to variable A
A=0x4d      ' assigning hexadecimal value 0x4d to variable A
            ' corresponding at decimal value 77
B=1.236     ' assigning floating-point value 1.236 to variable B
```

VTBII does not check the variable dimension with the assign value

7.2 Internal Variables

From Project Manager
Variables

Select Tab Var Internal

Press Button:

insert the fields:

Name → Variable name
Type → Select the desired type
Shared → Select True if it is shared in other TASKS
Export → Select the class name for export to NG Framework. Clear this field for not export the variable
Description → Variable description (optional)

These variables are declared in internal volatile RAM

TYPE	DIMENSION	RANGE
BIT	1 bit	From 0 to 1
CHAR	8 bit signed	From –128 to +127
UCHAR	8 bit unsigned	From 0 to 255
INT	16 bit signed	From –32.768 to +32.767
UINT	16 bit unsigned	From 0 to 65.535
LONG	32 bit signed	From –2.147.483.648 to +2.147.483.647
FLOAT	64 bit (standard DOUBLE format IEEE 75)	From -1,79769313486232e308 to +1,79769313486232e308
STRING	Supported only as constant	
VECTOR	Single dimension for all variable types except BIT type	
STRUCTURE	Standard declaration	
POINTER	Char, Uchar, Int, Uint, Long, Float 32 bit	
DELEGATE	Pointer to FUNCTIONS 32 bit	

7.3 BIT Variables

The BIT Variables are declared on an existing Internal Variable

From Project Manager

Select Tab 

:

Choose the internal variable

Press Button: 

Automatically all Bits will be declared (based on variable dimension).

Variable	Name	Description
Var0		
Bit0	Var0_Bit0	?
Bit1	Var0_Bit1	?
Bit2	Var0_Bit2	?
Bit3	Var0_Bit3	?
Bit4	Var0_Bit4	?
Bit5	Var0_Bit5	?
Bit6	Var0_Bit6	?
Bit7	Var0_Bit7	?

Inert desired Name and Description

This type of variable can have only two values: 0 or 1, normally associated to a state OFF/ON or FALSE/TRUE. The variable BIT must always refer to an original variable which will can contain more bits.
These variables are very useful to manage FLAGS, digital I/O lines and in all cases where we need to read or write a single bit directly.
For example declaring an INTERNAL variable named STATE of type INT (16 bit) it's possible to associate it up to 16 bit variables.

VARBIT1STATE.0 (first bit of STATE)
VARBIT2STATE.1 (second bit of STATE)
.
VARBIT16 STATE.15 (16th bit of STATE)

```
If VARBIT1 = 1          ' test if first bit of STATE is set
        VARBIT2=1       ' set second bit of STATE
        VARBIT3=0       'reset third bit of STATE
endif
```

A common use of these variables is the manage of the digital **INPUT** and **OUTPUT** lines of the system, as they are equipped inside system (ex. NGIO) or they are remote channels in a **CANOPEN** or **ETHERCAT** net. That means declaring the bit variables we shell control physically the state of these I/O lines simply reading or writing the relative bit variable.

7.4 Define

The Define, is a constant value (numeric or alphanumeric), that can be used inside the source code.
When inside the code will be found the DEFINE, it will be replaced with DEFINE VALUE (like to C)

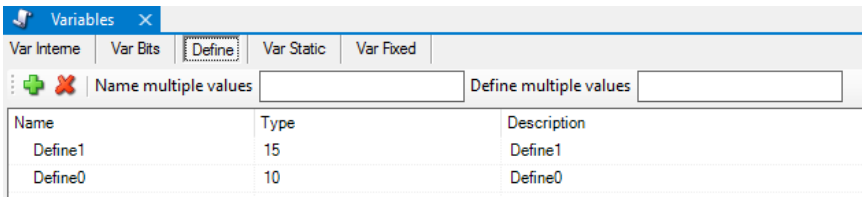
From Project Manager


Select Tab 

Press Button: 

Insert the fields:
Name → DEFINE name
Type → DEFINE return Value
Description → DEFINE description (optional)

Example:



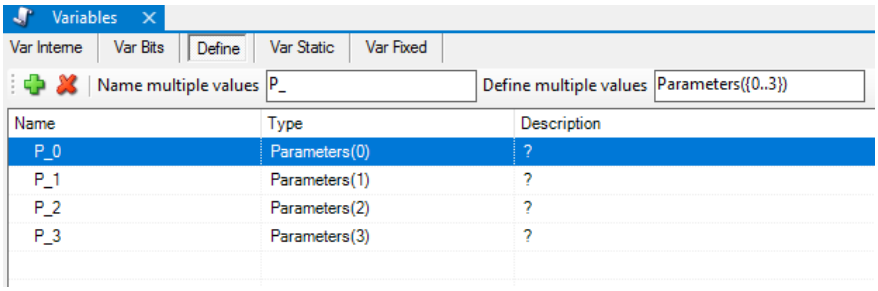

```
If Var0=Define0          ' Like to 10
        ....
        ....
Endif

If Var0=Define1          ' Like to 15
        ....
        ....
Endif
```

Is possible to insert multiple values with an index in the defines list, by entering data in the boxes **Name multiple values** e **Define multiple values** in this mode:

- In **Name multiple values** box, enter the initial name of the define
- In the **Define multiple values** box enter the value of the define with the reference **{N0..Nn}** where you want to insert the growing index. The values of **N0** and **Nn** must be respectively the start index and the final index.

Un esempio di inserimento è visualizzato nella seguente immagine:



If the value of **Define multiple values** is not present, will be insert a default value in the list

7.5 Fixed Variables

Variables with FIXED ADDRESS

From Project Manager

Select Tab 

Select Type:

Select the position for FIXED:

Press Button: 

Insert the fields:

Name → FIXED name
Export → Select the class name for export to NG Framework. Clear this field for not export the variable
Description → DEFINE description (optional)

The FIXED variables are allocated at a fixed address in the internal memory of the device which, unlike common variables, doesn't change modifying the program. This type of variable simplifies the use of systems connected to an external HOST (ex. PC). In fact using FIXED variables there will be no need to recompile the HOST application at each change in VTB program.
FIXED variables are always GLOBAL that is visible in all page and in all tasks.

TYPE	DIMENSION	RANGE
BIT	1 bit	From 0 to 1
CHAR	8 bit signed	From –128 to +127
UCHAR	8 bit unsigned	From 0 to 255
INT	16 bit signed	From –32.768 to +32.767
UINT	16 bit unsigned	From 0 to 65.535
LONG	32 bit signed	From –2.147.483.648 to +2.147.483.647
FLOAT	64 bit (standard DOUBLE format IEEE 75)	From -1,79769313486232e308 to +1,79769313486232e308

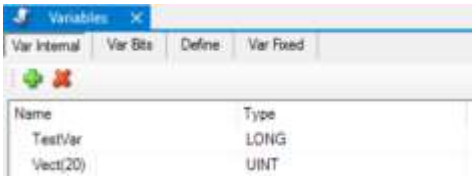
The START address of FIXED area is:

NGMEVO ***Addr = 536874496***
NGWARP ***Addr = 1051648***
NGQ-NGQx ***Addr = 8389632***

7.6 Pointers

VTB is able to manage the pointers to variables too. Pointers defines the address of allocation memory of the variables, not its content. Some VTB functions need of pointers as parameter particularly when the function for manage arrays or strings. To define the address of a variable it's enough insert the postfix ***()*** except for the functions.

Example:



TestVar() *'refers to the address of variable TestVar*
vect() *'refers to the address of the first element of array*

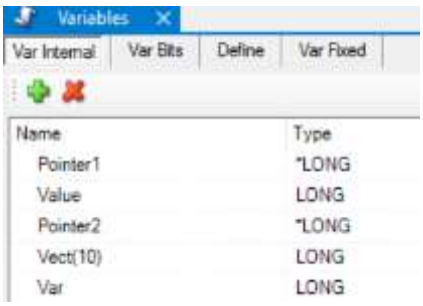
Pointers can be declared only to following types:
Char, Uchar, Int, Uint, Long, Float, Functions

The Pointer declaration is like to Internal Varaibles **, but selecting pointer type (*)**

To assign an address to the pointer it's need:
refer to the name of pointer (without brakes)
assign the desired address to pointer

To assign the value to a pointed field it's need:
refer to the pointer with square brackets
put the right index inside the brackets
assign the value

Example

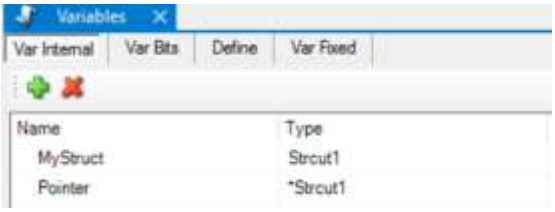


Read/Write from Pointere:
Pointer1=value() **' Assign to Pointer1 the addr of Value**
Pointer1[0]=2000 **'Assign to Value=2000**
Var= Pointer1 [0] **'Assign to Var Value**

Read/Write in Array from Pointer:
Pointer2=Vect() **'Assign to Pointer2 the addr of Vect**
Pointer2 [0]=13
Pointer2 [1]=27
Pointer2 [9]=55 **'Assign the Array Vect from pointer**
Var= Pointer2 [7] **'Assign to Var Vect[7]**

The pointer can be used also in the Structure.

Example

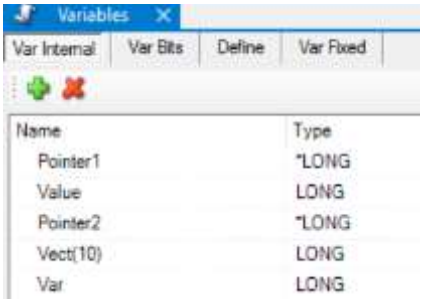


Pointer=MyStruct() ***'Pointer to MyStruct***
Pointer ->str1=300 ***' Str1***
Pointer ->str2=200 ***' Str2***

As we have seen, to use pointer with the structures we need the **token →**

WARNING: VTBII doesn't make any control on the index of pointer therefore **with pointers it's possible to write anywhere in memory with consequent risks to crash the system**.

Example:



Pointer1=Value()
Pointer1 [10]=50

The instruction Pointer1[10] = 1234 doesn't generate any compiling or run-time error, but it can cause unexpected operations. The correct use is **Pointer1** ***[0]=50***

To get the address of a function to assign to a variable we have to refer at the function simply with its name (without brackets):

Example
VarPunt=MyFun

Where MyFun is a Function decelared

7.7 Array

The arrays can be declared in the INTERNAL or STATIC variables and they can be defined as any type except the BIT one. The arrays managed by VTBII are SINGLE-DIMENSION and the maximum limit depends on the free memory available. To declare an array we have to do as for a normal variable putting after the name, between parenthesis, the desired dimension.
If there was the need to use a TWO-DIMENSION array (matrix) we have to work with STRUCTURES. Simply we have to declare a structure with a field of type array then to declare an array of type structure.

ARRAY(10) Array of 10 elements

The first element of the array always start from 0 (zero) then:

ARRAY(0) first element
ARRAY(9) last element

Some VTB functions need the address of the array, that is specified writing the name of array followed by parenthesis with no index inside (see also pointer).

ARRAY() refers to the memory address of ARRAY

DECLARING AN ARRAY

Name	Type
Vect(20)	LONG

WARNING: VTBII doesn't make any control on the index of array therefore **with it's possible to write over the array's dimension with consequent risks of unexpected operations**.

7.8 System Variable

Variables of type System are variables already defined by operative system, therefore we must not to declare them but they can be used as like to variables. This is the list of the SYSTEM VARIABLES available. There are more system variables but reserved to the system.

NOME	TIPO	R/W	FUNZIONE
_SYSTEM_PXC	LONG	R/W	Utilizzate nei sistemi con NGMEVO. Contengono il doppio del numero di passi generati dai 4 assi P/P presenti.
_SYSTEM_PYC	LONG	R/W	
_SYSTEM_PZC	LONG	R/W	
_SYSTEM_PAC	LONG	R/W	
_SYSTEM_EMCY(8)	CHAR	R	Contiene il dati relativi al pacchetto Emergency Object del CanOpen. Viene aggiornata tramite la funzione ***read_emcy()***.
_SYSTEM_SDOAC0	LONG	R	Contengono gli 8 byte dell'eventuale SDO ABORT CODE inviato da uno slave CANOPEN a seguito di una chiamata alle funzioni ***pxco_sdodl(...)*** o ***pxco_sdoul(...)***. Se queste ritornano con errore 2, nelle variabili _SYSTEM_SDOAC0 e _SYSTEM_SDOAC1 è presente il codice di errore.
_SYSTEM_SDOAC1	LONG	R	
_SYSTEM_PLC_ACT_TIME	UINT	R	Contiene il tempo attuale di scansione della TASK PLC in cicli MACCHINA. Per riportarlo in Millisecondi occorre moltiplicare il valore per una costante dipendente dal tipo di CPU. Serve in fase di DEBUG per capire la durata del TASK PLC. Questo tempo deve essere inferiore del 30% del Parametro CAMPIONAMENTO (inserito nelle opzioni generali) per evitare rallentamenti negli altri task.
_SYSTEM_PLC_MAX_TIME	UINT	R	E' simile al precedente e rappresenta il picco massimo memorizzato.
_SYSTEM_VER	INT	R	Ritorna la versione del firmware. Es. 10317 → Vers. 1.03.17
_SYSTEM_CANERR_CNT0	LONG	R/W	Contatore errori linea Canopen canale 1 Vengono contati tutti gli errori di trasmissione che la linea presenta
_SYSTEM_CANERR_CNT1	LONG	R/W	Contatore errori linea Canopen canale 2 Vengono contati tutti gli errori di trasmissione che la linea presenta
_SYSTEM_ECERR_CNT	LONG	R/W	Contatore errori linea ETHERCAT Vengono contati tutti gli errori di trasmissione che la linea presenta

7.9 Static Variables

The variables of type STATIC are declared in NON-VOLATILE RAM: they aren't zeroed at reset and maintain their value also after turn off. They are very useful to retain data which change frequently (as encoders, counters, etc.), and which could not be saved in flash memory (IMS). Besides they are common variables.
STATIC variables are always GLOBAL that is visible in all page and in all tasks.

TYPE	DIMENSION	RANGE
BIT	1 bit	From 0 to 1
CHAR	8 bit signed	From –128 to +127
UCHAR	8 bit unsigned	From 0 to 255
INT	16 bit signed	From –32.768 to +32.767
UINT	16 bit unsigned	From 0 to 65.535
LONG	32 bit signed	From –2.147.483.648 to +2.147.483.647
FLOAT	64 bit (standard DOUBLE format IEEE 75)	From -1,79769313486232e308 to +1,79769313486232e308
ARRAY	Single dimension for all variable types except BIT type	
DELEGATE	Pointer to FUNCTIONS 32 bit	

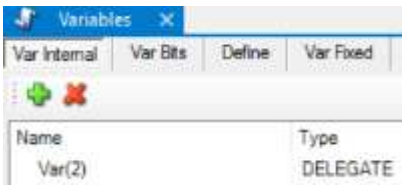
ATTENZIONE:
ONLY NGWARP CAN USE THE STATIC VARIABLES

7.10 Delegate

This type of variables is used to call a function by a variable. First of all the address of the function to call must be written in the DELEGATE variable. Then we can use this variable to call the function with the instruction ***call_delegate***. It can also be created an array of DELEGATE variables and then call a function according to the index of the delegate.
Using of DELEGATES is very powerful because it allows the access to the functions in the fastest way without writing a long series of conditional cycles.

WARNING: The function called by CALL_DELEGATE must be VOID both for arguments and return parameter.
VTB doesn't make any control to the initialization of the DELEGATE. **Calling a delegate not initialized can go the system in CRASH**

Example:



Init of Main (Delegate inititialization):

```
Var(0)=function1          'Assign to Var(0) Function1 Addr
Var(1)= function2         'Assign to Var(1) Function2 Addr
```

Functions declared on Main Functions:

```
Function function1() as void
.
Endfunction

Function function2() as void
.
Endfunction
```

Main Cycle call Delegate:

```
Call_delegate var(0)      'Call Function1
Call_delegate var(1)      'Call Function2
```

7.11 Structure

The STRUCTURES can be declared only as INTERNAL variables. The fields of a structure can be of any type except BIT and pointer. To declare a STRUCTURE open the STRUCTURE TABLES and define the NAME of the structure and all single elements we need.

When a structure is declared, in the list of the variable types the NAME of the STRUCTURE will be showed, allowing to define a new variable of all types declared as structure.
(See Project Manager)

Add a New Structure

Name	Type	Description
Struct1		

Insert the structure name

Remove the selected Structure

Insert a new Item for the selected structure

Name	Type	Description
Strcut1		
Var1	LONG	Descr Var 1

Inert

Name	Name for the structure Variable della variabile dell struttura
Type	Type of Variable
Description	Description (optional)

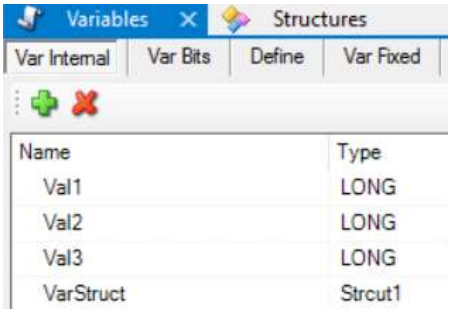
Remove the selected Item from the structure

Variables | Structures

Name	Type
Strcut1	
Str1	LONG
Str2	INT
Struct2	
Item1	CHAR
Item1	LONG

To use the elements of the structure it's necessary to write the NAME of the STRUCTURE followed by **dot** character (**.**) and by the name of the field at which we want to refer.
It's also possible manage the structures with pointers (see Pointers).

Example:



VarStruct1.Str1=13
val1= VarStruct1.Str1
VarStruct1.Str2=23

8 OPERATORS

The operators of VTBII are common to other compilers.

8.1 Logic and Mathematical Operators

These are all the logic and mathematical operators available in VTBII:

OPERATOR	DESCRIPTION	EXAMPLE
(	Parenthesis	It identifies the begin of a group of calculation or function ***a=(c+b)/(x+y)*** ***fun(10,20)***
+	Addition	Mathematical addition ***a=b+c***
-	Subtraction	Mathematical subtraction ***a=b-c***
*****	Multiplication	Mathematical multiplication ***a=b*c***
/	Division	Mathematical division ***a=b/c***
)	Parenthesis	It identifies the end of a group of calculation or function ***a=(c+b)/(x+y)*** ***fun(10,20)***
>	Greater	Greater than condition ***if a>b***
<	Less	Less than condition ***if a<b***
>=	Greater Equal	Greater or equal than condition ***if a>=b***
<=	Less Equal	Less or equal than condition ***if a<=b***
<>	Not equal	Not equal condition ***if a<>b***
=	Equal	Equal condition ***if a=b*** or assignment ***a=b***
||	Logic OR	OR logic condition ***if (a=b) || (b=c)*** condition it's true if at least one expression is true
&&	Logic AND	AND logic condition ***if (a=b) && (b=c)*** condition it's true if both expressions are true
|	OR bit	Execute the OR between two value ***a=a|3*** Bits 1 and 2 of variable ***a*** are set leaving unchanged the others
&	AND bit	Execute the AND between two value ***a=a&3*** All bit of variable ***a*** are reset except the bits 1 and 2
!	Logic NOT	Negation of an expression i**f !(a<>b)** The expression is true if ***a*** is equal to ***b***
~	NOT bit	Execute a not on all the bits of a value, all bits will change its state **a=85 a=~a** After NOT instruction the variable ***a*** will take the value 170 **85 → 01010101** **170 → 10101010**
>>	Shift to right	The bits of the variable are shifted to left n times **a=8 a=a>>3** After shift the variable ***a*** will take the value 1
<<	Shift to left	The bits of the variable are shifted to right n times **a=1 a=a<<3** After shift the variable ***a*** will take the value 8

8.2 Notes on Expressions

VTBII manages the mathematical expressions completely. Anyway we have to make WARNING when in the expression there are INTEGER variables together FLOAT variables. We have to remind these rules:

1) If in the expression there is at least one variable of type FLOAT all the expression is calculated in FLOAT;

2) If the result of an expression must be FLOAT at least one variable in the expression must be FLOAT;

Look at this example:

A=10
B=4
R=A/B

According to the type of the variables VTB calculates the following results:

A	B	R	
LONG	LONG	FLOAT	2
FLOAT	LONG	FLOAT	2,5
FLOAT	FLOAT	LONG	2

Enabling the Warning level of the compiler, some messages will be displayed in coincidence with the possibility of data truncation.

9 MATH FUNCTIONS

VTB manages a wide SET of mathematical functions.

9.1 SIN

Return the **sinus** of an angle in a FLOAT value.
Hardware ***All***

Syntax

Sin (***angle***) as float

The argument ***angle*** can be a FLOAT value or any numeric expression which represents the **angle in radians**.

Example:
Used variables:
angle float
Cosec float
angle = 1.3 ***' Define the angle in radians.***
cosec = 1 / Sin (angle) ***' Calculate the cosecant.***

9.2 COS

Return the **cosinus** of an angle in a FLOAT value.
Hardware ***All***

Syntax

Cos (***angle***) as float

The argument ***angle*** can be a FLOAT value or any numeric expression which represents the **angle in radians**.

Example:
Used variables:
angle float
sec float
angle = 1.3 ***' Define the angle in radians.***
sec = 1 / Cos (angle) ***' Calculate the secant.***

9.3 SQR

Return the **square root** of a number.
Hardware ***All***
Syntax

Sqr (***number***) as float

The argument ***number*** can be a FLOAT value or any numeric expression greater or equal than zero.

Example
Used variables:
vsqr float
vsqr = sqr (4) ***' return the value 2***

9.4 TAN

Return the **tangent** of an angle in a FLOAT value.
Hardware ***All***

Syntax

Tan (***angle***) as float

The argument ***angle*** can be a FLOAT value or any numeric expression which represents the **angle in radiant**.

Example:
Used variables:
angle float
ctan float

```
angle = 1.3                 ' Define the angle in radians.
ctan = 1 / Tan (angle)      ' Calculate the cotangent.
```

9.5 ATAN

Return the **arctangent** of a number in a FLOAT value between -π/2 and +π/2.
Hardware ***All***

Syntax

Atan (***number***) as float

The argument ***number*** can be a FLOAT value or any numeric expression.

9.6 ASIN

Return the **arcsin** of a number in a FLOAT value.
Hardware ***All***

Syntax

Asin (***number***) as float

The argument ***number*** can be a FLOAT value or any numeric expression between 1 and -1.

Example
Used variables:
angle float
var float

```
angle = 1.3
var = asin (angle)
```

9.7 ACOS

Return the **arccos** of a number in a FLOAT value.
Hardware ***All***

Syntax

Acos (***number***) as float

The argument ***number*** can be a FLOAT value or any numeric expression between 1 and -1.
Example
Used variables:
angle float
var float

```
angle = 1.3
var = acos (angle)
```

9.8 ATAN2

It's similar to atan but it returns a value from -π and +π .

Hardware ***All***

Syntax

Atan2 (***y, x***) as float

The arguments **y** and **x** are of type FLOAT.

Return Value

The return value coincides with the angle whose tangent is **y / x.**

Example

Used variables:

```
x float
y float
angle float
radians float
result float
PI float

PI= 3.141592
x=1.0
y=2.0
angle = 30
radians = angle * (PI/180)
result = Tan(radians)        ' Calculate the tangent of 30 degree
radians = Atan(result)       ' Calculate the Arctangent of the result
angle = radians * (180/PI)
radians = Atan2(y, x)        ' Calculate the Atan2
angle = radians * (180/PI);
```

- ## ABS

Return the absolute INTEGER value

Hardware ***All***

Syntax

Abs (***number***) as long

The argument ***number*** can be a LONG value or any numeric expression.

Example

Used variables:

```
Num long

Num = -3250
Num = Abs(Num)' return the value 3250
```

9.9 FABS

Return the absolute FLOAT value

Hardware ***All***

Syntax

FAbs (***numero***) as float

The argument ***number*** can be a FLOAT value or any numeric expression.

Example

Used variables:

Num float

Num = -3.250

Num = Abs(Num)' return the value 3.250

10 INSTRUCTIONS TO CONTROL THE PROGRAM FLOW

In VTB there are a lot of instruction to control the program flow. They are similar to other compiler and **THEY ARE AVAILABLE IN ALL THE HARDWARE TYPES.**

10.1 IF-ELSE-ENDIF

Allow the conditional execution of a group of instruction according to the result of an expression.

Syntax

```
If condition
        [instruction]
Else
        [instructionelse]
endif
```

The syntax of instruction **if... else** is composed by the following elements:

condition Mandatory. Any expression with the result True (value not zero) or False (value zero).
instruction List of the instruction to execute if the condition **IF** is TRUE.
instructionelse Optional. List of the instruction to execute if the condition **IF** is FALSE.
endif End of cycle **IF ELSE**

Notes

The instruction **Select Case** can be more useful when there are a lot of continuous cycles IF because it creates a source code more readable.

Example

Used variables:

```
var1 int
var2 int
if var1*var2 > 120
        var1=0
else
        var1=120
endif
```

10.2 LABEL

Identifies a reference point for the **GOSUB** or **GOTO** jumps.

Syntax

```
Label    labelname
```

labelname name of the reference of the LABEL.

WARNING: The LABEL instruction is OBSOLETE. It is preferred to use the FUNCTIONS.

Example

```
if condiition
        goto label1
else
        goto label2
endif
.
Label Label1
.
Label Label2
```

10.3 GOSUB-RETURN

Allow to pass the control to a SOUBRUTINE and to return at the next program instruction.

Syntax

GoSub labelname

The argument ***labelname*** can be any LABEL inside the current PAGE or inside the MAIN task.

Notes

GoSub and **Return** can be used everywhere in the code. A subroutine can be composed by more than one **Return** instructions, but the first **Return** founded by the program flow will act the return of the program to the first instruction after the last **GoSub.**.

WARNING: The LABEL instruction is OBSOLETE. It is preferred to use the FUNCTIONS.

Example

```
if condition
        gosub label1
else
        gosub label2
endif

Label Label1
Return
Label Label2
Return
```

10.4 GOTO

Allows to jump to a LABEL.

Syntax

Goto ***labelname***

The argument ***labelname*** can be any LABEL.

Notes

Goto passes the control to a point of the program referenced by a LABEL. Unlike GOSUB the instruction **RETURN** isn't necessary.

WARNING: The LABEL instruction is OBSOLETE. It is preferred to use the FUNCTIONS.

Example

```
if condition
        goto label1
else
        goto label2
endif

Label Label1
.
Label Label2
```

10.5 INC

Increments a variable of any type.

Syntax

Inc ***varname***

The argument ***varname*** can be any variable declared in the program.

Description

Inc is the same as **VAR=VAR+1** but it is executed more quickly.

Example

INC var1'var1 is incremented by 1

10.6 DEC

Decrements a variable of any type.

Syntax

Dec ***varname***

The argument ***varname*** può essere una qualsiasi variabile dichiarata nel programma.

Description

Dec is the same as **VAR=VAR-1** but it is executed more quickly.

Example

DEC var1 ' var1 is incremented by 1

10.7 SELECT-CASE-ENDSELECT

Allow to execute blocks of instructions according the result of an expression.

Syntax

```
Select expression
        [Case condition_1
                [instruction_1]] ...
        [Case condition_2
                [instruction_n]] …
        …
        [Case Else
                [instructionelse]]
EndSelect
```

The syntax of the instruction **Select Case** is composed by the following elements:

expression	Mandatory. Any expression.
condition_n	Mandatory. It can be in two forms: **expression**, **expression To expression**. The keyword **To** specifies a range of value.
instruction_n	Optional. Instructions executed if the **expression** matches the **condition_n.**
instructionelse	Optional. Instructions executed if no **condition_n** is matched.

Notes

If the result of **expression** equals a **condition_n**, the following instructions will be executed until the next instruction **Case** or **Case Else** or **EndSelect**.

If more than one **condition_n** is matched, only the first encountered will be execute. **Case Else** is used to execute a block of instruction if no condition are verified. Although it isn't mandatory, it is recommended the use of **Case Else** statement in each **Select** to manage also unexpected value of **expression.**

More instruction **Select Case** can be nested. At each instruction **Select Case** there must be an associated **EndSelect**.

Example

Used variables:

```
var1 int
var2 int
var3 int
Select var1
        case 10             ' if var1=10
                ...
        case var2+var3      ' if var1=var2+var3
                ...
        case 5 TO 20                ' if var1 is between 5 and 20
                ...
        case 1,6,8                  ' if var1=1 or var1=6 or var1=8
                ...
        case else                   ' all other value of var1
                ...
Endselect
```

10.8 FOR-NEXT-STEP-EXITFOR

Allow the iteration of a block of instructions for a number of times according to a variable. It is a mix between BASIC and C languages.

Syntax

```
        For counter = init To condition [Step increment]
                [intructions]
                …
                ExitFor
                …
        Next [counter]
```

The syntax of the instruction **For...Next** is composed by the following elements:

counter Mandatory. Numeric variable used as counter of iteration. It can be a BIT variable.
init Mandatory. Initial value of the counter.
condition Mandatory. Iteration will continue until condition is true.
increment Optional. Value added to the counter at the end of each iteration. If it isn't specified it will assume the value 1. It can be any numeric expression and can assume any value positive as well as negative.
instructions Optional. Block of instructions to execute during the iteration.

ExitFor It is used to force the stop of the iterations, the program will continue from the line immediately after the instruction **Next**.

Notes

It is possible to nest more cycles **For...Next** Assigning to each cycle a different counter:

Examples

```
For I = 1 To I<10
        For J = 1 To J<10
                For K = 1 To K<10
                        ...
                Next K
        Next J
```

Next I

```
For var1=0 to var1<8        ' Repeat 8 times
        ...
Next var1

For var1=1 to var1<var4 step var3
        ...
Next var1

For var2=1 to var2<=10
        ...
Next var2

For var1=10 to var1<var3*var4 step –1
        ...
Next var1
```

10.9 WHILE-LOOP-EXITWHILE

Allow the execution of a block of instructions until a condition is true.

Syntax

```
While condition
        [instructions]
        …
        ExitWhile
        …
Loop
```

The syntax of the instruction **While...loop** is composed by the following elements:

condition Mandatory. Any expression with the result True (value not zero) or False (value zero).
instructions Optional. Block of instructions executed until condition is true.

ExitWhile It is used to force the stop of the cycle, the program will continue from the line immediately after the instruction **Loop**.

Notes

If the condition is True, the block of instruction will be executed then yhe cycle will be repeated.
More cycles **While...loop** can be nested at any level. Each instruction **loop** will correspond to the more recent instruction **While.**

Example

Used variables:
Var1 int

```
while var1<10
...
loop
```

11 FUNCTIONS

VTB manages functions with the same syntax as VISUAL BASIC. It exist a limitation in the declaration of internal variables: **they can not be ARRAYS, STRUCTURES or BITS.**

11.1 Declaration of a function

Syntax

```
function function_name(par_1 as int, par_2 as char, ....., par_n as *long) as function_type
        dim var as int      'local variables
        ....
        ....                'body of the function
        ....
        function_name = return_value
endfunction
```

The syntax of a **function** is composed by the following elements:

function	Mandatory. Keyword identifying the begin of a function.
function_name	Mandatory. Unambiguous name of the function chosen by programmer.
par_1...par_n	Optional. They are the parameter passed to the function. If no parameter have to be passed (VOID) there must be nothing inside the parenthesis.
function_type	Mandatory. It defines the data type returned from the function. If no data have to be returned write as void.
local variables	Optional. Local variables are allocate at the moment when function is called and then destroyed when it returns. They can be of any types **except ARRAYS, STRUCTURES or BITS.**
body of the function	Optional. Block of instruction execute by the function.
function_name=…	Optional. It assigns the value returned from the function.
endfunction	Mandatory. Keyword to identifying the end of the function.

Notes

A function can be called simply writing its name passing to it the eventual parameters declared.
To return from the function in any moment it can be used the instruction **return**.
The assignment **Function_Name = ….** doesn't cause the return from the function but only the assignment of the return value.

Example:

Used variables:

```
result as int
number_a as int
number_b as int
```

Page Function of Main task (functions declaration):

```
function int_average(number_1 as int, number_2 as int) as int
        dim temp as int
        temp=(number_1+number_2)/2
        int_average=temp
endfunction
```

Anywhere in the source code (function calling):

```
number_a=13
number_b=33
result=int_average(number_a, number_b)
```

11.2 Internal Function variables

Syntax

Dim varname as type

The syntax of instruction **dim** is composed by the following elements:

varname Mandatory. Name of the variable.
type Mandatory. Type of the variable. It can be of any types **except ARRAYS, STRUCTURES or BITS.**

Example
dim var as long
dim var1 as uint
dim var2 as float

12 SYSTEM FUNCTIONS

VTB provides a wide LIBRERY to a complete management of the hardware devices. Some function can be available only for some type of hardware

12.1 FUNCTIONS FOR THE SERIAL PORT CONTROL

All Promax hardware devices have 1 or 2 serial channel available to the application.
In VTB there are some object to manage the common serial protocol, for example MODBUS protocol both MASTER and SLAVE. However it's possible to use one serial channel to customize the protocol.
To do that there are some API function **which always refer to the SECOND SERIAL PORT of the hardware.**

12.1.1 SER_SETBAUD

Programming the BaudRate of the second SERIALE PORT.
Hardware ***All***

Syntax

SER_SETBAUD (long Baud)

Parameters

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

12.1.2 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 bits per character, 1 stop bit.
Hardware ***All***

Syntax

SER_MODE(char par, char nbit, char nstop)

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)

Example

```
ser_mode(1,8,2)        ' Program the 2nd serial port with:
                       ' ODD-PARITY, 8 BIT/CHAR 2 STOP-BIT
```

12.1.3 SER_GETCHAR

Reads the receive buffer of the serial port. It doesn't wait for the presence of a character.
Hardware ***All***

Syntax

SER_GETCHAR () as int

Return value:

-1 ***No character is in the buffer***
>=0 ***Code of the character read from the buffer***

12.1.4 SER_PUTCHAR

Sends a character to the serial port.

Hardware ***All***

Syntax

SER_PUTCHAR (int CodeChar)

Parameters

CodeChar Code of the character to send

12.1.5 SER_PUTS

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

Hardware ***All***

WARNING: This function can not be used in a BINARY transmision but only with ASCII transmision.

Syntax

SER_PUTS (char *str)

Parameters

***str** Pointer to the string

Example

```
Ser_puts("TEXT MESSAGE")        ' Send the string TEXT MESSAGE
Strcpy(Vect(),"MESSAGE1")       ' Copy the string MESSAGE1 to Vect
Ser_puts(Vect())                ' Send again the string TEXT MESSAGE
```

12.1.6 SER_PRINTL

Formatting print of an INTEGER value.

Hardware ***All***

Syntax

SER_PRINTL (const char *Format, long Val)

Parameters

Format String corresponding to the format to be printed

Val Any integer value or expression

Avalaible 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

Example

```
var=12345
ser_printl("###.##",var) ' It will be printed: "123.45"
var=2
ser_printl("###.##",var) ' It will be printed: "  . 2"
ser_printl("###.00",var) ' It will be printed: "  .02"
ser_printl("##0.00",var) ' It will be printed: "  0.02"
```

12.1.7 SER_PRINTF

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

Hardware ***All***

Syntax

SER_PRINTF (const char *Format, float Val)

Parameters

Format String corresponding to the format to be printed

Val Any integer value or expression

12.1.8 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.

Hardware ***All***

WARNING: This function allows to manage BINARY and RS485 protocols.

Syntax

SER_PUTBLK (char *Buffer, int Len)

Parameters

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

Len Number of bytes to send

Example

```
Ser_putblk(Vect(),11)          ' Send 11 bytes of array vect
```

12.1.9 SER_PUTST

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

Hardware ***All***

Syntax

SER_PUTST () as int

Return value:

-1 ***Transmit error***

>=0 ***Number of characters to be transmitted***

Example

```
Ser_putblk(Vect(),11)     ' Send 11 bytes
while Ser_putst()         ' Wait for the complete transmission
loop
```

12.2 MISCELLANEOUS API FUNCTIONS

12.2.1 GET_TIMER

Reads the system timer in units of TASK PLC (scan time).

Hardware ***All***

Syntax

Long GET_TIMER ()

Return value:

Value of the system timer in sampling units

Some defines are automatically generated by VTB to adapt the application at the scan time:

TAU	Scan time of TASK PLC in milliseconds (INTEGER value)
TAUFLOAT	Scan time of TASK PLC in milliseconds (FLOAT value)
TAUMICRO	Scan time of TASK PLC in 0.1 milliseconds

Example

Used variables:

```
Tick long
Tick=Get_timer()                    ' Get initial value of timer
while Test_timer(Tick,1000/TAU)     ' Waiting for 1 second
Loop
```

12.2.2 TEST_TIMER

Compares the system timer with a value. It is used together the function ***get_timer*** to make timing.

Hardware ***All***

Syntax

char TEST_TIMER (long Timer, long Time)

Parameters

Timer	Initial value of system timer
Time	Time to compare

Return value:

1= time elapsed

0=time not elapsed

Example

Used variables:

```
Tick long
Tick=Get_timer()                    ' Get initial value of timer
while Test_timer(Tick,1000/TAU)     ' Waiting for 1 second
Loop
```

12.2.3 ALLOC

Dynamic allocating of memory area.

Hardware ***NGWARP***

Syntax

ALLOC (Long Mem) as long

Parameters

Mem	Total amount of memory to be allacated

Return value:
<>0 Pointer to the allocated memory
0 Allocation error

Example

```
Pnt As *Char
N as Long
Pnt=Alloc(3000)          ' Alloc 3000 byte of memory
FOR N=0 to N<3000
        PUNT[N]=N
NEXT N
```

12.2.4 FREE

Frees the a memory area previously allocated with ***alloc***.
Hardware ***NGWARP***

Syntax

Free (Char *Pnt)

Parameters
Pnt Pointer to the memory to free

Example

```
Pnt As *Char

Pnt=Alloc(3000)          ' Alloc 3000 bytes of memory
....
....
Free(pnt)                ' Free the memory
```

12.2.5 SYSTEM_RESET

Executes a software RESET on the hardware.
Hardware ***All***

Syntax

SYSTEM_RESET (Char mode)

Parameters
mode **=0** Executes a normal RESET running the application
=1 Executes a RESET putting device in BOOT state

12.3 API FUNCTIONS FOR MANAGING OF STRINGS

VTB doesn't use STRING variables, to manage them there are some apposite functions similar to the **"C"** language.

12.3.1 STRCPY

Copies the string pointed by SOURCE into the array pointed by DEST. The string must terminate with the character 0 (NULL).

Hardware ***All***

Syntax

STRCPY (Char *Dest, Char *Source)

Parameters

Dest Pointer to destination
Source Pointer to source

Example

Used variables:
Dest(10) char
Dest1(10) char
strcpy(Dest(),"My Text") ***' copy the string "My Text" in dest***
strcpy(Dest1(),Dest()) **'copy the string "*My Text*" in dest1**

12.3.2 STRLEN

Returns the length of a string.

Hardware ***All***

Syntax

STRLEN(Char *Str) as int

Parameters

Str Pointer to the string

Return value:
Length of the string.

Example

Used variables:
Len int
Len=StrLen("My Text") ***' return value 7***

12.3.3 STRCMP

Comparing of two strings.

Hardware ***All***

Syntax

STRCMP(Char *Str1, Char *Str2) as char

Parameters

Str1 Pointer to the first string
Str2 Pointer to the second string

Return value:
0 ***Equal strings***
< ***String Str1 less than Str2***
>0 ***String Str1 greater than Str2***

12.3.4 STRCAT

Appends a copy of the source string to the destination string.

Hardware ***All***

Syntax

STRCMP(Char *Dest, Char *Source)

Parameters

Dest	Pointer to destination
Source	Pointer to source

Example

Used variables:

Str(30) Char

Strcpy(Str(),"My ")

StrCat(Str(),"Text") ***' str will contain "My Text"***

12.3.5 STR_PRINTL

Converts an INTEGER variable to a characters STRING.

Hardware ***All***

Syntax

STR_PRINTL(Char *Dest, Char *Format, Long Val)

Parameters

Dest	Pointer to the destination string
Format	String corresponding to the format to be printed
Val	Any integer value or expression

Avalaible 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

Example

var=12345

STR_Printl("###.##",var) ' It will be printed: "123.45"

var=2

STR_Printl ("###.##",var) ' It will be printed: " . 2"

STR_Printl ("###.00",var) ' It will be printed: " .02"

STR_Printl ("##0.00",var) ' It will be printed: " 0.02"

12.3.6 STR_PRINTF

Converts a FLOAT variable to a characters STRING.

Hardware ***All***

Syntax

STR_PRINTF(Char *Dest, Char *Format, Float Val)

Parameters

Dest	Pointer to the destination string
Format	String corresponding to the format to be printed
Val	Any float value or expression

Avalaible 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

12.4 FUNCTIONS FOR AXES INTERPOLATION

The axis interpolation functions are contained in an OBJECT in the CLASS COBJINTERPOLA. In this chapter are descriped this function with the primitive name. Remember to put the prefix of the OBJECT NAME. If, for example the object is named ***obj*** the function ***moveto*** will must be called as ***obj.moveto.***

12.4.1 PROPERTY

This is the list of the common properties of the OBJECT COBJINTERPOLA.

N.assi Number of axis to be interpolate. It can be changed only at VTB environment. A DEFINE named **Objname.*Nassi*** is automatically generated with this value.

N.tratti Number of elements in the movement buffer. It can be changed only at VTB environment and **must have a value as power of 2 (4, 8, 16, etc.)**. A DEFINE named **Objname.*Ntratti*** is automatically generated with this value.

.vper Value for the changing of the speed "on-fly". Together ***Div.vper*** form a ratio: when it is 1 the speed corresponds to the set one.

Div.vper Divisor of ***vper***. It can be changed only at VTB environment.

Abilita arcto Usually it is set to 1, if 0 the circular interpolation functions will be not avalaible. It is used to short the code size. It can be changed only at VTB environment.

.acc Acceleration and deceleration. During the execution of ramps, at each sample (TASK PLC) the speed, as unit/sample is incremented (o decremented) of this value. Default value 10.

.sglr Threshold of the radius error. Default value 10.

.sglp Threshold edge 2D as tenth of degree. It is used by moveto and lineto to calculate the presence of an edge on the working plane. Default value 10.(20 degrees).

.sgl3d(NASSI) Threshold edge 3D. Default value 0.2 (for all axis).

.pc(NASSI) Actual calculated value of the axis position.

.cmd Output of virtual axis managed by ***setcmd***.

12.4.2 MOVETO

Movement with linear interpolation. The interpolation is executed at speed **vel**. The parameter **mode** defines if the axis have to stop in the position or continue with the next movement. To do that there is a apposite BUFFER where movement are latched.

Hardware ***All***

Syntax

.MOVETO(Long Vel, Char mode, Long *PntAx) as char

Parameters

Vel Velocity of interpolation as unit/sample

mode Flag to control the stop before the next movement

mode=0 never stop
mode=1 always stop at the end of movement
mode=2 stop only on edge 3D (***sgl3d***)
mode=3 stop only on edge 3D (***sglp***)

PntAx Pointer to the array of the axis position as unit

Return value

Char ***0*** Command not written in the buffer (buffer full)
1 Command written in the buffer

Notes

Moveto is usually used to interpolate more than 2 axes. The speed vector is distributed on all axes to be interpolated. When **mode=2** it is calculated the presence of a multidimensional edge according to the values in ***sgl3d.*** When ***mode=2*** the test of edge is made only on the axis of the working plane and according to the value in ***sglp***. If the comand isn't written in the BUFFER, we have to wait and repeat otherwise it will be lost.

Approximative reference values of parameter SGL3D

THRESHOLD in DEGREE	VALUE OF SGL3D (min-max)
5	60-90
10	125-175
20	250-350
30	300-500
45	400-700

Example (object name = OBJ)

Used variables:

```
VectAssi(4) long
Vel long
Test char
'*****************************************************************************
'Fast interpolation of several segments on axis X,Y holding Z and A stopped
'*****************************************************************************
vel=1000
VectAssi(0)=1000 'X
VectAssi(1)=2000 'Y
VectAssi(2)=OBJ.pc(2) 'Z remain stopped
VectAssi(3)=OBJ.pc(3) 'A remain stopped
muovi()
VectAssi(0)=4000 'X
VectAssi(1)=6000 'Y
VectAssi(2)=OBJ.pc(2) 'Z remain stopped
VectAssi(3)=OBJ.pc(3) 'A remain stopped
muovi()
VectAssi(0)=5000 'X
VectAssi(1)=2000 'Y
VectAssi(2)=OBJ.pc(2) 'Z remain stopped
VectAssi(3)=OBJ.pc(3) 'A remain stopped
muovi()
' ********************************************************
' Movement function waiting if the buffer is full
' ********************************************************
Function muovi() as Void
Dim test as Char
Label Move
test=Obj.moveto(vel,3,VectAssi())
if test=0
        goto Move
endif
EndFunction
```

12.4.3 LINETO

Lineto interpolates the axis distributing the vector speed ONLY ON THE AXES OF THE CURRENT WORKING PLANE. The other axis will be TRANSPORTED.
The function is useful to manage TANGENTIAL AXIS such as cutting machine, where the blade have to be transported to increasing the fluidity of the movement. The eventual stop of axis is calculated according to the threshold value in ***sglp***. If the resultant edge is less or equal than this threshold axis don't stop in the position but continue filleting the two segments.
Hardware ***All***

Syntax

.LINETO(Long Vel, Long *PntAx) as char

Parameters

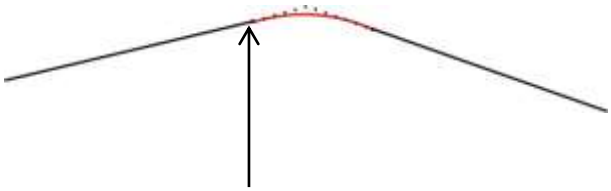
Vel	Velocity of interpolation as unit/sample
PntAx	Pointer to the array of the axis position as unit

Return value

Char	***0***	Command not written in the buffer (buffer full)
	1	Command written in the buffer

Notes

Lineto, unlike Moveto, doesn't distribute the velocity on all enables axis, but only on the working plane making this function not able to tridimensional interpolation.



If the edge is less or equal than SGLP axis don't stop

Example (object name = OBJ)
Used variables:

```
VectAssi(4) long
Vel long
Test char
' Fast interpolation with tansported third axis
vel=1000
VectAssi(0)=1000 'X
VectAssi(1)=2000 'Y
VectAssi(2)=100 'Z transported
VectAssi(3)=OBJ.pc(3) 'A remain stopped
muovi()
VectAssi(0)=4000 'X
VectAssi(1)=6000 'Y
VectAssi(2)=200 'Z transported
VectAssi(3)=OBJ.pc(3) 'A remain stopped
muovi()
VectAssi(0)=5000 'X
VectAssi(1)=2000 'Y
VectAssi(2)=300 'Z transported
VectAssi(3)=OBJ.pc(3) 'A remain stopped
muovi()

' ********************************************************
```

```
' Movement function waiting if the buffer is full
' ********************************************************
Function muovi() as Void
Dim test as Char
Label Move
test=Obj.lineto(vel,VectAssi())
if test=0
        goto Move
endif
EndFunction
```

12.4.4 ARCTO

Movement with CIRCULAR interpolation on the axes of the current WORKING PLANE. Two axes execute a CIRCULAR interpolation while the others are interpolated in LINEAR mode. As function LINETO, the property ***sglp*** defines the edge threshold for axis stopping. The direction of rotation is determined by the parameter **mode**.

Hardware *All*

Syntax

.ARCTO(Long Vel, Char mode, Long *PntAx, Long CX, Long CY) as char

Parameters

Vel	Velocity of interpolation as unit/sample
mode	Direction of rotation
	mode=2 CW interpolation
	mode=3 CCW interpolation
PntAx	Pointer to the array of the axis position as unit
Cx,CY	Coordinate X,Y (axis of the working plane) of the CENTER

Return value

Char	***0***	Command not written in the buffer (buffer full)
	1	Command written in the buffer
	-1	Radius error (dipends by ***sglr***)

Note

Arcto executes a CIRCULAR interpolation ON WORKING PLANE while the other axis are interpolated in LINEAR MODE.

Example (object name = OBJ)

Used variables:

```
VectAssi(4) long
Cx long
Cy long
Vel long
'*********************************************************
'Circular interpolation CW on X,Y Z and A
'to realize the programmed arc the axis X and Y must be in precise positions, for Example at 0,2000
'*********************************************************
vel=1000
VectAssi(4) long
VectAssi(0)=1000' final position X
VectAssi(1)=2000' final position Y
VectAssi(2)=5000' final position Z
VectAssi(3)=1000' final position A
Cx=500  'center X
Cy=500  'center Y
muovi()
Function muovi() as Void
Dim test as Char
```

```
Label Move
test=px_arcto(vel,2,VectAssi(), Cx, Cy)
if test = 0
        goto Move
endif
EndFunction
```

12.4.5 SETCMD

This function allows the synchronization of commands with the axis movement. In fact because of BUFFER OF AXIS MOVEMENT the interpolation functions don't wait the execution of the command but write it in the buffer. This implies the impossibility to command, for example, the digital output in a precise point of the path if axis don't stop in each position. This function enables the writing of a command value in the buffer when a interpolation function is called (***moveto, lineto, arcto***), it will be written in ***cmd*** at the instant the movement starts.

Hardware *All*

Syntax

.SETCMD(Long CMD)

Parameters

CMD Value of the command

Example

```
muovi()
OBJ.setcmd(10)
muovi()
OBJ.setcmd (20)

'Insert the following code in the TASK PLC
if OBJ.CMD=10
        …
endif
if OBJ.CMD=20
        …
endif
```

12.4.6 SETPIANO

Selects the current working plane on desired axis. By default the plane is set on the first two axis X, Y (ax1=0, ax2=1). Ax1 can not be equal to ax2.

Hardware *All*

Syntax

.SETPIANO(Char Ax1, Char Ax2)

Parameters

Ax1 Index of the first axis of the plane

Ax2 Index of the second axis of the plane

Note

The WORKING PLANE selects the axis for the CIRCULAR interpolation, for calculation of the edge 2D (***sglp***) and for calculation of the SPEED VECTOR in the function LINETO.

Example

```
Obj.setpiano(0,1)        'select the plane on axis X and Y
Obj.setpiano(1,2)        'select the plane on axis Y and Z
```

..1 STOP

Stops axis with the programmed deceleration (***acc***) waiting for the complete execution (axis stopped).

STOP is used to stop the axis before the TARGET point, programmed with MOVETO, LINETO or ARCTO, is reached. **The movement**

buffer will be emptied.

Hardware ***All***

Syntax

.STOP()

Notes

STOP, unlike FSTOP, waits the axis are stopped, for this **IT MUST NOT BE CALLED IN TASK PLC.**

12.4.7 FSTOP

Stops axis with the programmed deceleration (***acc***) without waiting for the complete execution (axis stopped).
FSTOP is used to stop the axis before the TARGET point, programmed with MOVETO, LINETO or ARCTO, is reached. **The movement buffer will be emptied**.

Hardware ***All***

Syntax

FSTOP()

Note

FSTOP, unlike STOP, doesn't wait the axis are stopped, for this **IT CAN BE CALLED IN TASK PLC.**

12.4.8 MOVE

Returns the state of the interpolation.
Hardware ***All***

Syntax

.MOVE() as char

Return value

char	**0**	No interpolation is running
	1	Interpolation is running

Note

MOVE retunrs 0 only the axis are stopped and the movement buffer is empty.
ATTENZIONE: MOVE tests only the DEMAND POSITION of AXIS.

Example

```
Muovi()          'start interpolation
while Obj.move()          'wait for complete execution
endif
```

12.4.9 PRESET

Preset the AXIS position without move them. Axis will assume the position as passed by parameters.
Hardware ***All***

Syntax

.PRESET(long *Pos)

Parameters

Pos Pointer to the array of the position value to preset

Note

Keep in mind these rules:

- **AXIS MUST BE STOPPED**
- **CHANGING INSTANTLY THE POSITION IT OCCURS A PARTICULAR SEQUENCE TO AVOID THE PHISICAL AXIS MOVES ROUGHLY**

For example WHEN USING THE CANOPEN AXIS IT NEEDS:
- **REMOVING THE CANOPEN FROM THE INTERPOLATION MODE**
- **PRESETTING THE CANOPEN AXIS BY METHOD .HOME**
- **PRESETTIN THE INTERPOLATOR WITH FUNCTION PRESET(pos())**
- **SETTING AGAIN THE CANOPEN AXIS IN INTERPOLATION MODE**

Example with the axis X as CanOpen (object name *AxisCan)*
Used variables:
PresetValue(3) as long

```
AxisCan.start=0                 ' remove the start condition
AxisCan.modo=0                  ' set the position mode (remove from interpolation mode)
AxisCan.home=1000               ' preset of axis at 1000
PresetValue (0)=1000            ' set the preset value in the position array for X
PresetValue (1)=OBJ.pc(1)       ' value to not modify the Y position
PresetValue (2)=OBJ.pc(2)       ' value to not modify the Z position
OBJ.PRESET(PresetValue ())      ' preset of the interpolator
AxisCan.modo=2                  ' set the Interpolation Mode
AxisCan.start=1                 ' start
```

In similar way the same problem can occur using the STEP/DIR axis. **Refer to the chapter of STEP/DIR channels for a correct preset of them**.

12.5 CANOPEN FUNCTIONS

This group of functions allow the management of CANOPEN line at application level. A lot of library OBJECTS use these functions to make it more simple but in some cases it is necessary using the primitive functions directly.

12.5.1 PXCO_SDODL

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.
Hardware ***All***

Syntax

PXCO_SDODL(char node, unsigned index,unsigned char subidx,long len,char *data) as char

Parameters

Node	Node ID of the SLAVE to whch 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.

WARNING: Cause the different allocation of bytes inside variables be careful to set the length corresponding to the variable type passed by pointer.

Example
Used variables:

```
value int
Ret char
value=100
Ret=pxco_sdodl(1,2000,0,2,value())          'node=1, index=2000, subidx=0,
                                                    'len=2 byte, value=100
if Ret<>0                                   'test if error occurs
        if Ret=2
                read_sdoac()'read eventual SDO ABORT CODE
                ...
        endif
        ...
endif
```

12.5.2 PXCO_SDOUL

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.

Hardware *All*

Syntax

PXCO_SDOUL(char node, unsigned index,unsigned char subidx,char *dati) as char

Parameters

Node	Node ID of the SLAVE to whch 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*** int the system variables _SYSTEM_SDOAC0 e _SYSTEM_SDOAC0 will be available the relative error code.

WARNING: Cause the different allocation of bytes inside variables be careful to use the variable passed by pointer of the type corresponding to the length of the data to be read.

Example

Used variables:

```
value int
Ret char
Ret=pxco_sdoul(1,2000,0,value())  'node=1, index=2000, subidx=0,
                                          'value=data read
if Ret<>0                                 'test if error occurs
        if Ret=2
                read_sdoac()'read eventual SDO ABORT CODE
                ...
        endif
        ...
endif
```

12.5.3 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 PXCO_SDODL or PXCO_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.

Hardware *All*

Syntax

READ_SDOAC()

..2 PXCO_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.
Hardware ***All***

Syntax

PXCO_SEND(int id, char Len,char *Data) 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

Example
Used variables:
value int
Ret char
value=100
Ret=pxco_send(0x201,2,value()) ***'Send a PDO (cob-id=0x201) with 2 byte***
if Ret<>0 ***'test if error occurs***
...
endif

12.5.4 PXCO_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.
Hardware ***All***

Syntax

PXCO_NMT(char state, char node) 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

Example
Used variables:
pxco_nmt(2,1) ***'Set in STOP the node 1***

12.5.5 READ_EMCY

Reads the last EMERGENCY OBJECT frame sent by a CAN OPEN node.
The emergency code is written in the system array _SYSTEM_EMCY(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.
Hardware ***All***

Syntax

READ_EMCY() as char

Return value

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

_SYSTEM_EMCY							
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 <u>if more EMERGENCY OBJECT are sended along a single task plc, only the last will be read</u>.
An EMERGENCY OBJECT non significa che effettivamente ci sia un nodo in emergenza. 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.

Example
Used variables:
Err Long
NodeErr Char

```
function Alarm() as void
        NodeErr=read_emcy()
        if NodeErr=0                        ' no error
                return
        endif
        err=(_SYSTEM_EMCY(7)&0xff)          ' Read 4 byte of Manufactured specific
        err=err<<8                          ' field masking eventual bit not
        err=err|(_SYSTEM_EMCY(6)&0xff)      ' interested
        err=err<<8
        err=err|(_SYSTEM_EMCY(5)&0xff)
        err=err<<8
        err=err|(_SYSTEM_EMCY(4)&0xff)
endfunction
```

12.6 DATA SAVING FUNCTIONS

All hardware are equipped with several type of memory usable for DATA SAVING. According to the type of memory (Fash, Fram, etc.) some rules are to be implemented.
For example a FLASH memory has a ***maximum number of writing, block erase, etc.***

12.6.1 IMS_WRITE

Writes in the internal FLASH at the address contained in ADDR, the data pointed by Punt for a total of NBYTE of data.
The FLASH memory is managed in BLOCKS of 256 bytes, for this it's recommended to write multiple of 256 bytes. That because also writing less than 256 bytes the entire BLOCK is erased, therefore to avoid the loss of data it needs at beginning to read all the

block, save the interested data and overwrite again all the block. The systems NGWARP or PEC70 have enough FLASH memory to be used without problems in blocks of 256 bytes also there is the need of less data.
Using the **NGM13,NGMEVO,NGQ,NGQx**, this function works on a FRAM memory which can be managed at single BYTE.
Hardware ***All***

Syntax

IMS_WRITE(char *Punt, long Addr, long Nbyte) 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

Example
Used variables:
Vett(10) long
Ims_Write(Vett(),0,40) ' write 40 bytes (10 long * 4) to ADDR 0

WARNING: In this case the entire block of 256 byte is written if we are working with FLASH (NGWARP).

12.6.2 IMS_READ

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

Syntax

IMS_READ(char *Punt, long Addr, long Nbyte) 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

Example
Used variables:
Vett(10) long
Ims_Read(Vett(),0,40) ' read 40 bytes (10 Long) from Addr 0

12.7 ETHERNET FUNCTIONS

Systems equipped with ETHERNET manage AUTOMATICALLY the STACK TCP/IP. To work with protocols at upper level than TCP/IP it must be written some source code in the application. For example to process the MODBUS-TCP protocol there is a specific object in library which uses the functions of this group. In the same way it's possible to create customized protocols.

..3 SET_IP

Sets the parameters of TCP/IP protocol.
Hardware ***NGWARP,NGMEVO***

Syntax

SET_IP(ip as *char, sm as *char, gw as *char)

Parameters

ip	IP address of the device

sm subnet mask
gw gateway

Example

```
Set_ip("10.0.0.15","255.255.255.0",0)      'IP = 10.0.0.15
                                           'SUBNET = 255.255.255.0
                                           'GATEWAY = nothing
```

WARNING: This function must be called in the INIT section of the MAIN or PLC TASK.

12.7.1 PXETH_ADD_PROT

Adds a custom protocol to a specific port of TCP/IP. A custom function to process the new protocol must be written and its pointer must be pass to this function.

Hardware ***NGWARP,NGMEVO***

Syntax

PXETH_ADD_PROT(port as long, fun as delegate)

Parameters

port TCP port on which the new protocol is added
fun Pointer to the custom process function

Example

Used variables:
fun delegate

Init section of main:

```
Set_ip("10,0,0,15",0,0)      'set IP = 10,0,0,15
fun=my_protocol
pxeth_add_prot(502,fun) 'Add the protocol my_protocol on port 502

'protocol process function
function my_protocol(len as long, buftx as *char) as long
...
endfunction
```

12.7.2 PROTOCOL PROCESS FUNCTION

This function isn't defined by system but it must be written in the application. The system will call this function, by the pointer passed with ***pxeth_add_prot,*** each time a data packet is received from the port associated to this protocol. To read the received data the function **pxeth_rx** have to be call while to send the response data they must be written in the transmit buffer (buftx) and return from the function the number of bytes we want to send.

Hardware ***NGWARP,NGMEVO***

Syntax

PROCESS_MY_PROTOCOL(len as long, buftx as *char) as long

Parameters

len Length of data packet received

buftx Pointer to the transmit buffer

Return value

long Number of bytes to be send

Example

Used variables:

bufrx(100) char

```
'protocol process function
function my_protocol(len as long, buftx as *char) as long
dim i as int

for i=0 to i<len            'Read all received data
        bufrx(i)=pxeth_rx()
next i
...                         'Process the data
buftx(0)=12
buftx(1)=34
my_protocol=2               '2 will be sent as response
endfunction
```

12.7.3 PXETH_RX

Read a single byte from the TCP/IP receive buffer. It is called by the protocol process function to read the received data.

Hardware ***NGWARP,NGMEVO***

Syntax

PXETH_RX() as char

Return value

Char Data read from the receive buffer

13 COMPONENT FOR FRAMEWORK

VTBII compiler can create a DLL COMPONENT MODEL which can be imported in .NET (dot net) projects. That allows the full control of hardware resource directly by a PC: READ/WRITE VARIABLES, CALL FUNCTION IN REMOTE PROCEDURE CALL.
For details refer to the **NG Framework manual**.

13.1 Enabling the creation of the COMPONENT NGFRAMEWORK

The generation for DLL for framework, is enable by VTBII Option
(See **.NET FrameWork** *)*

13.2 Exporting VARIABLES

We can export the desired variable to FRAMEWORK and then, on PC, write or read them as normal variables of the project.

In the example the variables will be contained in Generic.VAR1EXP and it can be read or written on the PC project as a common variable.
We remember the time of execute the READ or WRITE operation depends by the enabled LINK: serial port RS232 or ETHERNET. Obviously the second one will be more fast.
Only the INTERNAL VARIABLES can be exported, also if the it is refer to a structure.

13.3 Exporting FUNCTIONS

In a similar way as for variables it can be exported also functions.
That must be declared with a specific POSTFIX :

function FunctionName(...Parameters...) as Type **$_EXPORT_$ CLASS**
. . .
endfunction

$_EXPORT_$ **Keyword to enable function exporting**
CLASSE **Name of the exporting class where the function will be found**

Example:
function MyFunction(Val1 As Long,Val2 As Long) as Long **$_EXPORT_$ FunzSistem**
. . .
endfunction

14 APPLICATION DEBUG

The DEBUG utility allows to control, both read and write, of all the application variables, to insert BREAK POINT and to execute the code STEP by STEP. That makes more simple the development of the application. The application DEBUG can be execute by RS232 port as well as ETHERNET.
When the serial port is used, the PC must be connected to the first port of the target hardware (**SER-1/ PROG)**.
WARNING: If application uses the first serial port, (ex. MODBUS, etc.) DEBUG will not work.

For Start Debug See Tools Bar

14.1 Button bar

Add a variable to the WATCH window.
It allow to insert a variable which will be update in REAL time and it will be also written.

Writing in the field Nome VARIABILE the alphabetical list of the variables of the project will appear making the searching very simple. Variables can be added also in the following ways:

Drag&Drop. Select the desired variable in the code window and drag it in the WATCH window.

```
if AsseX_flagb=1 && AsseX.move=0
    AsseX_flagb=0
    gosub AsseX_OnEndMove
endif
```

Right button. Click with the right button on the selected variable and then **Send to Debug**.

```
if AsseX_flagb=1 && AsseX.move=0
    AsseX_flagb=0
    gosub AsseX_OnEndMove
endif
```

Invia a Debug

Vai a Definizione

Pagina
It selects the page of the VARIABLE (if it is a local variable of a page), PAGINA 0 refer to the GLOBAL variables.

Contesto
If the watching VARIABLE is local of a FUNCTION (defined with **dim**) we can select the contest (function) of this variable.
These types of variables are visible only if a BREAK POINT in the relative contest is reached.

Remove the selected variable.
The selected variable will be removed from the WATCH window.

Remove all variables from the WATCH window.

Remove all Break-Points in the project.

Information about DEBUG.NET
With this button we can display some informations about DEBUG.NET and the target hardware. Also it is possible to update the FIRMWARE of the target. (See section Firmware Update).

Stop array reading.
When arrays of BIG DIMENSION are read can happen a TIME OUT of the system, with this button we can stop the read.

Reset
It simulates a RESET of the HARDWARE.
WARNING: The application will be restarted.

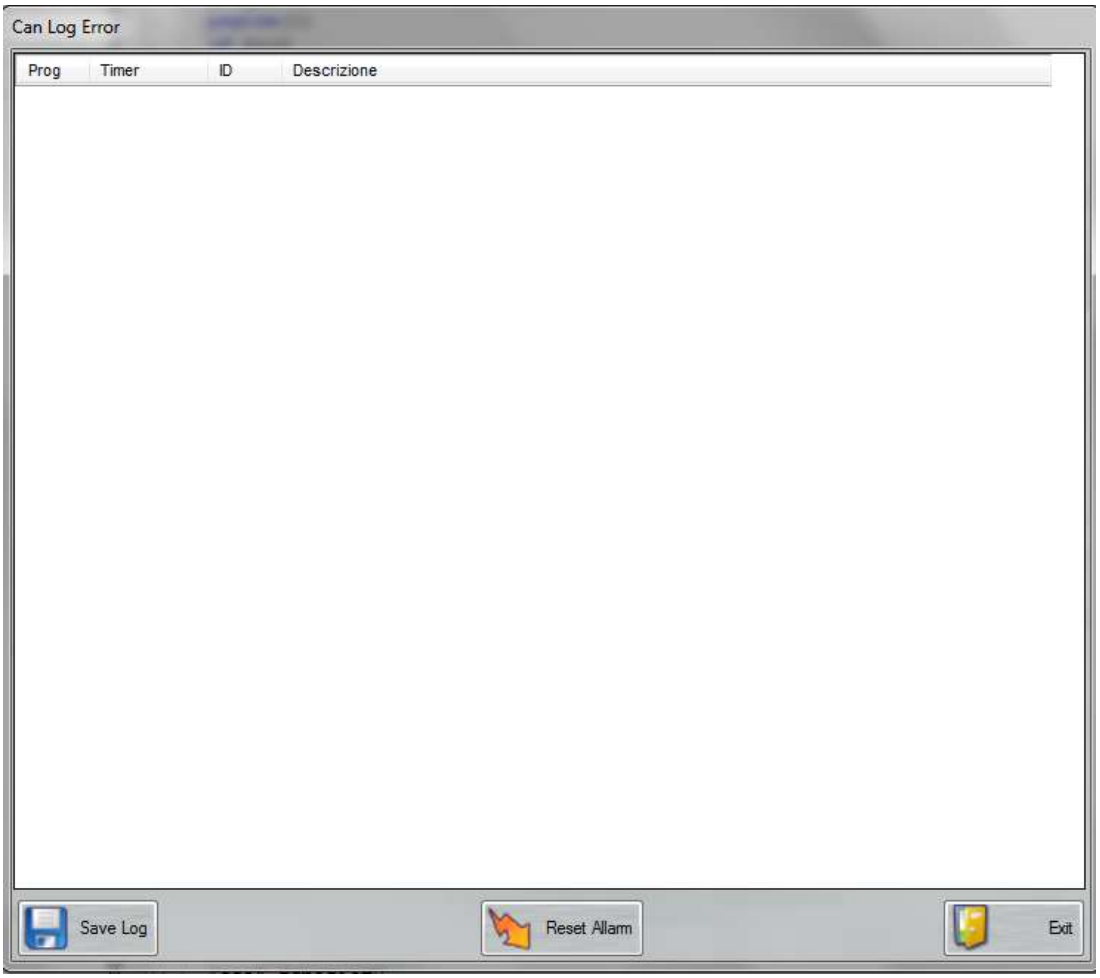
Save the list of variables on file
It is possible to create a file with the list of the variables in the WATCH windows to reload it afterword.

Load a variables list file
It allow to reload a list of variables previously saved.
The content of the variables WILL NOT BE INIZIALIZED.

Load a variables list file with value
It allow to reload a list of variables previously saved.
The content of the variables WILL BE INIZIALIZED with the saved value.

Load the last variables list
DEBUG.NET always saves the list when it is closed. With this button we can reload the last variables.

Display the LOG of HARDWARE ERRORS
All run-time errors are saved in this list. It is very useful particularly with CanOpen applications to test if in the CANBUS net there are some errors or it works correctly.



Errors are sampled by directly by the target hardware in REAL TIME and they are showed in TEMPORAL order. It is also possible to save the logging list in a file to analyze them afterword.

Scope
Enable the digital scope (see relative section)

DEBUG.NET options
It allows to set some DEBUG options.

Block Read Delay (Ms)
If this option is greater than ZERO a delay is added after the read of a block. If DEBUG uses the serial port RS232 IT ISN'T NECESSARY.
It can be useful in ETHERNET because the high speed of the protocol could create some problem to the VTB application (slowdowns).
We recommend to set the delay, when using ETHERNET to debug the application, with a value of at least one Ms.

HEXADECIMAL/DECIMAL display
If activated the numeric value of the variables will be displayed in HEXADECIMAL format.

ASCII display
If activated, the ASCII character corresponding to the value of the variable will be displayed (it is useful for array of alphanumeric STRINGS).

It shows the elapsed time (in Milliseconds) of the TASK PLC and the relative percentage of CPU using. If the system read a value near the CRITICAL one it will be signal by RED BLINKS af the value.

Run after BreakPoint (or F5 key)
When a Break-Point is reached, it allow to resume the normal running of the program.

Execute Intruction/Routine (or F10 key)
When a Break-Point is reached, with this button it is possible to execute a single line of source code.
Eventual functions will be execute completely **without enter inside them.**

Execute Intruction (or F11 key)
When a Break-Point is reached, with this button it is possible to execute a single line of source code.
If a function is encountered, program will stop inside it.

Find text
Find a text in the source code windows.

Task Plc

Display the content of TASK PLC
WARNING: in TASK PLC it isn't possible to set a Break-Point.

14.2 Writing of a variable

It is possible to change the value of all the variables in the WATCH list. Double click on the value and then write the desired value.

Nome	Valore	Pag.	Contesto
PROVA	7458171	0	

If the variable is a type BIT the double click switches from TRUE to FALSE and vice versa.

14.3 Insert/Remove a Break-Point

The insert of a Break-Point allows to break the program in a specified point. When a Break-Point is reached it is possible to execute STEP by STEP the program checking the correctness.
WARNING: Break-Points can not be inserted in the hardware NGM13.

By Select File select the desired page of code.
Click with the left button of the mouse on the left of the source code window.

```
19  inc prova
20  if _on_paint=2
21      _on_paint=0
22  endif
23  return
```

Click here

When the program passes from that line, the bar, from BROWN, will turn YELLOW and the execution will be BROKEN. At this point it will be possible re-run the program with **Run after BreakPoint (F5)** or execute it Step by Step**.**

```
20  if _on_paint=2
21      _on_paint=0
22  endif
23  return
```

To remove a Break-Point click again on the Break-Point

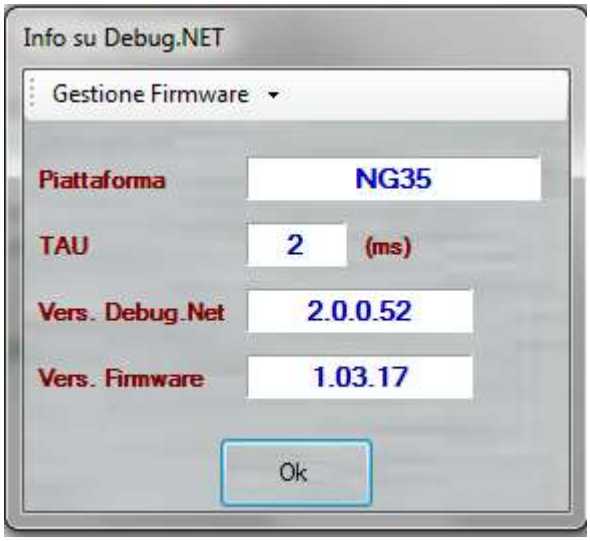
WARNING: When a Break-Point is reached and the program is stopped, the TASK PLC continues to run. Anyway breaking the program in CRITICAL points we can create unsafe situation operating on machine. BE CAREFUL !

14.4 Firmware update

With DEBUG application it is possible to update the FIRMWARE of the hardware in use.

WARNING: FIRMWARE update can be executed only by serial port RS232.

With the INFO button this window is showed:



From Menu Gestione Firmware we can chose between two options:

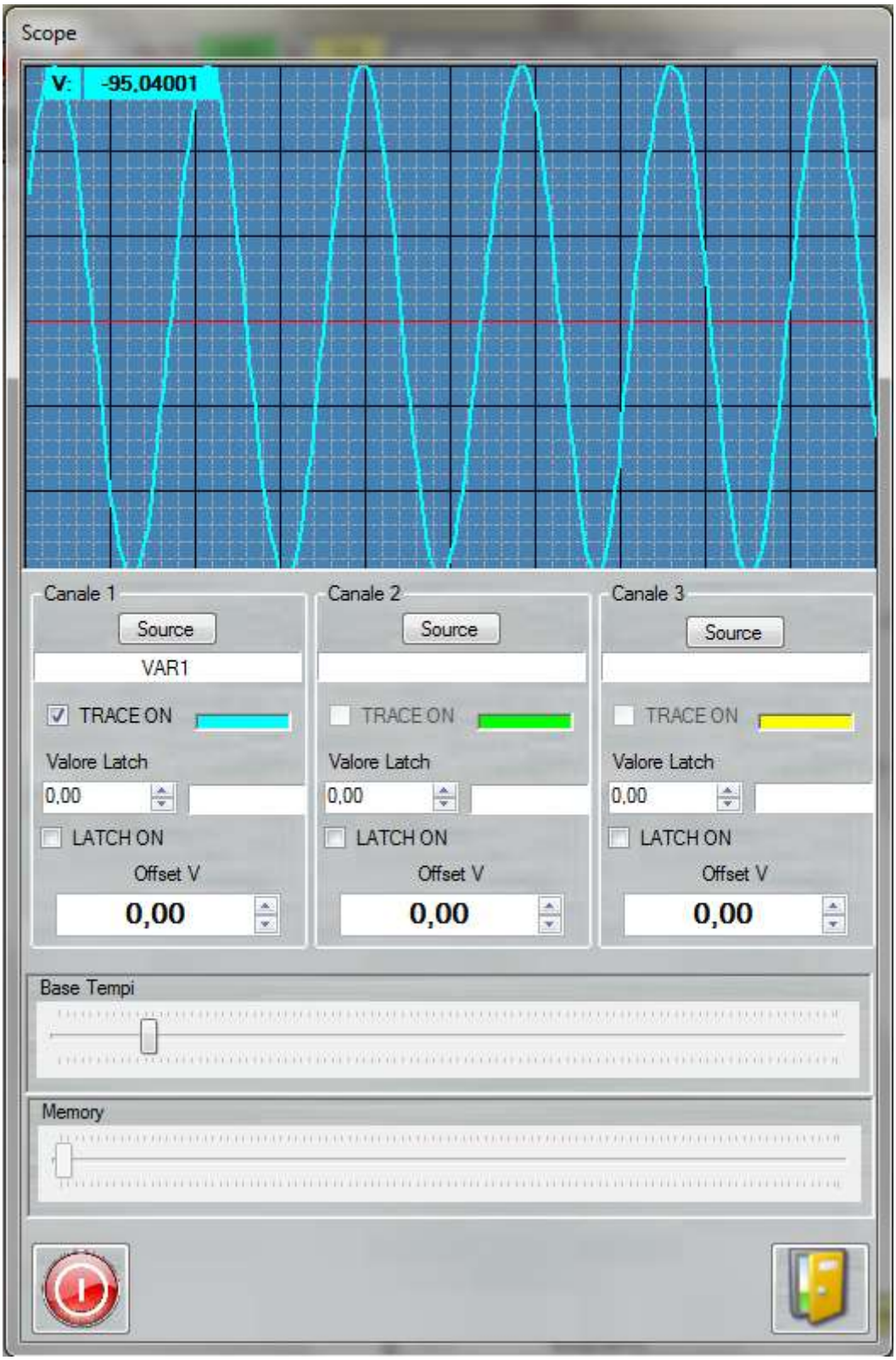
Update from Server
In this case an INTERNET connection is necessary. The application checks if on SERVER PROMAX there is a newer version of the FIRMWARE proposing the updating.

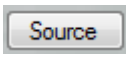
Update from file
It allows to update the hardware FIRMWARE with a file .SREC.
WARNING: Updating from file, no control of the firmware revision and compatibility with the hardware is made.

WARNING: During the phase of updating the application are stopped but it WILL NOT BE LOST.

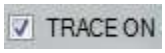
14.5 Digital Scope

DEBUG.NET provides a SCOPE application to further support of debugging. DIGITAL SCOPE is able to monitor the variables in the **WATCH** window.
The scope can display up to 3 CHANNEL.





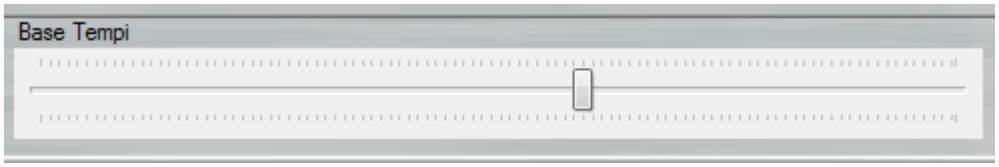
Selects the variable to connect to a channel.
The variable must be in the WATCH window.



Enables or disables the TRACK of a channel.

Sets an OFFSET on the TRACK.

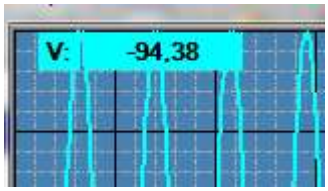
Enabling LATCH, when the variable overcomes the Latch value, the TRACK will be FROZEN.



Set the BASE-TIME for all the tracks.

When scope is in OFF state, it aloows to scroll the track in the sampled memory.

Scope ON/OFF.

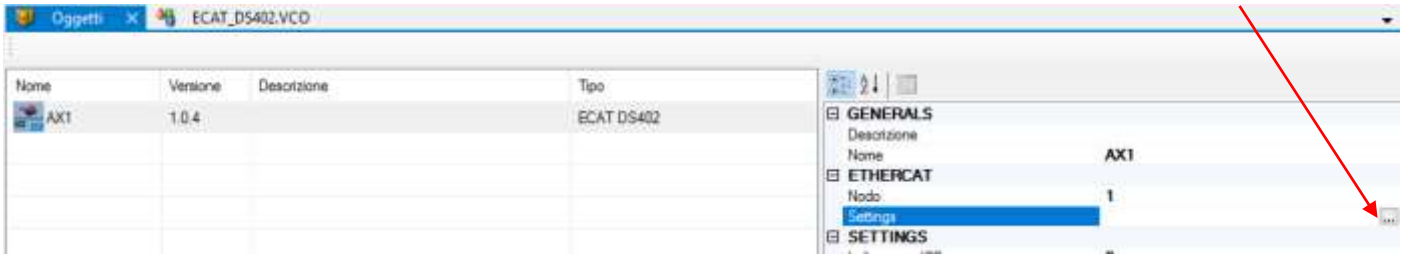


Positioning the mouse on a point of the track, the value of the variable will be showed.

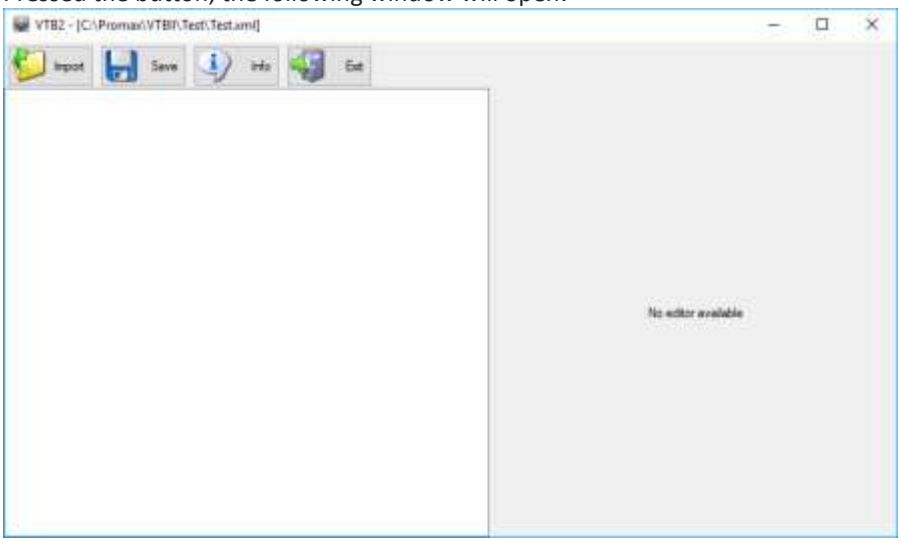
15 ETHERCAT CONFIGURATION

This chapter explains the steps to configure a network of EtherCAT devices.

To access to the EtherCAT configuration, select the inserted object for Etherca device, and open the section ***ETHERCAT->Settings***:



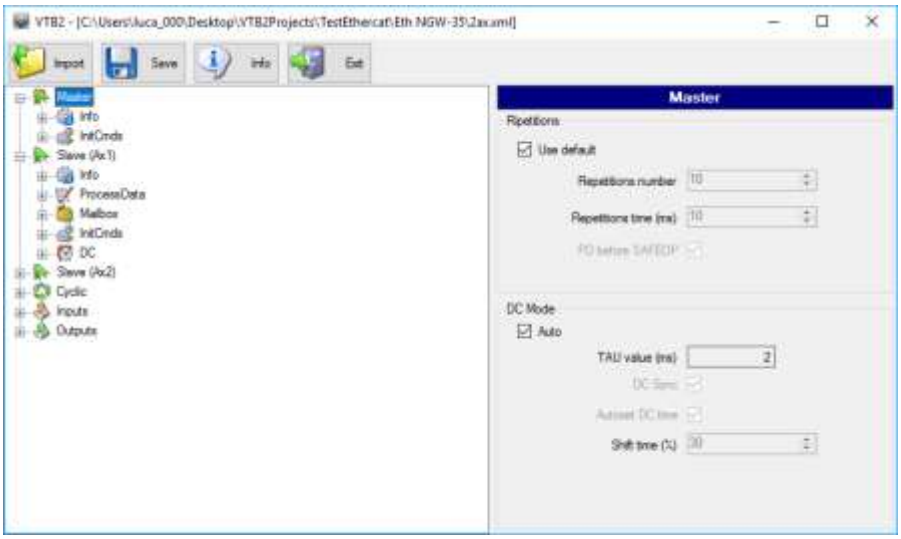
Pressed the button, the following window will open:



To import one of the EtherCAT network configuration, you have to follow the following steps:

- Create the configuration of the network with an EtherCAT configuration software(We normally use Twincat)
- Import the created XML file with **Import** button

At this point, the window will look like, and display all data imported:

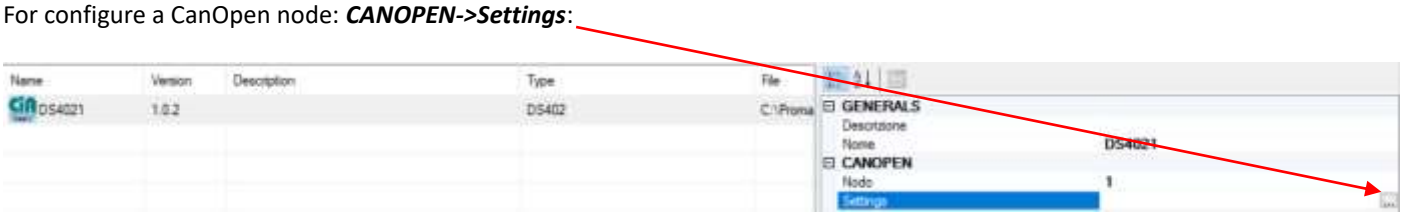


Now you can change certain data such as the names of variables, the DC sync mode, the shift time to the data of InitCmd the Master. Make any changes you can save the configuration to be used in the project.

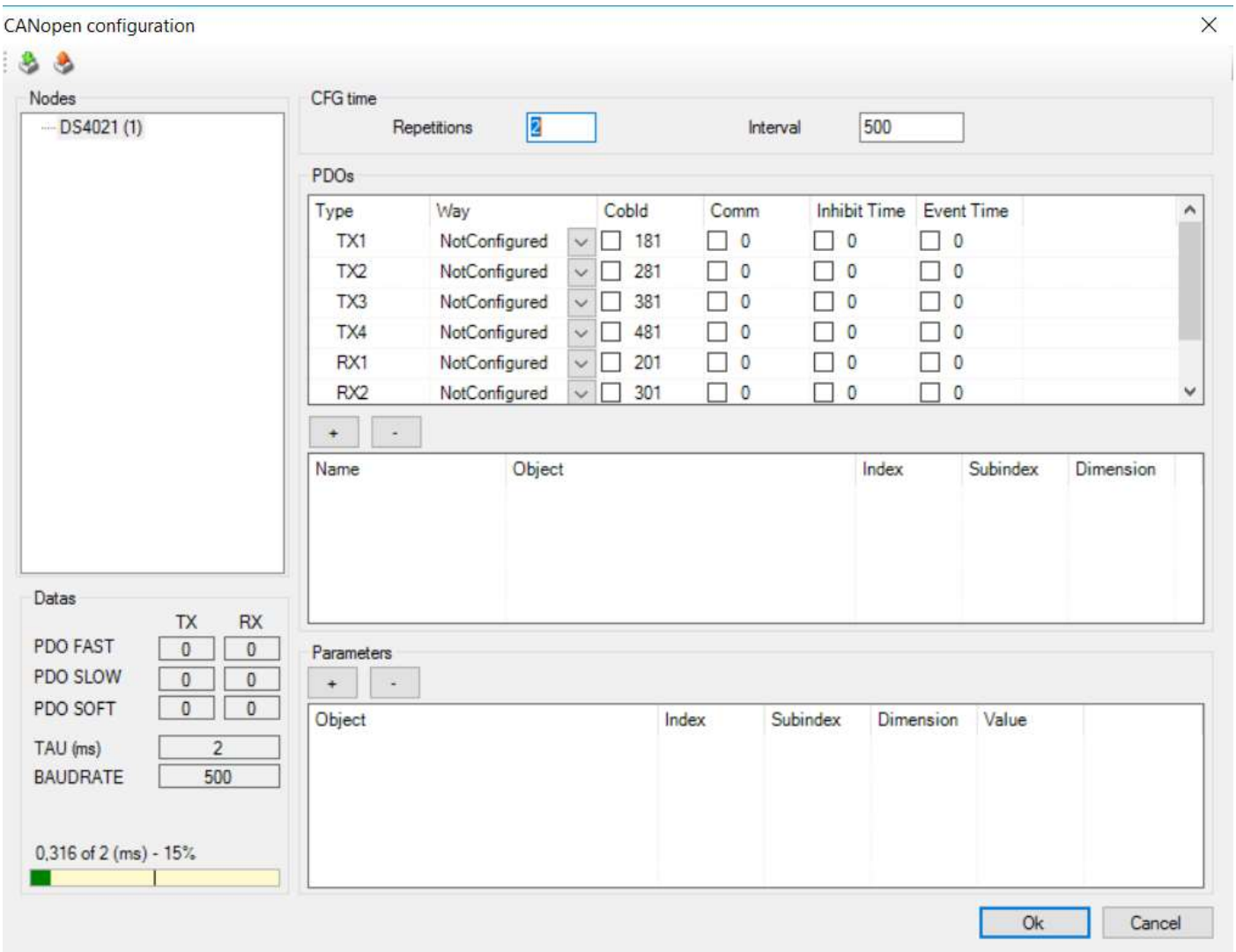
16 CANOpen NODE CONFIGURATION

This chapter not explain the CanOpen informations, so, SDO, PDO, TX,RX etc. is assuming that these are known

For configure a CanOpen node: ***CANOPEN->Settings***:



Following will be shown the window configuration:



16.1 Import an Existing Configuration

An existing configuration, contains all PDO configured.

Press button and choose the configuration file previous exported

16.2 Export the Current Configuration

When all PDO are configured, is possible export the configuration

Press button and save the file

16.3 Cfg Time

In normal situations, when the machine starts, it's usual to have the master CNC that makes initialization functions in a very short time, like the NGWARP for ex., while one or more slaves starts in a longer time, like drives.
Promax system makes the net configuration as the first initialization operation, then in this case could be possible that a device will be not configured, cause it's not ready to accept master instructions. Therefore the device will not be able to exchange PDO informations, like interpolated target quotas for ex.
How can we avoid this situation?
With the Configuration time-out. Using this strategy, when the master try to set the slave and the slave doesn't answer, it will try once again after a little while, then can try again and again...
How many times the master must try to reach the slave and how many time have to wait from one and another try?
These are the data that are closed inside the round parenthesis: making a double-click, will open a form, where can be selected in the right way.

16.3.1 Repetitions

Repeat Number, are the times that the master will try to reach the slave

16.3.2 Interval

Time for repeat, expressed in ms, is the time between two repetitions.
In the example showed, the master will try to reach the slave (node 3) 5 times, wait 1000ms (1sec) every time. It means that the slave must have a start time, less then 5 sec.

16.4 PDOs

PDO TX and RX configuration for the node selected

16.4.1 Type

Type of PDO RX or TX

16.4.2 Way

Enable or Disable PDO
Mode

Manual
Fast
Slow
Soft
Not Configured
Disabled

16.4.3 COBID

PDO CoBId. Can be changed for include more PDOs

16.4.4 Inhibit Time

Not Used

16.4.5 Event Time

Not Used

16.4.6 Declaration VTBII variable for PDO

When the PDO Type is selected, can be the VTBII variable refer to PDO.

 Add a new VTBII variable for the PDO

Name Variable name visible on VTBII code
Object INDEX and SUBINDEX of PDO.
All standard CanOpen types are already defined. The CUSTOM Type allows to set manually INDEX, SUBINDEX and DIMENSION

Name	Object	Index	Subindex	Dimension
AxisPosition	[6064.0] - Position actual value in user units	6064	0	4

In this example, the variable AxisPosition contains the Node Axis value position

If AxisPosition >=10000 ***' Read Axis Position***

……

Endif

16.5 Parameters

In this section, are defined all node parameters.

16.5.1 Add a Parameter

Add a new Parameter

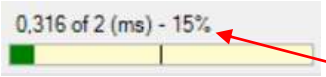
Select from list or CUSTOM for insert INDEX , SUB INDEX and DIMENSION
VALUE Defines the parameter value during StartUp

Object	Index	Subindex	Dimension	Value
[6040.0] - Controlword	6040	0	2	3

In the above example the ControlWord will be set to value 3 when the node will be configured.

16.6 Datas

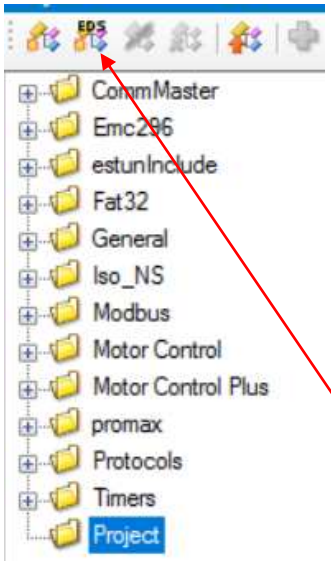
All Nodes configuration summary.
Is very important, check the Bandwith used compared to that available.



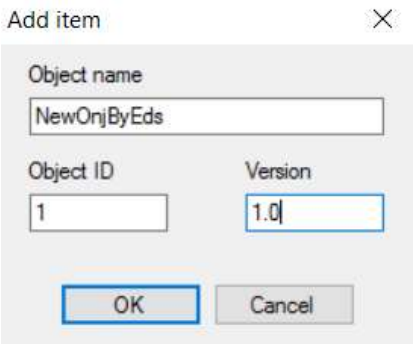
The Bandwith **MUST NOT EXCEED** over 90% of Bandwith available

17 IMPORT AN EDS FILE CANOpen

If the node is not present in the Objects list is possible to import an EDS file and create e new OBJECT.



1) Open the Objects TAB
2) Select the Project Folder
3) Import the EDS file from button "**Import EDS**"

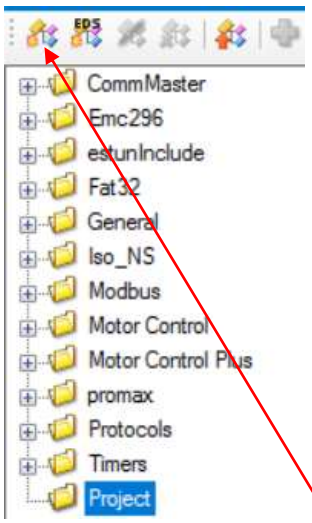


Object Name → Insert the new Object Name
Object ID → Set to 1
Version → Insert the Version

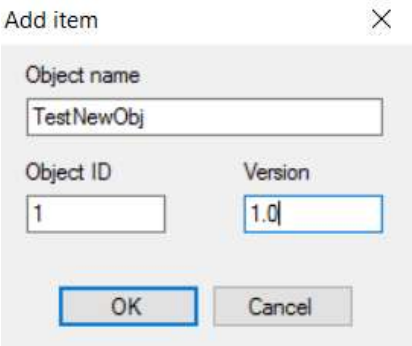
In the folder Project will be inserted the new Object. After that is possible import it like to others objects
The new Object is saved in the current project folder.

18 MAKING A CUSTOM VTBII OBJECT

VTBII allows to making a CUSTOM object for use in more projects.



1) Open the Objects TAB
2) Select the Project Folder
3) Press button "**New Objeact**"



Object Name → Insert the new Object Name
Object ID → Set to 1
Version → Insert the Version

18.1 New Object Properties

The new Object must have some propertiesi:

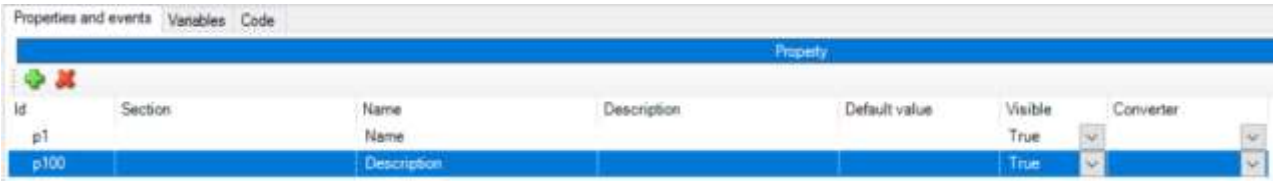
Variables Used
VTBII Code
Events

Select the new Object and press button "***Modify***"

Following is possble to insert Properties, Code and events for the new Object

18.1.1 Properties and Events

In the table Properties and Events are defined all Properties and Events for the new Object



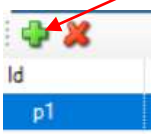
The Properties are the values that are inserted during the Application development (not in RunTime)
In the below example, **BitOFF**, **BitON** and **Enable** are Properties

The Properties can be grouped in the Categories .
When the new Object is created, two default properties are already defined:
P1 and **P100** where:

P1 → Object Name
P100 → Object Decription

Add a Property

Press Button "**Add**"



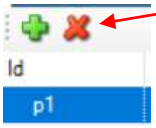
Insert the Fields:

ID → Property Name available in VTBII Code. It returns the Value field
For use in VTBII code is necessary the prefix **"?"** and Postfix **"?"**
Ex: property name P2 in VTBII code is::
VarProp=?P2?

Section → Property Group. If is not insert the Group is selected to Default
Name → Property Name Visible in the browser
Description → Short description
Default Value → Return Value (normally it is a numerical value)
Visible → **True** Visible in the Browser
False Invisible in the Browser
Converter → **None** - In the Browser only the value can be inserted
Variable - Is showed the button for a VARIABLE SELECTION
Function - Is showed the button for a FUNCTION SELECTION

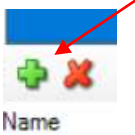
Delete a Property

Select the Property an press button "**Remove**"



Add an Events

Press Button "**Add**" in the Event Table



Delete an Events

Press Button "**Remove**" in the Event Table

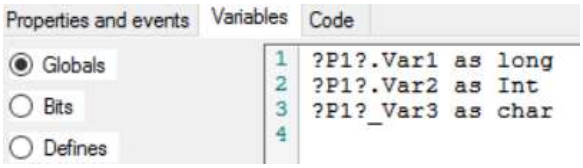
18.1.2 Variables

In the table Variables are defined all Variables used in the Object
The variables can be:

Globals → VTBII Variables
Bits → BIT Variables
Define → Define

Add a Global Variable

Select **GLOBAL** and insert the variable in the following format:
?P1?.VarName As Type
Where **P**1 is the Object Name Property
If is used **".**", the variable is shown in the Intellisense Help.
If is used **"_"**, the variable is not shown in the Intellisense Help.

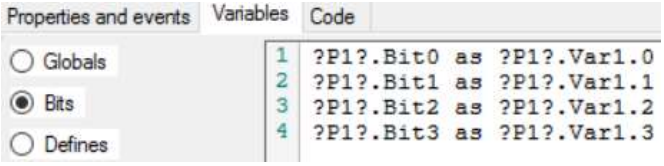


The variables are available the SOURCE code of OBJECT with the defined name **?P1?.Var1, ?P1?.Var2** etc.
In the VTBII application the variables are available with **ObjectName.Var1**, **ObjectName.Var2**, **ObjectName _Var3** etc.

WARNING
Is possible declare a variable without prefix ?P1?
<u>THIS IS AN ERROR. NOT USE</u>

Add a BIT Variable

Select **BIT** and insert the variable in the following format:
?P1?.NomeBIT As VAROBJ.BitNumber
Where **P**1 is the Object Name Property
If is used **".**", the variable is shown in the Intellisense Help.
If is used **"_"**, the variable is not shown in the Intellisense Help.
VAROBJ is a **GLOABL** variable.
BitNumber is the Gloab Variable bit number

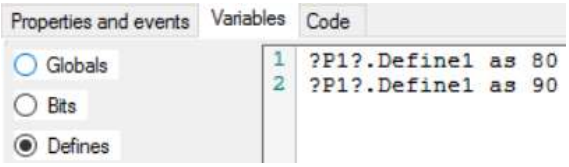


The BIT variables are available the SOURCE code of OBJECT with the defined name **?P1?.Bit0, ?P1?.Bit1** etc.
In the VTBII application the BIT variables are available with **ObjectName.Bit0**, **ObjectName.Bit1** etc.

WARNING
Is possible declare a BIT variable without prefix ?P1?
THIS IS AN ERROR. NOT USE

Add a DEFINE

Select **DEFINE** and insert the variable in the following format:
?P1?.NomeDEFINE As Value
Where **P**1 is the Object Name Property
If is used **".**", the variable is shown in the Intellisense Help.
If is used **"_"**, the variable is not shown in the Intellisense Help.
Value is the return value



Le DEFINE sono disponibili nel codice **OGGETTO** con il nome definito (?P1?.Define1, ?P1?. Define2 ecc.)
Mentre sono visibili nel codice di VTB con NomeOggetto. Define1, NomeOggetto. Define2
The DEFINE are available the SOURCE code of OBJECT with the defined name **?P1?.Define1, ?P1?. Define1** etc.
In the VTBII application the DEFINE are available with **ObjectName.Define1**, **ObjectName.Define2** etc.

WARNING
Is possible declare a DEFINEe without prefix ?P1?
THIS IS AN ERROR. NOT USE

18.1.3 Object Code

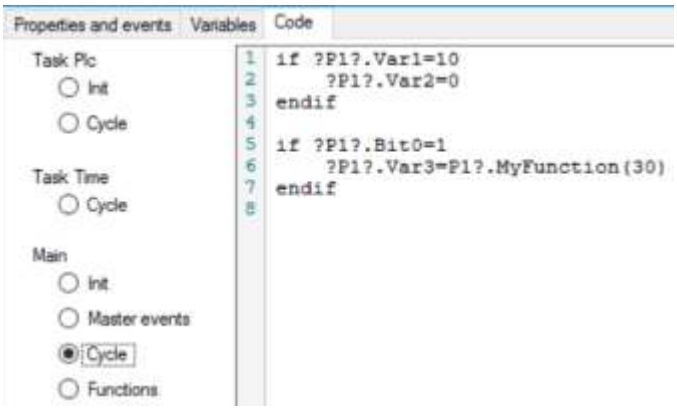
This section contains the Object VTBII code

The Code can be inserted in the following TASK:

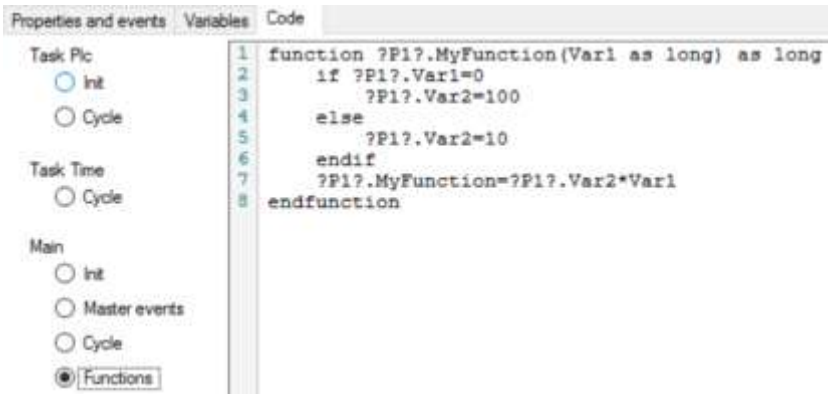
TASK PLC → **Init** - Init Task PLC
Cycle - Cycle Task Plc
TASK TIME → **Cycle** - Cycle Task Time
TASK MAIN → **Init** - Init Task Main
Master Events - Events Task Main (Events Manager)
Cycle - Cycle Task Main
Functions - Global function Declaration

Add Code

Select the desire TASK
Remember the PREFIX and POSTFIX for variables



Function

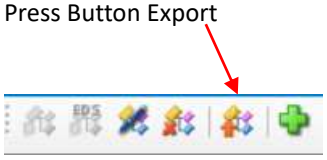


19 EXPORT NEW OBJECT IN THE CUSTOM LIBRARY

The Objects generated by **EDS** or from **New Object**, are inserted in the Current Project Folder.
Is possible to Export these Objects in the **VTBII CUSTOM LIBRARY**
This allows to use the new Objects in other projects.

1) Select the Object to Export in Custom Library 

2) Press Button Export

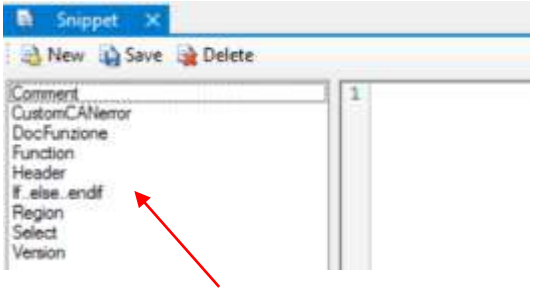


20 SNIPPET

The “Snippets” are the Code Fragments that can be used during the application development
This allows to CUSTOMIZE VTBII for a faster code writing.

20.1 New Snippet

From Menù ***Tools →Snippet Management***

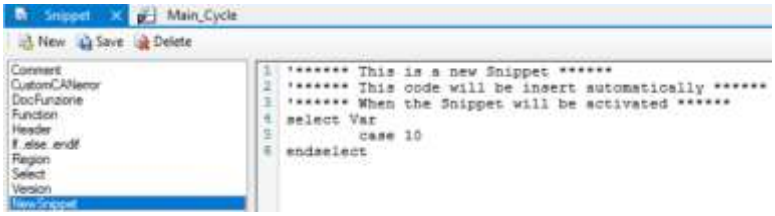


This List contains the Snippets already created (for modify, select the Sniipet and modify the Code)

New → New Snippet

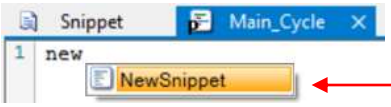
Insert Snippet Name
Insert Snippet Decsription

After, insert the Code that will recalled when the Snippet will be activated

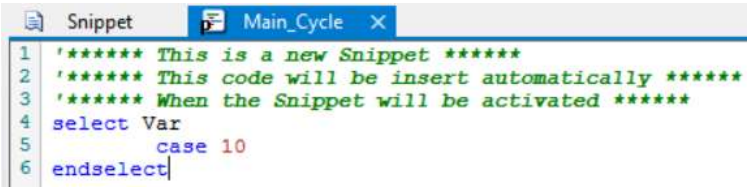


Save → Save the Snippet
Delete → Delete the selected Snippet

For recall the Snippet, during the code insertion, insert the Snippet Name.
The intellisense will show the Snippet Icon, and with Double Click on the Icon the Snippet will be inserted



Double Click



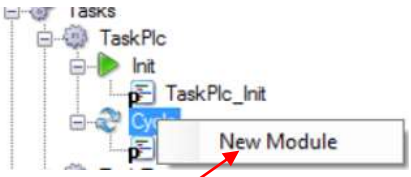
21 LADDER

VTBII allows to combine LADDER FUNCTIONS with BASIC MOTION
The Ladder language, uses a different logic respect to Basic Motion Language. The Ladder language is more simple to I/O managing.
VTBII is not based its programming structure on Ladder language, therefore it is not developed as the PLC.

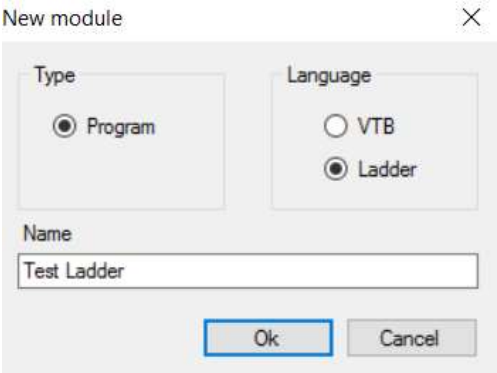
21.1 Add a Ladder Module

A Ladder Module can be added in Task **Main_Cycle, Time_Cycle, Plc_Cycle**.
Therefore, it is executed about Task priority.

Select the Task with Right Click:

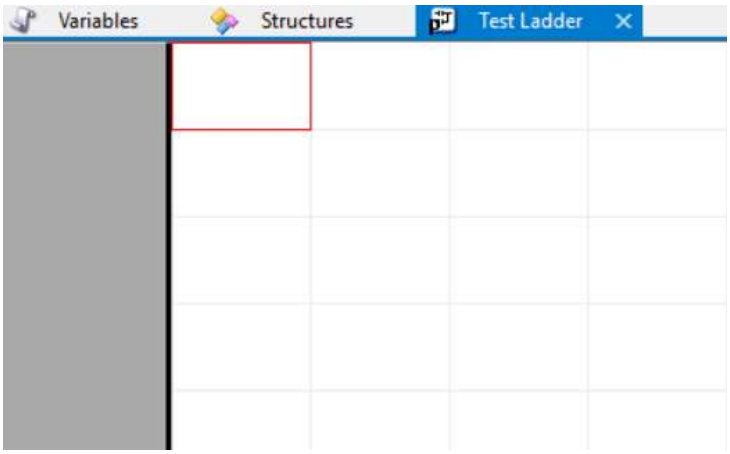


Select **New Module**.
From Menù **Project→New Module**



Select Ladder
Insert the Module Name

Following, the empty Ladder Module will be shown



21.2 Ladder Functions

Following is explained the Ladder Functions in VTBII

21.2.1 Insert a New NetWork

Insert a New NetWork in the Current Ladder Module

A minimum NetWork contains an Input and an Output (coil)

21.2.2 Remove a NetWork

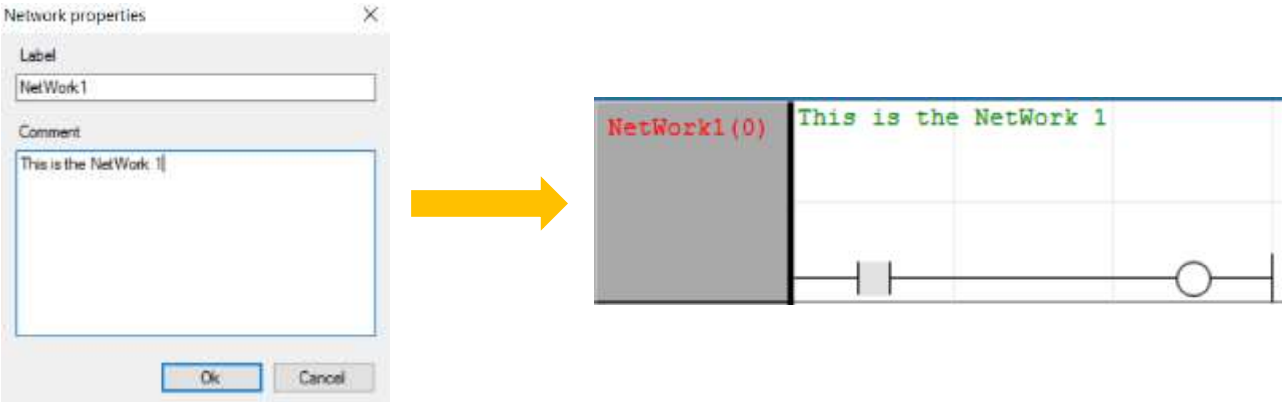
Remove a selected NetWork

21.2.3 Move a NetWork

Move **Up-Down-Start-End** the selected NetWork

21.2.4 Insert the NetWork Properties

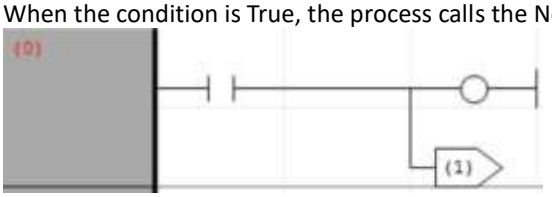
Insert the properties of selected NetWork (Label and Comment)



21.2.5 Insert a Jump in the NetWork

Insert a Jump.
The JUMP is an Output element and it is inserted to NetWork End

For insert the Jump parameters, double Click on the Jump Label and after insert the NetWork Number for the Jump
When the condition is True, the process calls the NetWork number



21.2.6 Insert Element Properties

Allows to insert the properties for the selected element
(see the single object)

21.2.7 Add a Contact in serires

Add a contact **AFTER-BEFORE** of the element selected:
Properties:
Name → INPUT Name. This is a variable can be used also in the VTBII Basic Motion Code
The INPUT can be chosen from the VTBII variables with the Button ...
This allows to insert the Digital Inputs declared in the Project
Normal → Normale
Negate → Negate

21.2.8 Add a Contact in Parall

Add a contact **AFTER-BEFORE** of the element selected:
Properties:
Name → INPUT Name. This is a variable can be used also in the VTBII Basic Motion Code
The INPUT can be chosen from the VTBII variables with the Button ...
This allows to insert the Digital Inputs declared in the Project
Normal → Normale
Negate → Negate

21.2.9 Add an Output (Coil)

Add an Output (Coil) to selected Network
The Output is added in parallel to Default Output
Proprietà:
Properties:
Name → INPUT Name. This is a variable can be used also in the VTBII Basic Motion Code
The INPUT can be chosen from the VTBII variables with the Button ...
This allows to insert the Digital Inputs declared in the Project
Normal → Normale
Negate → Negate
Set → Rising Edge
Reset → Falling Edge

21.2.10 Remove an Element

Select the element, and press key **DEL** from keyboard

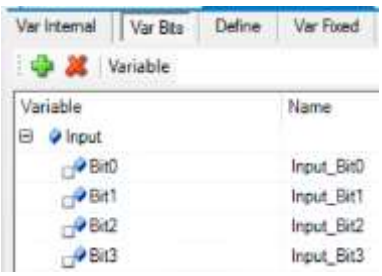
21.3 Debug Ladder Application

For Debugging the Ladder, open the Ladder Module and press Button
The contact, will shown OPEN or CLOSED with a different color

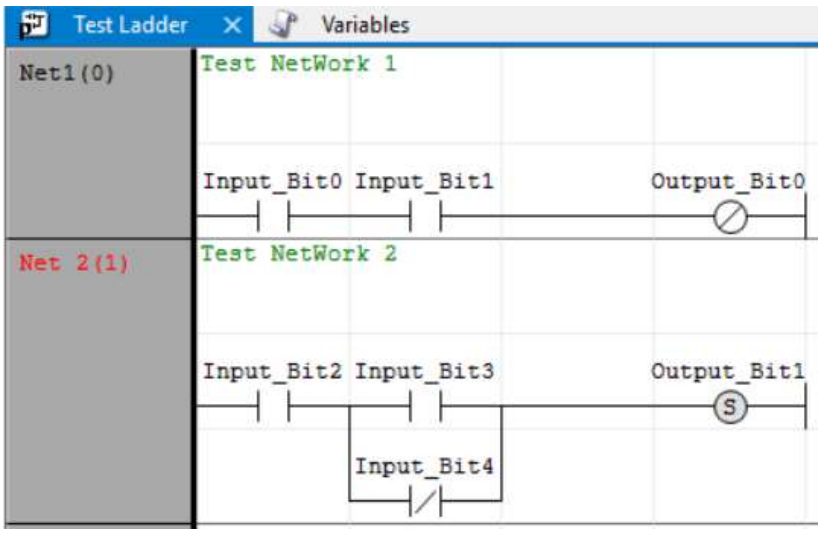
21.4 Ladder NetWork Example

Variables declared on Basic Motion:

BIT declared on the variables:



Ladder NetWork



22 GENERAL SETTINGS

Allows to customize the IDE

22.1 Option

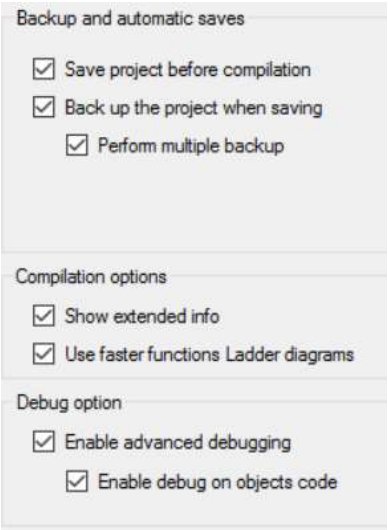
From Menù **Tools→Option**

22.1.1 General Option

Language setting

22.1.2 Project

Project Options



Save Project Before Compilation
Save the Project before the compilation

Back Up the project when saving
Create a project ZIP in the Folder **Versions** during the Saving
The ZIP file, can be recovery from the folder Versions, located in the current project folder

Perform Multiple Backup
Creates a project ZIP in the Folder **Versions** during the Saving with added a new Version to each ZIP
The ZIP file, can be recovery from the folder Versions, located in the current project folder

Show extended info
Show the compilation informations in the Output window

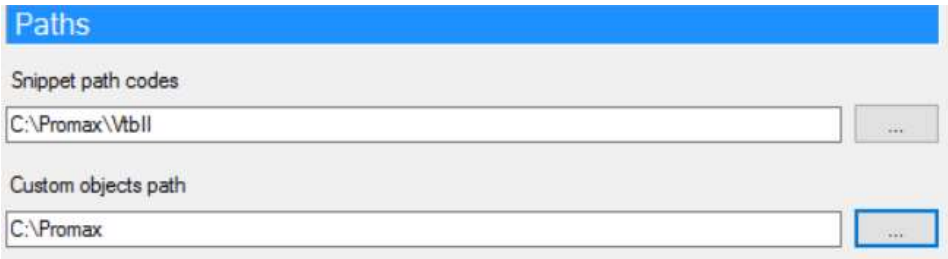
Use Faster functions Ladder diagrams
Uses the Fast functions for LADDER

Enabled advanced debugging
Set to enabled

Enabled debug on Objects code
Set to enabled

22.1.3 Paths

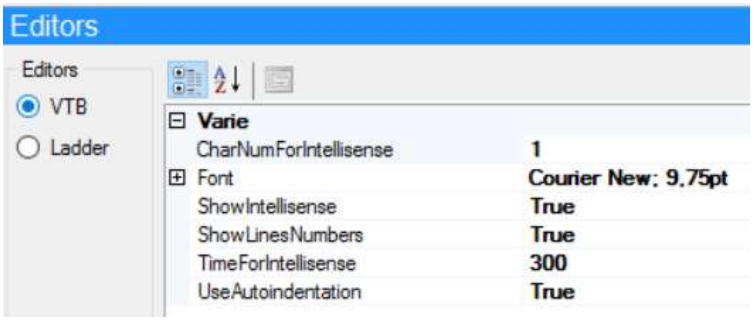
Defines the local paths for Snippets and Custom Objects
These paths are already inserted during the VTBII installation



22.1.4 Editors

VTBII Editors properties

VTB



CharNumForIntellisense
Number of characters before Intellisense activation

Font
Font type

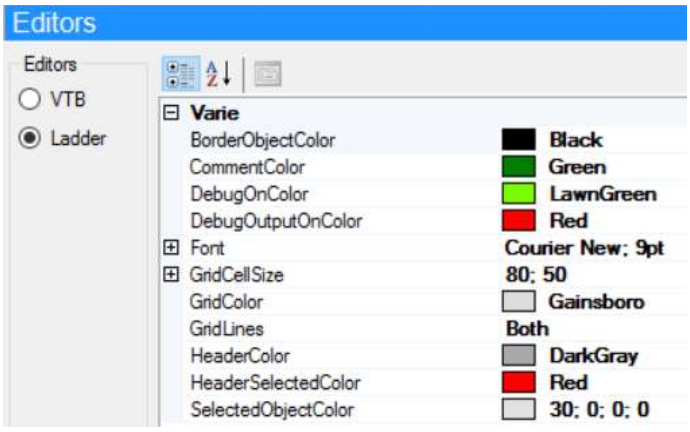
ShowIntellisense
True Intellisense Enabled

ShowLinesNumbers
True show lines number activated

TimeForIntellisense
Time in Milliseconds before Intellisense activation

UseAutoidentation
Set to True

LADDER



BorderObjectColor
Objects Border Color

CommentColor
Comment Color

DebugOnColor
Object color during debug when the INPUT is ON

DebugOutputOnColor
Object color during debug when the OUTPUT is ON

Font
Font Type

GridCellSize
Dimensione Griglia Oggetti

GridColor
Grid Color

GridLines
Grid Line type

HeaderColor
Network header Color

HeaderSelectedColor
Selected Network header Color

SelectedObjectColor
Selected Object Color

Index