VTB Visual Tool Basic

www.promax.it

Manuale D' Uso

PROMAX Motion & Control

Rev. 3.00.1

1 PREFAZIONE

VTB è un ambiente di sviluppo integrato per la programmazione ad OGGETTI su piattaforme PROMAX. L' ambiente riporta al suo interno tutti i tools necessari per lo sviluppo di applicazioni in modo semplice ed intuitivo. La filosofia di VTB si basa sulle recenti tecnologie R.A.D. (RAPID APPLICATION DEVELOPMENT) che permettono un rapido sviluppo di applicazioni scrivendo una quantità ridotta di codice. Una vasta libreria di OGGETTI e FUNZIONI TECNOLOGICHE permette di creare applicazioni per tutti i settori dell' automazione industriale. VTB integra un linguaggio ad alto livello tipo BASIC MOTION evoluto . E' possibile gestire in modo semplice e trasparente, FIELD BUS quali:

CAN OPEN

ETHERCAT

MODBUS

Potenti funzioni di MOVIMENTAZIONE ASSI permettono la gestione di qualsiasi tipo di macchina utilizzando funzioni per interpolazione LINEARE, CIRCOLARE, LINEARE VELOCE, ASSI ELETTRICI, PROFILI CAM ecc.

VTB predisposto per APPLICAZIONI MULTI LINGUA semplicemente selezionando la LINGUA DI UTILIZZO.

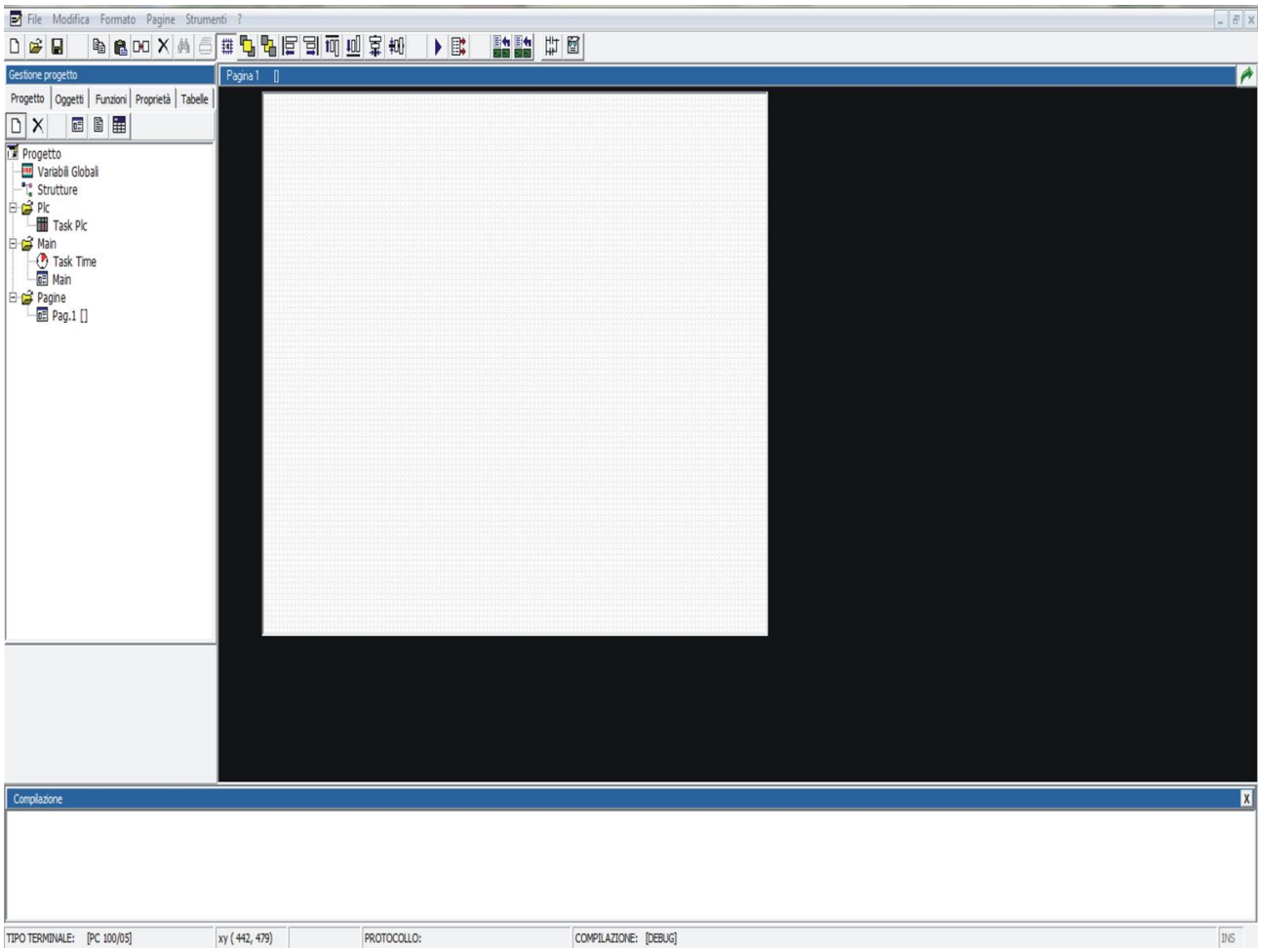
2 NOTE SUL LINGUAGGIO DI PROGRAMMAZIONE

VTB usa un linguaggio di programmazione definito **BASIC MOTION**.
La sintassi risulta essere molto simile ad un BASIC EVOLUTO con alcune terminologie derivate dal **LINGUAGGIO C**. La gestione delle funzioni è del tutto simile a **VISUAL BASIC** come le **STRUTTURE DATI**.
Alcune **ISTRUZIONI** sono invece **PROPRIETARIE DI VTB** ma comunque che seguono sempre una filosofia nota.
VTB è un linguaggio **CASE INSENSITIVE** cioè non fa differenza tra **MINUSCOLE** e **MAIUSCOLE** per quanto riguarda le istruzioni, funzioni, variabile ecc. **VTB** considera tutto in **MAIUSCOLO** convertendo automaticamente i caratteri. L'unica eccezione a questa regola riguarda la gestione delle **DEFINE** dove in presenza di caratteri minuscoli non vengono convertiti e rimangono tali fino in tutti i passaggi di compilazione.
Essendo un linguaggio indirizzato al MOTION, alcune funzionalità sono rimaste a livello PRIMITIVO poiché considerate di secondaria importanza, es il trattamento delle stringhe, vien effettuato a livello BASE come nel linguaggio C utilizzando le varie funzioni quali STRCPY, STRCAT, STRCMP ecc.

3 AMBIENTE DI SVILUPPO

L' ambiente di sviluppo VTB risulta avere un interfaccia intuitiva comune a tutti gli applicativi Windows. Tutte le funzioni sono a portata di CLICK e non è necessario avere una grande esperienza di programmazione.

L' ambiente è comprensivo di un EDITOR ottimizzato per la programmazione con VTB.



3.1 Barra dei pulsanti

Nuovo Progetto - Da menù ***File → Nuovo progetto***

Inizializza una nuova applicazione. La precedente viene abbandonata con richiesta di conferma per il salvataggio dei dati.

Apri Progetto - Da menù ***File → Apri progetto***

Apre un progetto esistente

Salva Progetto - Da menù ***File → Salva Progetto***

Salva un progetto

Copia Oggetto/i Selezionato - Da menù ***Modifica → Copia (Ctrl+C)***

Viene effettuata una COPIA dell' OGGETTO/I SELEZIONATO. Rimangono inalterate tutte le proprietà compresa la posizione dell' oggetto nella pagina. All' oggetto verrà assegnato il primo nome disponibile per quella classe. La copia viene effettuata in memoria e ripresa dalla funzione INCOLLA. **Il codice aggiunto negli eventi dell' oggetto non viene considerato.**

Incolla Oggetto/i Copiati - Da menù ***Modifica → Incolla (Ctrl+V)***

Vengono incollati nella pagina gli OGGETTI precedentemente copiati con la funzione COPIA. Rimangono inalterate tutte le proprietà compresa la posizione. La funzione INCOLLA risulta utile per effettuare pagine con uguali OGGETTI.

Duplica Oggetto/i Selezionati - Da menù ***Modifica → Duplica (Ctrl+D)***

Viene effettuata una COPIA dell' OGGETTO/I SELEZIONATO. Rimangono inalterate tutte le proprietà escluse quelle relative alla posizione dell' oggetto nella pagina. All' oggetto verrà assegnato il primo nome disponibile per quella classe.

Il codice aggiunto negli eventi dell' oggetto non viene considerato.

Elimina Oggetto/i Selezionati - Da menù ***Modifica → Elimina***

L' OGGETTO/I SELEZIONATO viene cancellato in modo definitivo dalla pagina eliminando anche i riferimenti al codice aggiunto negli eventi.

Cerca - Da menù ***Modifica → Trova***

Trova una stringa di testo nel progetto.

Stampa

Stampa il codice della finestra corrente.

Aggancia alla Griglia

Tutti gli OGGETTI inseriti nella pagina vengono vincolati al PASSO DELLA GRIGLIA se attivato il PULSANTE "AGGANCIA ALLA GRIGLIA". Risulta comodo per effettuare allineamenti degli oggetti in modo semplice ed immediato. Il passo della griglia viene impostato in PIXEL dal menù ***Opzioni -> Passo Griglia***.

Porta in primo piano

L' OGGETTO/i SELEZIONATO viene portato in primo piano sulla pagina rendendolo completamente visibile.

Porta in secondo piano

L' OGGETTO/i SELEZIONATO viene portato in secondo piano sulla pagina rendendolo non completamente visibile se questo si trova coperto da altri OGGETTI.

Allinea a sinistra

Gli oggetti selezionati vengono allineati sul margine sinistro, il riferimento viene preso da l' ultimo oggetto selezionato.

Allinea a destra

Gli oggetti selezionati vengono allineati sul margine destro, il riferimento viene preso da l' ultimo oggetto selezionato.

Allinea in alto

Gli oggetti selezionati vengono allineati sul margine alto, il riferimento viene preso da l' ultimo oggetto selezionato.

Allinea in basso

Gli oggetti selezionati vengono allineati sul margine basso, il riferimento viene preso da l' ultimo oggetto selezionato.

Centra orizzontale

Gli oggetti selezionati vengono centrati orizzontalmente, il riferimento viene preso da l' ultimo oggetto selezionato.

Centra verticale

Gli oggetti selezionati vengono centrati verticalmente, il riferimento viene preso da l' ultimo oggetto selezionato.

Compila Programma

L' intera applicazione viene compilata per renderla disponibile al trasferimento sulla piattaforma selezionata. I risultati della compilazione vengono riportati nella FINESTRA MESSAGGI. Ovviamente se ci sono errori di compilazione questi vanno corretti per poi effettuare una nuova compilazione.

Trasferisci Programma

Il programma precedentemente compilato viene trasferito alla scheda di controllo tramite linea seriale RS232 o linea ETHERNET. Il programma sarà inoltre salvato nella memoria permanente del controllo e quindi mandato in esecuzione.

Configuratore CanOpen

Viene attivato il Tool di configurazione della linea CaNOpen
(vedi capitolo CONFIGURATORE CAN OPEN).

Configuratore Ethercat

Viene attivato il Tool di configurazione della linea Ethercat
(vedi capitolo CONFIGURATORE ETHERCAT).

DEBUG

Attiva il DEBUG DELL' APPLICAZIONE.
(vedi capitolo DEBUG APPLICAZIONE)

3.2 Gestore Del Progetto

Il GESTORE DEL PROGETTO permette una rapida selezione e navigazione in tutte le PAGINE del PROGETTO stesso. Da questa AREA si ha in pratica l' intero controllo dell' applicazione, visualizzazione pagine, gestione variabili, introduzione codice ecc.

Nuova Pagina - Da menù ***Pagine → Nuova***

Aggiunge una nuova pagina al progetto. La pagina assume una numerazione automatica partendo dall' ultima disponibile. Le pagine possono contenere oggetti GRAFICI che verranno visualizzati nel terminale associato. Di fatto la pagina può contenere solo codice che viene eseguito solo quando questa viene caricata. Per passare da una pagina all'altra si utilizza la funzione di sistema:

Pagina(NrPag)

Elimina Pagina - Da menù ***Pagine → Elimina***

Elimina la PAGINA VISUALIZZATA. L' intero contenuto viene perso. Le pagine successive a quella eliminata vengono rinumerate, **quindi eventuali tasti di CAMBIO PAGINA dovranno essere modificati**.

Visualizza Grafica della Pagina

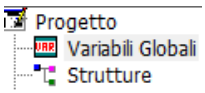
Visualizza l' ambiente grafico della Pagina.

Visualizza Codice della Pagina

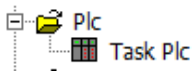
Visualizza l' ambiente di programmazione del codice della Pagina.

Visualizza Variabili della pagina

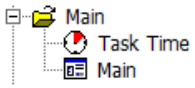
Visualizza la tabella delle variabili private della pagina in uso



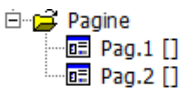
Visualizza Variabili GLOBALI e definizione STRUTTURE



Visualizza L' editor del TASK PLC



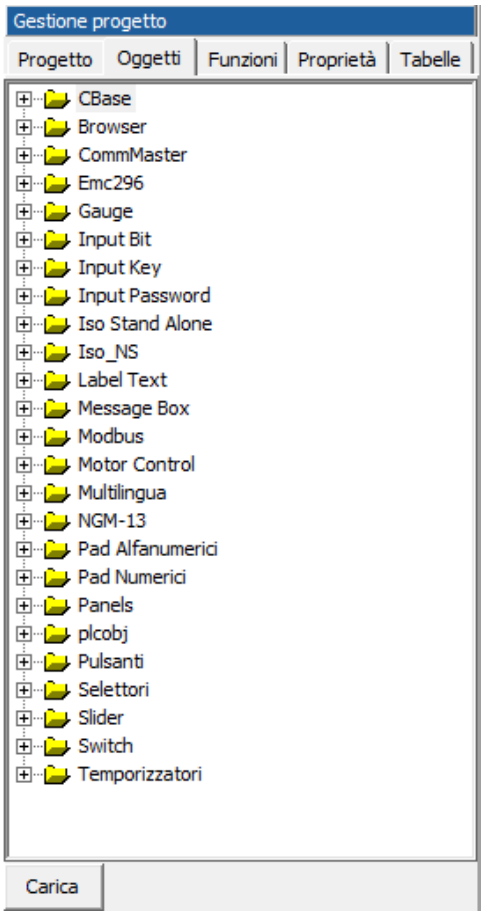
Visualizza L' editor della TASK MAIN o TASK TIME



Visualizza L' editor delle pagine

3.3 Gestore Oggetti

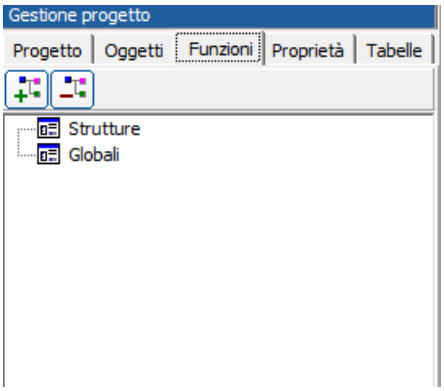
Il GESTORE DEGLI OGGETTI permette una rapida selezione degli oggetti da inserire nella pagina visualizzata.
In esso ci sono sia gli oggetti base che gli oggetti estesi, a cui si rimanda per una descrizione più dettagliata.
Per inserire un oggetto della classe base esso va prima selezionato e quindi trascinato.



Il pulsante CARICA permette di accedere ad OGGETTI personalizzati che non fanno parte degli OGGETTI standard di VTB ma che sono creati fa terze parti.

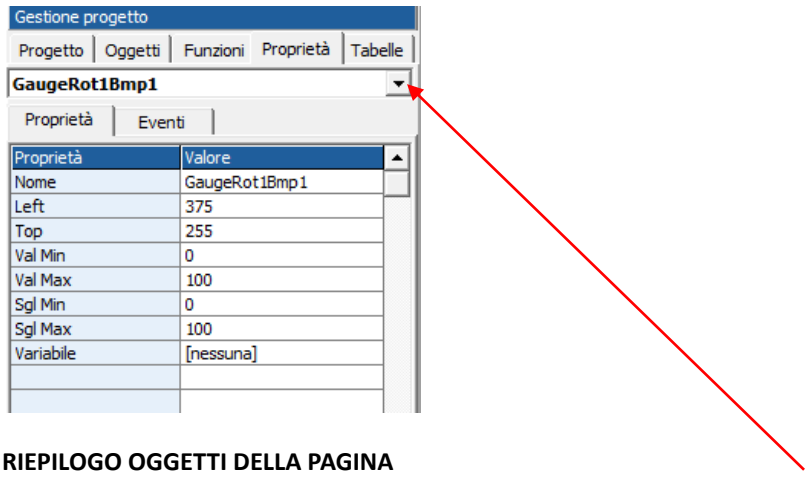
3.4 Gestore Funzioni

In questa Tree View vengono visualizzate tutte le STRUTTURE e le FUNZIONI, raggruppate per pagina. Basta aprire i singoli nodi per visualizzare le informazioni.
Nella sezione STRUTTURE esiste la possibilità di aggiungere una nuova struttura mediante il pulsante aggiungi un elemento, mentre è possibile rimuovere una struttura selezionata mediante il pulsante togli elemento.
Se si apre una struttura già esistente vengono mostrati i campi di cui è composta. Con un click sul campo è possibile modificare il tipo mentre con i due pulsanti aggiungi e togli elemento si può aggiungere o rimuovere un campo alla struttura.
La sezione FUNZIONI raggruppa le funzioni per pagina, selezionando una funzione viene aperta la finestra di editor posizionandosi all'inizio del codice.



3.5 Proprietà Oggetti

Dall' area PROPRIETÁ OGGETTI è possibile impostare tutte le caratteristiche di funzionamento di un OGGETTO sia grafiche che non. Le proprietà impostate verranno mantenute in fase di esecuzione del programma. In fase di sviluppo alcune proprietà di alcuni OGGETTI non creano nessuna conseguenza sulla rappresentazione grafica in quanto agiscono solo in fase di esecuzione del programma.
Per impostare una proprietà è sufficiente CLICCARE con il tasto sinistro del mouse nella voce desiderata e inserire il valore.
L'OGGETTO DEVE ESSERE STATO PRECEDENTEMENTE SELEZIONATO.



RIEPILOGO OGGETTI DELLA PAGINA
Per rendere più comodo la selezione degli OGGETTI PRESENTI NELLA PAGINA, è possibile utilizzare il MENÚ COMBO per una facile identificazione dell'OGGETTO DA SELEZIONARE. È sufficiente Selezionare il nome dell'oggetto dal menù COMBO.

3.6 Gestore Tabelle di Testo

Questa sezione è descritta in modo dettagliato più avanti alla sezione ***Tabelle di Testo*** del capitolo relativo alle ***VARIABILI.***

4 CONFIGURAZIONE DI VTB

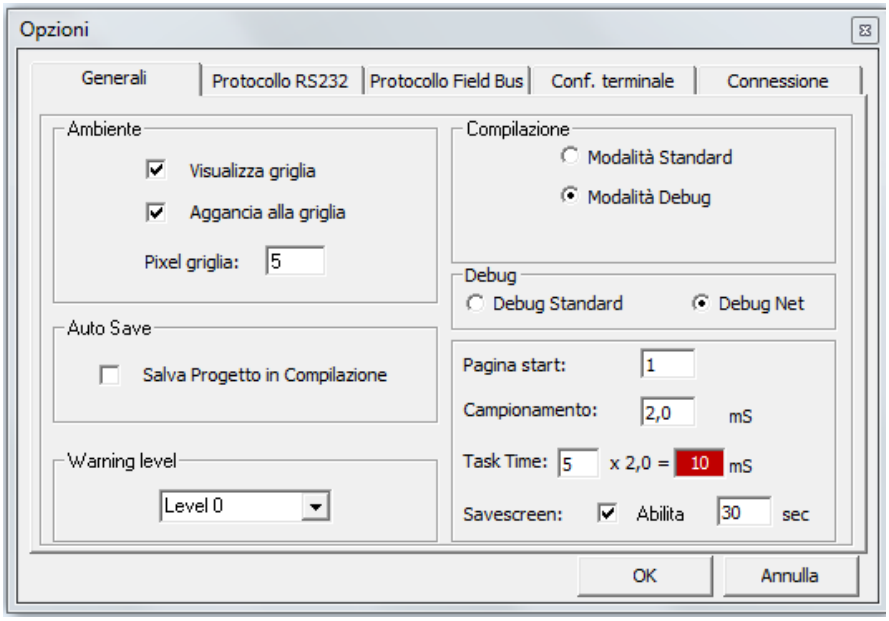
VTB per funzionare correttamente necessita di una configurazione a livello di sistema.
Tale configurazione prevede diverse categorie le quali interessano l'ambiente o l' applicazione che stiamo eseguendo.

ATTIVA CONFIGURAZIONE VTB

Da Menù ***Strumenti Opzioni***

4.1 Opzioni Generali

Questa tabella contiene le opzioni Generali di VTB



Visualizza Griglia
Se attivata la casella, viene visualizzata la griglia nelle pagine dell' ambiente VTB. La griglia risulta utile come riferimento per la posizione degli oggetti grafici.

Aggancia alla Griglia
Se attivata la casella, lo spostamento degli oggetti con il mouse viene agganciato al passo della griglia, ottenendo così allineamenti precisi.

Pixel Griglia
Numero di pixel di passo della griglia.

Pagina Start
Numero di pagina iniziale che deve essere visualizzato allo start nella piattaforma selezionata.

Campionamento
Tempo di esecuzione in millisecondi del TASK PLC, TASK CONTROLLO ASSI e TASK CANOPEN o ETHERCAT. Può essere variato a piacere tenendo presente che valori molto bassi in certi casi possono generare blocchi di programma. **Prendere sempre visione del tempo di esecuzione del TASK PLC tramite il DEBUG**.

Task time
Tempo di esecuzione del TASK TIME, espresso in multipli del tempo di campionamento del TASK PLC, il tempo risultante, in millisecondi, viene scritto a destra. Variando il campionamento del TASK PLC varia di conseguenza anche questo tempo

Savescreen
Tempo in secondi per l' intervento di abbassamento luminosità della lampada del display. Solo per controlli dotati di display con questa funzione implementata (es.).

Modalità Standard (OBSOLETO)
Compila il programma escludendo l' opzione per il DEBUG APPLICAZIONE. **Solo per compatibilità con i vecchi controlli a 16 bit**.

Modalità Debug (OBSOLETO)
Compila il programma includendo l' opzione per il DEBUG APPLICAZIONE. In questo caso non possono essere abilitati protocolli RS232 sulla prima porta seriale dei sistemi. **Solo per compatibilità con i vecchi controlli a 16 bit**.

Debug Standard (OBSOLETO)
Utilizza il DEBUG STANDARD di VTB. **Solo per compatibilità con le vecchie versioni**.

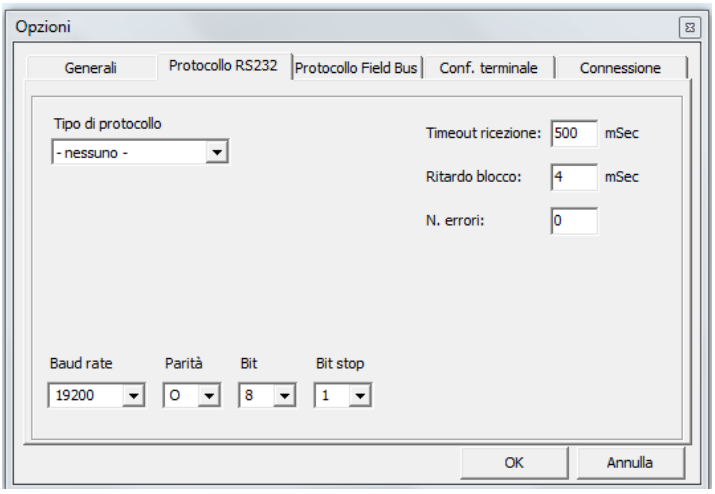
Debug.NET
Utilizza la nuova applicazione di DEBUG .NET. Sul PC deve essere installato il Framework 2.0 o superiore. Scelta consigliata.

Warning Level

Level 0 Non vengono generati avvisi dal compilatore
Level 1 Vengono generati degli avvisi quando vengono effettuate delle operazioni su variabili. Il compilatore avvisa ma comunque crea l' applicazione.

4.2 Protocollo RS232 (OBSOLETO)

Queste opzioni impostano il tipo di protocollo sulla prima porta seriale RS232.

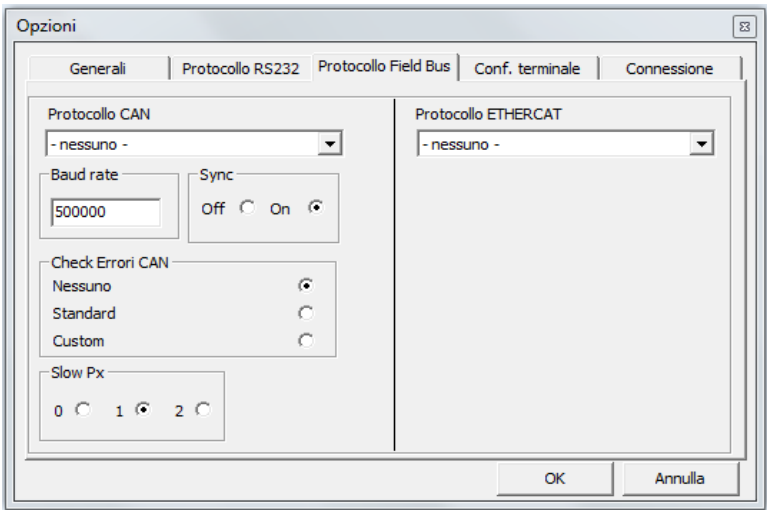


Solo per compatibilità con le vecchie versioni.

4.3 Protocollo FieldBus

Queste opzioni riguardano il protocollo FiedlBus Utilizzato.
Attualmente i protocolli implementati sono DUE:

CanOpen ***Standard DS301 DS4xxx***
Ethercat CoE ***(Can Over Ethercat)***



Protocollo CanOpen
Attivazione del protocollo tramite CheckBox

BaudRate
Seleziona il BaudRate della linea CanOpen

Sync
Attivazione dell messaggio di SYNC sulla linea CanOpen.
Il Sync viene inviato ad ogni campionamento dipendente dal tempo di selezione del TASK PLC da ***Opzioni Generali.*** **Il SYNC risulta INDISPENSABILE per sistemi con ASSI INTERPOLATI**

Chek Errori CAN
Indica il modo di visualizzazione di eventuali errori occorsi nella **configurazione** della linea CanOpen (vedi configuratore CanOpen), esistono tre opzioni:

Nessuno Su sistemi dotati di display viene visualizzato l'esito della configurazione di ogni singolo nodo dopodichè l'applicazione prosegue indipendentemente che ci siano stati errori oppure no. Utilizzando sistemi senza display non si ha nessuna indicazione di eventuali errori nella configurazione.

Standard **Questa opzione è valida solo su sistemi con terminale incorporato**.
Il sistema aggiunge automaticamente un Oggetto CanErr sulla pagina MAIN.
Viene visualizzato lo stato dei Nodi, ma se un NODO è in errore il programma si Blocca al Boot. Viene visualizzato un pulsante che permette comunque di continuare l' esecuzione del programma.

Custom Con questa opzione il sistema non esegue autonomamente nessuna azione ma richiama alcune funzioni per permette la personalizzazione della gestione degli errori di configurazione CanOpen.
Le funzioni richiamate dal sistema sono tre e dovranno essere definite nell'applicazione:

function open_cancfgerr(nodi as char) as void
nodi = Numero totale di nodi presenti in configurazione
Questa funzione viene richiamata dal sistema prima di eseguire la configurazione CanOpen. Nel parametro ***nodi*** viene passato il numero totale di nodi presenti nella configurazione.

function cancfgerr(nodo as int, err as uchar) as void
nodo=Numero nodo configurato
err=Esito della configurazione
0 = Nodo configurato correttamente
<>0 = Codice di Errore. Vedi capitolo relativo alle funzioni CanOpen
Viene chiamata al termine della configurazione di ogni nodo inviando l'esito della stessa nel parametro ***err***.

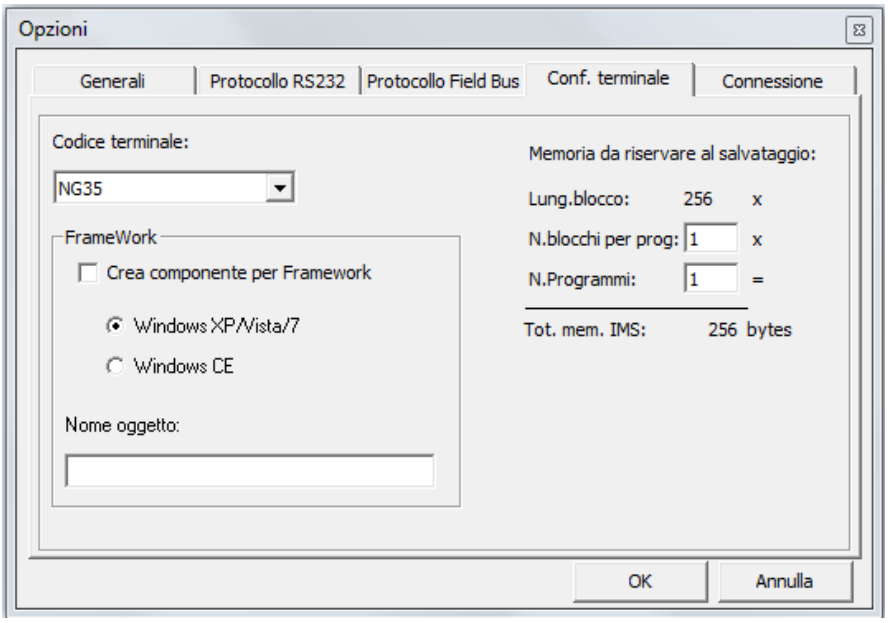
function close_cancfgerr() as void
Viene chiamata al termine della configurazione di tutti i Nodi.

Slow Px
Di default viene impostato ad uno ma per compatibilità con tutti i sistemi si consiglia di tenerlo sempre a ZERO. Utilizzato per future espansioni.

Protocollo Ethercat
Attivazione del protocollo Ethercat per sistemi che gestiscono questo protocollo.
Ethercat può convivere insieme al protocollo CanOpen.

4.4 Configurazione piattaforma hardware

Prima di iniziare un nuovo progetto occorre sempre configurare VTB adattandolo alla piattaforma hardware in uso. Gli hardware disponibili sono diversi e con diverse combinazioni tra essi. Occorre scegliere quello appropriato. Possiamo cambiare la piattaforma anche successivamente in modo da adattare la stessa applicazione ad un altro hardware.



Codice Terminale

Combo che permette di scegliere la piattaforma Hardware.

Per facilitare la programmazione ed integrare al meglio l' ambiente di sviluppo, esistono delle combinazioni hardware preconfigurate quali ad esempio:

NGM13/LPC20 – NG35/LPC40 ecc.

Queste si riferiscono ad una CPU NGM13 o NG35 accoppiata ad un terminale Promax LPC20, LPC40.

Memoria da riservare al salvataggio

Questo indica la memoria FLASH interna riservata per il salvataggio dei dati (es. oggetti ricette ecc.) IMS. Questa memoria è organizzata in blocchi di 256 byte. Pertanto occorre selezionare il numero dei blocchi riservati ad ogni ricetta, e il numero max di ricette diverse. Pertanto se una ricetta è composta da 300 byte, occorre selezionare 2 blocchi per ricetta (512 byte). La memoria IMS viene tolta da quella riservata all' applicazione, quindi occorre un accurata verifica alla quantità da riservare alla memoria IMS. **Questa opzione non è valida per le piattaforme che non condividono la memoria FLASH CODICE con quella salvataggio dati (es. NGM13-NGMEVO).**

Crea componente per framework

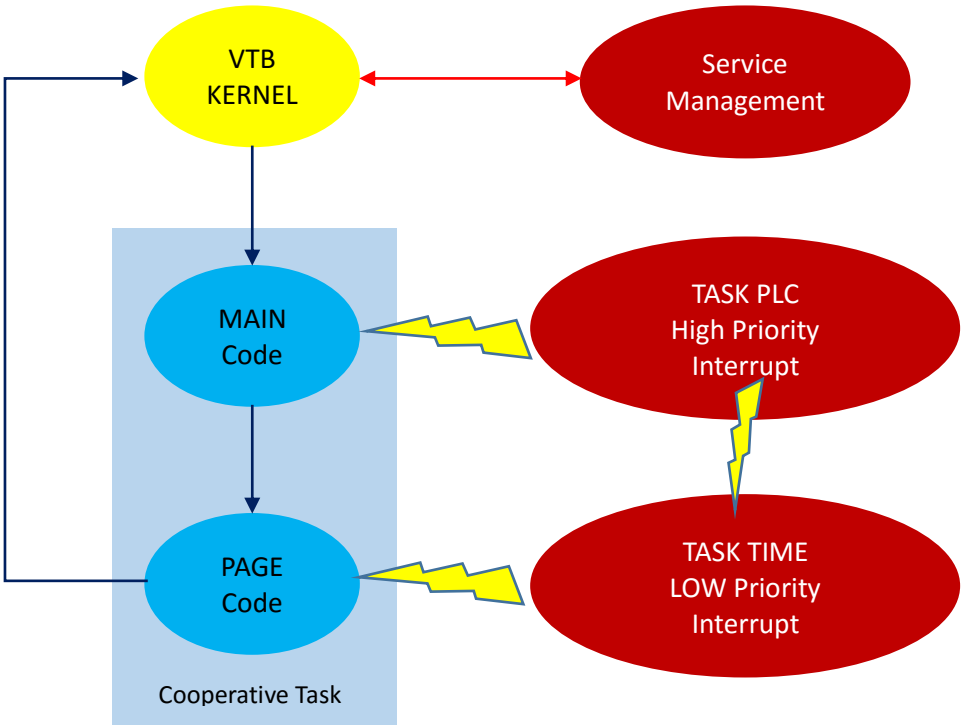
VTB può compilare una DLL Component Model da integrare in applicazione Framework .NET.

Questa permette un controllo diretto delle risorse Hardware da parte di Host esterni quali PC dotati di sistema operativo tipo Windows XP,Vista, 7, CE e altri che integrano il Framework. (Vedi Capitolo Componente per Framework).

Occorre selezionare il tipo di componente da Creare Windows Xp o Windows CE e il nome del componente DLL. Questo verrà creato nella stessa cartella del Progetto VTB.

5 TASK GESTITI DA VTB

VTB mette a disposizione del programmatore quattro tipi di TASK che possono essere combinati tra di loro per creare un applicazione.
Due di essi sono gestiti ad ***interrupt***, vengono cioè eseguiti, interrompendo gli altri processi, a tempi fissi e costanti; gli altri due processi in modo ***cooperativo***, vengono cioè eseguiti uno di seguito all'altro. Il **TASK PLC** è il task ***DETERMINISTICO*** a più alto livello che interrompe tutti gli altri task, il **TASK TIME** ha una priorità più bassa del TASK PLC, infine il **TASK** di **PAGINA** e il **MAIN** girano in modo cooperativo tra di loro e sono interrotti dagli altri due task.



5.1 Task Plc

Questo risulta essere il **TASK** con più alta priorità: il suo utilizzo è deterministico e pertanto è consigliato per gestire situazioni che devono avere tempi di risposta VELOCI e precisi. Questo TASK non può essere INTERROTTO da nessun altro TASK ma può invece interrompere ogni altro. Di norma viene utilizzato dagli OGGETTI per il CONTROLLO ASSI o cicli PLC VELOCI, ma può contenere qualsiasi tipo di codice o di gestione ad esclusione di alcune funzioni IFS tipo:
GESTIONE GRAFICA
FUNZIONI DI INTERPOLAZIONE ASSI quali PX_MOVETO,LINE_TO
SDO CANOPEN.
CICLI STATICI
(vedere in dettaglio ogni singola funzione)
Il suo tempo di campionamento tipico risulta essere di 2 millisecondi, tale tempo è sufficiente per gestire molte applicazioni (tipo gestione di interpolazione di 6 ASSI), comunque può scendere anche sotto 1 millisecondo quando il carico di lavoro è meno stressante e per tipi di CPU con elevata potenza di calcolo. In questo TASK vengono gestiti anche i protocolli CAN OPEN e ETHERCAT in modo DETERMINISTICO. È consigliabile comunque che il suo TEMPO CPU UTILIZZATO non superi il 60% del suo tempo di campionamento impostato, altrimenti si rischia di rallentare o congelare gli altri TASK. Il **TASK PLC NON HA UNA SEZIONE PER INSERIMENTO DEGLI OGGETTI**, pertanto il codice di questi che deve girare al suo interno, deve essere definito al momento della progettazione dell'oggetto e
NON SI DEVONO INSERIRE ISTRUZIONI O CICLI CHE POSSONO BLOCCARLO.
Per verificare il tempo di esecuzione del TASK PLC è sufficiente da DEBUG.NET vedere il valori dei campi:
PLC TP e **PLC TM** questi non devono mai superare il tempo di campionamento impostato.
Il task plc ha anche una sezione di INIT che sarà eseguita solo una volta al reset del sistema.

5.1.1 NOTA SULLA PROGRAMMAZIONE CONCORRENTE

Occorre fare particolare attenzione alla tipologia di programmazione Concorrente relativa a tutti i sistemi MULTITASK. In pratica non è consigliabile richiamare le stesse funzioni dai TASK ad INTERRUPT e da quelli cooperativi. In pratica le funzioni gestite dal TASK MAIN possono essere tranquillamente gestite dal TASK DI PAGINA, ma NON ANCHE dal TASK TIME e TASK PLC e viceversa.
Questo poiché se da un TASK COOPERATIVO viene chiamata una funzione e durante l'esecuzione di questa un TASK AD INTERRUPT chiama la stessa funzione, questa potrebbe portare a dei malfunzionamenti dell'applicazione.

CONDIVISIONE DELLE VARIABILI

Possono esistere anche problemi di condivisione delle variabili tra i TASK AD INTERRUPT e i TASK COOPERATIVI. In pratica se la gestione della variabile non prevede un ISTRUZIONE ASSEMBLER ATOMICA, questa può assumere valori sbagliati a causa della condivisione.
I problemi di condivisione si possono verificano per i seguenti tipi:

Sistema	Tipo
Tutti	FLOAT

Tuttavia per ovviare a questo inconveniente VTB offre la possibilità di CONDIVISIONE SICURA DELLE VARIABILI. Nella fase di dichiarazione delle variabili è presente un Check che indica se la variabile è condivisa tra i diversi TASK. Abilitando questo si risolvono i problemi di condivisione. Un uso esagerato di variabili con la condivisione sicura abilitata può creare problemi di jitter quindi ***utilizzare l'abilitazione della condivisione solo quando STRETTAMENTE NECESSARIO***
Lo stesso problema può avvenire anche quando si utilizzano tabelle di dati condivise da più processi. Un semplice esempio può essere l'utilizzo di tabelle di scambio dati per protocolli MODBUS. I problemi possono nascere quando per esempio il processo di scrittura dei dati è asincrono rispetto a quello di lettura. Può accadere infatti che il processo di lettura inizi quando quello di scrittura ha riempito la tabella solo a metà. In questo caso ci troviamo a leggere alcuni dati nuovi e altri relativi alla vecchia scansione. E' evidente che in questa situazione si possono avere delle false letture. Il sistema non è in gradi di capire queste situazioni e quindi per risolverli occorre inserire dei ***semafori*** a livello di applicazione.

5.2 Task Time

Il TASK TIME per certi aspetti risulta analogo al TASK PLC, si differenzia da questo per due caratteristiche:
a) ha una priorità minore e può essere INTERROTTO dal TASK PLC stesso;
b) non ha limiti ha l'inserimento di funzioni IFS di VTB.
Il tempo di scansione di questo task è programmabile a multipli del tempo di campionamento del TASK PLC. Questo TASK risulta utile per la gestione di cicli temporizzati e con risposte comunque veloci, inoltre il fatto di poter richiamare tutte le funzioni IFS, lo rende di grande utilità, garantendo al software tempi precisi e costanti. Un suo tempo di campionamento tipico può essere nell'ordine di 5 o 10 millisecondi, tempo in cui può essere gestito anche un complesso ciclo PLC con molti punti di I/O. Nel caso che il tempo di durata di questo task superi il proprio tempo di scansione il sistema continuerà a funzionare bloccando i task cooperativi ma facendo girare il task plc.
Il TASK TIME HA UN APPOSITA SEZIONE PER INSERIMENTO DEGLI OGGETTI, pertanto tutti gli oggetti inseriti al suo interno saranno eseguiti nel CAMPIONAMENTO definito del TASK TIME.

5.3 Task Main

Il TASK MAIN viene sempre richiamato dal ciclo di VTB facendolo eseguire in modo COOPERATIVO con il TASK di pagina, cioè la loro esecuzione dipende dal codice contenuto. Pertanto un ciclo statico su TASK MAIN bloccherà il TASK DI PAGINA e viceversa. Il suo tempo di esecuzione dipende da tutti gli altri TASK. Normalmente questo TASK esegue i cicli ricorsivi tipo controllo stati di emergenza, stati di allarme, controllo della grafica ecc. dove non c'è la necessità di tempi di risposta costanti. Comunque il suo tempo di esecuzione può a sua volta essere molto veloce, anche nell'ordine di pochi ***microsecondi*** quando il codice contenuto è di poche righe.
Il **TASK MAIN HA UN APPOSITA SEZIONE PER INSERIMENTO DEGLI OGGETTI**, pertanto tutti gli oggetti inseriti al suo interno saranno eseguiti in modo COOPERATIVO e indipendentemente da quale pagina sia visualizzata.

Il **TASK MAIN** mette a disposizione tre punti di inserimento del CODICE che sono:

INIT PAGINA

MASTER CICLO

FUNZIONI DI PAGINA

E' presente anche una sezione **MASTER EVENT** ma è stata lasciata solo per compatibilità con le vecchie versioni e quindi **non deve essere utilizzata**.

INIT PAGINA
Viene eseguita una sola volta alla partenza del programma, normalmente serve come inizializzazione delle variabili globali utilizzate nell'applicazione. In questo punto di ingresso l'operatore può inserire del codice a piacere purché non sia codice STATICO, cioè che blocchi il programma in quel punto.

MASTER CICLO
Viene eseguita ciclicamente. Il tempo di esecuzione dipende dalla lunghezza della stessa MASTER CICLO e dal CODICE DI PAGINA. In questo ciclo possono essere inserite tutte le parti di programma che devono essere sempre eseguite (es. controlli stati di emergenza ecc.).

FUNZIONI DI PAGINA
In questa pagina si possono scrivere tutte le funzioni utilizzate dal sistema. Queste saranno visibili GLOBALMENTE da tutti i TASK.

5.4 Task di Pagina

Il TASK DI PAGINA è analogo al TASK MAIN, con il quale condivide il tempo di esecuzione in modo COOPERATIVO. La particolarità di questo TASK è che il suo codice viene caricato solo insieme alla pagina in esecuzione. La funzione IFS – ***pagina(n)*** permette di mettere in esecuzione una pagina, programmata con l'ambiente VTB, distruggendo la precedente. Le PAGINE sono fisicamente porzioni di un intero codice/grafica che possono essere gestite a piacere. Nei controlli con interfaccia HMI, le pagine rappresentano anche la parte grafica che viene visualizzata al momento. Nei controlli senza interfaccia HMI, le pagine rappresentano solamente il codice che viene eseguito al momento. Come per il TASK MAIN il tempo di scansione dipende dal codice presente in tutti gli altri TASK. Normalmente il TASK DI PAGINA esegue i cicli di impostazione, preparazione e visualizzazione dati della macchina, con controllo della grafica e dei dati di input. Il **TASK DI PAGINA HA UN APPOSITA SEZIONE PER INSERIMENTO DEGLI OGGETTI**, pertanto tutti gli oggetti inseriti al suo interno saranno eseguiti in modo COOPERATIVO.
Questo TASK mette a disposizione 3 punti di ingresso per l' inserimento del codice che si differenziano per il tipo di esecuzione.

INIT PAGINA

MASTER CICLO

FUNZIONI DI PAGINA

E' presente anche una sezione **MASTER EVENT** ma è stata lasciata solo per compatibilità con le vecchie versioni e quindi **non deve essere utilizzata**.

INIT PAGINA
Viene eseguita una sola volta al caricamento della PAGINA. In questo punto di ingresso l'operatore può inserire del codice a piacere purché non sia codice STATICO, cioè che blocchi il programma in quel punto.

MASTER CICLO
Viene eseguita ciclicamente. Il tempo di esecuzione dipende dalla lunghezza della stessa MASTER CICLO e dal CODICE DI PAGINA. In questo ciclo possono essere inserite tutte le parti di programma che devono essere sempre eseguite (es. controlli stati di emergenza ecc.).

FUNZIONI DI PAGINA
In questa pagina si possono scrivere tutte le funzioni utilizzate dalla pagina stessa. Queste non saranno visibili dagli altri TASK.

6 TIPI DI VARIABILI GESTITI DA VTB

VTB può gestire diversi tipi di variabili che possono essere utilizzate durante la fase di programmazione.
Normalmente le VARIABILI vengono allocate nella MEMORIA VOLATILE (RAM) del sistema e azzerate al reset. In alcune piattaforme dotate di RAM NON-VOLATILE (come NG35 o) possono essere allocate anche in questa area e vengono definite STATIC VAR. Queste ultime conserveranno il loro valore anche dopo lo spegnimento. Le VARIABILI seguono una terminologia STANDARD presa dai comuni linguaggi di programmazione.
Possono essere comunque dichiarate VARIABILI che fanno riferimento a componenti esterni (es. configuratre CANOPEN o ETHERCAT). Queste vengono gestite automaticamente dal sistema in modo trasparente all'operatore.

6.1 Valori Numerici

VTB gestisce i valori numerici nel modo convenzionale utilizzato dai vari compilatori. Un valore numerico può essere inserito sia nella **NOTAZIONE DECIMALE**, sia nella **NOTAZIONE ESADECIMALE** facendo precedere il numero dal prefisso 0x (ZERO X). Per esempio il numero 65535 decimale viene tradotto dal numero 0xFFFF esadecimale.
I valori FLOATING-POINT vengono inseriti con il punto decimale (non la virgola). I valori FLOAT non possono essere inseriti nel formato esadecimale

Esempio:
A=1236 *' assegna il valore 1236 alla variabile A*
A=0x4d *' assegna il valore esadecimale 0x4d alla variabile A*
' corrispondente a 77 decimale
B=1.236 *' assegna il valore alla variabile floating-point a 1,236*

6.2 Variabili Interne

Questo tipo di variabili vengono dichiarate nella memoria RAM volatile della piattaforma Hardware e sono azzerati al reset.
I tipi di variabili utilizzati da VTB rispecchiano i principali tipi definiti dai linguaggi di programmazione e sono i seguenti:

TIPO	DIMENSIONE	RANGE
BIT	1 bit	Da 0 a 1
CHAR	8 bit consegno	da –128 a +127
UCHAR	8 bit senza segno	da 0 a 255
INT	16 bit con segno	da –32.768 a +32.767
UINT	16 bit senza segno	da 0 a 65.535
LONG	32 bit con segno	da –2.147.483.648 a +2.147.483.647
FLOAT	64 bit (formato standard DOUBLE) IEEE 75	Da -1,79769313486232e308 a +1,79769313486232e308
STRING	Non supportato come variabile ma solamente come costante	
VETTORE	Monodimensionale per tutti i tipi di variabile escluso BIT	
STRUTTURE	Dichiarazione standard	
PUNTATORI	Char, Uchar, Int, Uint, Long, Float 32 bit	
DELEGATI	Puntatori a FUNZIONI 32 bit	

È opportuno utilizzare le variabili in base al valore minimo e massimo che queste dovranno contenere scegliendo il tipo più adatto. Le variabili INTERNE possono essere dichiarate **LOCALI DI PAGINA** oppure **GLOBALI** e quindi utilizzabili da tutte le pagine. La differenza tra LOCALI DI PAGINA e GLOBALI viene semplicemente definita dal punto di dichiarazione delle variabili.

VARIABILI LOCALI DI PAGINA Vengono dichiarate nella PAGINA e sono visibili solo a questa

VARIABILI GLOBALI Vengono dichiarate nel TASK MAIN e sono visibili a tutti

VTB non effettua nessun controllo sulla dimensione delle variabili ed il loro valore assegnato.

6.3 Puntatori

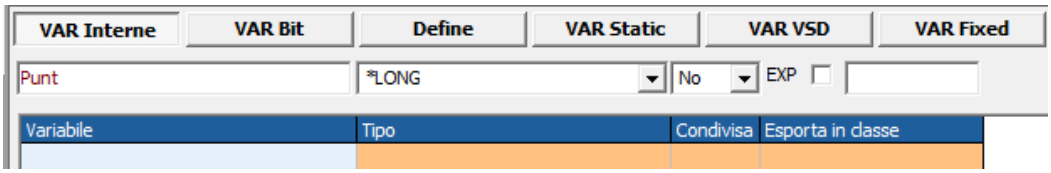
VTB è in grado di utilizzare anche i puntatori alle variabili. Il puntatore definisce non il contenuto della variabile, ma l'indirizzo di memoria dove questa è allocata. Alcune funzioni di VTB necessitano del passaggio del puntatore. Questo vale anche per i vettore dove viene passato il puntatore al primo elemento.
Per utilizzare il puntatore di una variabile è sufficiente inserire il postfisso ***()*** eccetto che per le funzioni:

Esempio:
provavar as long
vett(20) as uint

provavar() ***Passa il puntatore della variabile provavar***
vett() ***Passa il puntatore del primo elemento di vett***

I possono essere dichiarati solo ai seguenti tipi di variabili interne:
Char, Uchar, Int, Uint, Long, Float e funzioni

Dichiarazione di Un puntatore



Per assegnare un indirizzo al puntatore è necessario:
prendere il puntatore senza parentesi quadre
assegnare al puntatore l'indirizzo desiderato

Per assegnare un valore al campo puntato è necessario:
prendere il puntatore con parentesi quadre
Indirizzare il valore interno al giusto indice
assegnare il valore al puntatore;

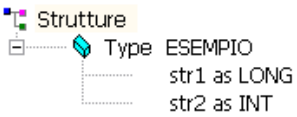
Esempi
Variabili utilizzate:
punt as *long
val as long
puntatore as *long
array(10) as long
variabile as long

Scrittura/lettura variabili tramite puntatore:
punt=val() ***'assegna a punt l'indirizzo della variabile val***
punt[0]=2000 ***' punt[0] corrisponde alla variabile val che assumerà il valore 2000***
variabile=punt[0] ***'assegna a variabile il contenuto di val tramite il puntatatore punt***

Scrittura/lettura array tramite puntatore:
puntatore=array() ***'assegna a puntatore l'indirizzo dell'array***
puntatore[0]=13
puntatore[1]=27
puntatore[9]=55 ***'assegna all'array tramite puntatore alcuni valori***
variabile=puntatore[7] ***'assegna a variabile il contenuto di array[7]***

E' possibile dichiarare un puntatore anche alle STRUTTURE di dati.

Esempio
E' stata dichiarata la seguente struttura



Variabili utilizzate:
puntatore as *ESEMPIO ***'puntatore alla struttura esempio***
struct as ESEMPIO ***'struct è una variabile di tipo struttura***

puntatore=struct() ***'il puntatore punta alla struttura***
puntatore->str1=300 ***' aggiorna entrambi i campi della struttura per mezzo del puntatore***
puntatore->str2=200

Per quanto riguarda i puntatori alle strutture dati, occorre utilizzare il token → per accedere al puntatore

ATTENZIONE: VTB non effettua nessun controllo sull'elemento del puntatore, pertanto **scrivere su indirizzi diversi da quello assegnato al puntatore è possibile ma questo può generare un malfunzionamento del programma**.

Esempio:
punt as *long
valore as long

punt=valore()
punt[10]=valore
L' istruzione punt[10] = valore non genera nessun errore di compilazione o di run-time, però causa sicuramente problemi di funzionamento al programma andando a scrivere in locazioni di memoria imprecisate. L'istruzione corretta è:

punt[0]=valore

Per utilizzare il puntatore ad una funzione è sufficiente assegnare ad una variabile il nome della funzione stessa senza eventuali argomenti:

Esempio
VarPunt=MiaFun
Dove MiaFun è una funzione

6.4 Variabili BIT

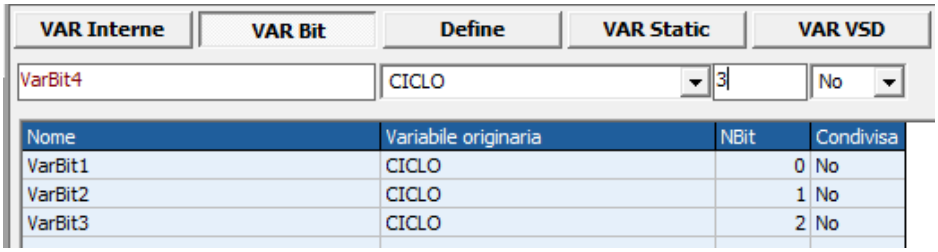
Questo tipo di variabili possono avere solo due stati 0 o 1 che normalmente sono associate ad uno stato OFF/ON oppure FALSO/VERO. La variabile BIT deve sempre essere associata ad una variabile intera che potrà quindi contenere più bit.
Queste variabili risultano molto utili per la gestione di FLAG, line I/O digitali e tutti quei casi dove è necessario leggere o scrivere direttamente un bit di una variabile.
Le variabili a bit possono essere sia GLOBALI che di PAGINA e vengono utilizzate come normali variabili.
Per esempio dichiarando una variabile INTERNA di nome CICLO e di tipo INT (16 bit) è possibile associare a questa 16 variabili bit.

VARBIT1CICLO.0 (primo bit di CICLO)
VARBIT2CICLO.1 (secondo bit di CICLO)
.
VARBIT16 CICLO.15 (sedicesimo bit di CICLO)

```
If VARBIT1 = 1          ' controlla se il primo bit di CICLO è settato
        VARBIT2=1       ' setta il secondo bit di CICLO
        VARBIT3=0       'reset del terzo bit di CICLO
endif
```

Un comune utilizzo di queste variabili è la gestione dei punti di **INPUT** ed **OUTPUT** digitali del sistema sia che essi risiedano internamente al sistema (es. NGIO) o siano dei canali remotati tramite linea **CAN OPEN** o **ETHERCAT.** Nel primo caso i bit faranno capo a delle normali variabili interne mentre nel secondo caso si appoggeranno a quelle di tipo **VCB**. Quindi dichiarando le variabili bit si potrà controllare fisicamente lo stato di queste linee di I/O semplicemente leggendo o scrivendo la variabile stessa.

DICHIARAZIONE di una VARIABILE BIT



CAMPI DI INSERIMENTO VARIABILI BIT

Nome Identifica il nome UNIVOCO della variabile a bit
Variabile Originaria Variabile a cui associare la variabile bit. Deve essere di tipo CHAR, UCHAR, INT, UINT, LONG (anche VETTORI)
Nbit Numero del bit associato alla variabile originaria.
ATTENZIONE: il primo bit è sempre il numero 0 (zero)

l numero di bit massimo dipende dal tipo della variabile originaria:
CHAR/UCHAR **0-7** (8 bit)
INT/UINT **0-15** (16 bit)
LONG **0-31** (32 bit)

6.5 Vettori

I vettori possono essere dichiarati nelle variabili INTERNE o in quelle STATIC e possono essere di qualsiasi tipo escluso BIT. I vettori gestiti da VTB sono MONODIMENSIONALI e il limite massimo di dimensione dipende dalla memoria libera disponibile nel progetto. Per dichiarare un vettore è sufficiente durante la fase di dichiarazione di una variabile interna inserire oltre al nome le parentesi tonde che indicano la dimensione voluta.
Nel caso ci fosse la necessità di utilizzare un vettore BIDIMENSIONALE (matrice) occorre lavorare con le STRUTTURE. E' sufficiente dichiarare una struttura con un vettore al suo interno quindi dichiarare un vettore di tipo struttura.

VETT(10) Vettore di 10 dimensioni

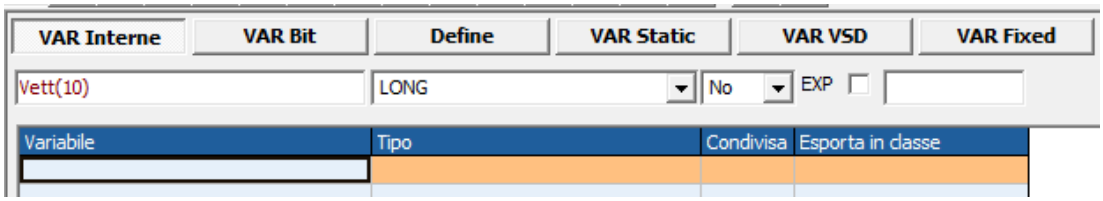
Il primo elemento del vettore parte sempre da 0 (zero) pertanto avremo:

VETT(0) primo elemento
VETT(9) ultimo elemento

Alcune funzioni di VTB richiedono l'indirizzo del vettore e questo si specifica inserendo il nome del vettore con le parentesi aperte e chiuse senza specificarne l'indice (vedi puntatori).

VETT() ***Passa alla funzione l' indirizzo in memoria del vettore VETT***

DICHIARAZIONE DI UN VETTORE



ATTENZIONE: VTB non effettua nessun controllo sull'indice del vettore permettendo quindi di scrivere oltre la posizione massima. Ciò potrà causare dei malfunzionamenti al programma.

6.6 Variabili VCB (CanOpen o EtherCAT)

Le variabili di tipo VCB sono di fatto delle normali variabili che rispecchiano lo stato di variabili presenti su dispositivi remoti esterni connessi all'unità centrale tramite bus di campo tipo CANOPEN o ETHERCAT. Queste variabili non vengono definite direttamente tramite l'ambiente VTB ma provengono da un configuratore esterno che definisce la tipologia del bus di campo e dei dispositivi collegati. In pratica la loro dichiarazione risulta molto semplice in quanto il configuratore automaticamente genera una tabella che viene interpretata da VTB rendendo le variabili disponibili agli OGGETTI e al CODICE AGGIUNTO. Per maggiori informazioni rimandiamo ai CAPITOLI ***CONFIGURATORE CANOPEN e CONFIGURATORE ETHERCAT***.
Le variabili VCB in pratica rappresentano le risorse condivise con un dispositivo esterno collegato all bus di campo. Per esempio un driver per motori brushless metterà a disposizione una serie di variabili che faranno riferimento al MOTION, mentre una scheda di I/O metterà a disposizione delle variabili che faranno riferimento alla gestione delle linee di INPUT e OUTPUT.
A differenza degli altri tipi di variabili, le VCB sono solamente GLOBALI e quindi visibili da tutte le pagine e da tutti i task. Le variabili VCB dichiarate dal configuratore possono essere quindi utilizzate sia nel CODICE AGGIUNTO sia nelle proprietà degli OGGETTI che ne fanno utilizzo
Non esiste di fatto una tabella per la dichiarazione di queste variabili, ma semplicemente per il loro utilizzo basta utilizzarle nel codice.

UTILIZZO DI UNA VARIABILE VCB NEL CODICE AGGIUNTO

Per utilizzare una variabile VCB nel codice aggiunto è sufficiente fare riferimento al nome associato.

```
If encoderx >=10000        ' Ciclo if su encoder asse X

.............

endif
```

6.7 Variabili System

Le variabili SYSTEM sono delle variabili predefinite dal sistema operativo interno, quindi non devono essere dichiarate ma possono essere utilizzate come normali variabili. Di seguito viene rilasciato un elenco delle variabili SYSTEM disponibili, ci sono ulteriori variabili ma riservate al sistema.

NOME	TIPO	R/W	FUNZIONE
_SYSTEM_PXC	LONG	R/W	Utilizzate nei sistemi con NGM13-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_ACT_PAGE	INT	R	Contiene il numero di pagina attualmente caricato/visualizzato.
_SYSTEM_OLD_PAGE	INT	R	Contiene il numero di pagina precedentemente caricato/visualizzato.
_SYSTEM_STRING(128)	CHAR	R	Array di 128 elementi che contiene la stringa letta dalla funzione ***Get_TabStr(…..)***
_SYSTEM_LINGUA	CHAR	R/W	Contiene la LINGUA utilizzata nell'applicazione. La lingua è un numero da 0 a 127 che selezione i messaggi della relativa tabella.
_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_TLUCE	LONG	R/W	Contiene il tempo in millisecondi di accensione della lampada del display (solo con dispositivi dotati di HMI).
_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_CARD_TYPE	INT	R	Se presente un SSD interno contiene la sua dimensione in Mbyte (8, 16, 32, 64, 128, ecc.).
_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
_SYSTEM_STDINP_DN	INT	R	Contiene il codice del tasto quando premuto
_SYSTEM_STDINP_UP	INT	R	Contiene il codice del tasto quando rilasciato

6.8 Variabili Static

Le variabili di tipo STATIC vengono dichiarate in una memoria RAM tamponata. Pertanto il loro valore rimane inalterato anche quando il sistema è spento. Risultano molto utili per il salvataggio di dati che variano frequentemente (come encoder, contatori ecc.), che non potrebbero essere salvati in memoria flash. Il loro utilizzo è identico ad una variabile interna, pertanto sono disponibili tutti i tipi di variabili gestiti da VTB.
Le variabili STATIC sono sempre GLOBALI cioè visibili in tutte le pagine e in tutti i task.

TIPO	DIMENSIONE	RANGE
BIT	1 bit	Da 0 a 1
CHAR	8 bit con segno	da –128 a +127
UCHAR	8 bit senza segno	da 0 a 255
INT	16 bit con segno	da –32.768 a +32.767
UINT	16 bit senza segno	da 0 a 65.535
LONG	32 bit con segno	da –2.147.483.648 a +2.147.483.647
FLOAT	64 bit (formato standard DOUBLE) IEEE 75	Da -1,79769313486232e308 a +1,79769313486232e308
VETTORE	Monodimensionale per tutti i tipi di variabile escluso BIT	
DELEGATI	Puntatori a FUNZIONI 32 bit	

ATTENZIONE:
SOLO IL SISTEMA NG35 UTILIZZA LE VARIABILI STATIC

6.9 Variabili Fixed

Le variabili di tipo FIXED sono variabili dichiarate ad un indirizzo fisso di memoria ram del controllo che, a differenza della normali variabili, non cambia modificando il programma. Queste semplificano notevolmente l'uso con un Host esterno. Infatti utilizzando le variabili FIXED non ci sarà bisogno di ricompilare l'applicazione HOST ad ogni modifica in VTB.
Le variabili FIXED sono sempre GLOBALI cioè visibili in tutte le pagine e tutti i task.

TIPO	DIMENSIONE	RANGE
BIT	1 bit	Da 0 a 1
CHAR	8 bit con segno	da –128 a +127
UCHAR	8 bit senza segno	da 0 a 255
INT	16 bit con segno	da –32.768 a +32.767
UINT	16 bit senza segno	da 0 a 65.535
LONG	32 bit con segno	da –2.147.483.648 a +2.147.483.647
FLOAT	64 bit (formato standard DOUBLE) IEEE 75	Da -1,79769313486232e308 a +1,79769313486232e308

L' indirizzo di START dell' area FIXED è il seguente:

NGM13-NGMEVO ***Addr = 536874496***
NG35 ***Addr = 1051648***
NGQ-NGQx ***Addr = 8389632***

6.10 Variabili Delegate

In questo tipo di variabile è memorizzato l'indirizzo di una funzione, una volta assegnato l'indirizzo si può richiamare la funzione attraverso l'istruzione ***call_delegate***. L'assegnazione dell'indirizzo della variabile deve essere eseguita prima dell'utilizzo del delegato. Solo dopo la variabile può essere usata per chiamare la funzione. Si può creare anche un array di variabili delegate e quindi richiamare una funzione in base ad un indice.
L' utilizzo dei DELEGATI risulta essere molto POTENTE poiché permette di accedere in modo molto veloce a delle funzioni senza dover utilizzare cicli condizionali.

ATTENZIONE: Le funzioni richiamate con CALL_DELEGATE devono essere VOID sia come argomenti sia come parametro di ritorno.
VTB NON EFFETTUA NESSUN CONTROLLO SULL' ESISTENZA DEL DELEGATO. Richiamare un delegato che non è stato inizializzato può mandare il sistema in CRASH

Esempio:
Variabili utilizzate:
var(2) as delegate

Init pagina del Main (inizializzazioni delegati):
Var(0)=fun1 *'assegna a var(0) l'indirizzo della funzione fun1*
Var(1)=fun2 *'assegna a var(1) 'indirizzo della funzione fun2*

Funzioni di pagina del Main (dichiarazione funzioni):
Function fun1() as void
.
Endfunction

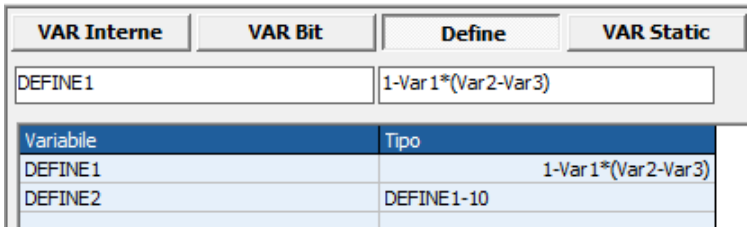
Function fun2() as void
.
Endfunction

Master Ciclo del Main (richiamo delle funzioni tramite delegati):
Call_delegate var(0) *'viene eseguita fun1*
Call_delegate var(1) *' viene eseguita fun2*

6.11 Define

Le DEFINE sono in pratica delle definizioni complesse. Queste si compongono da il NOME e il VALORE. Il nome identifica in modo univoco la define, il valore può contenere qualsiasi espressione alfanumerica. Il compilatore ogni volta che viene trovato il NOME della DEFINE questo viene sostituito dal suo VALORE. Queste servono per semplificare l' utilizzo di alcune espressioni complesse e possono essere combinate anche fra di se.

Dichiarazione di una DEFINE



Utilizzo di una DEFINE nel codice

Per utilizzare una DEFINE nel codice aggiunto è sufficiente richiamarla con il nome associato. Le DEFINE possono essere utilizzate in molte situazioni rendendo il programma più flessibile poiché è sufficiente cambiare il valore della DEFINE per ottenere una variazione immediata su tutto il progetto.

Esempio:

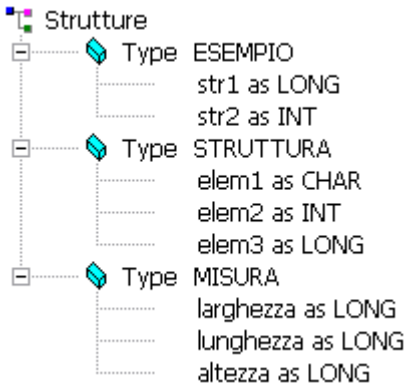
```
If Define1>=10000          ' Ciclo if DEFINE1
        ..............
        ..............
endif
```

6.12 Tabelle di testo

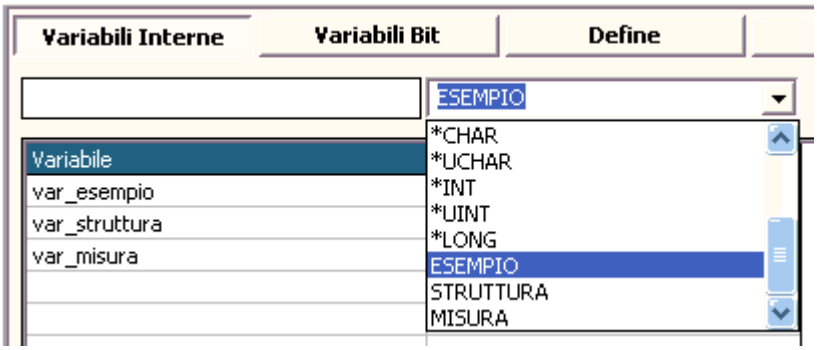
OBSOLETE

6.13 Strutture

Le strutture possono essere dichiarate solamente per le variabili interne. I campi definiti nelle strutture possono essere di qualsiasi tipo escluso BIT e puntatore.
Per dichiarare una struttura è necessario aprire la tabella delle strutture e definire, oltre al nome della struttura, ogni singolo elemento che vogliamo utilizzare al suo interno.



Una volta dichiarata la struttura, nell'elenco dei tipi disponibili apparirà il nome della struttura. E' possibile quindi definire una variabile appartenente ad uno dei tipi dichiarati come struttura.



Per accedere ad ogni elemento della struttura è necessario scrivere il nome della struttura seguito dal carattere **.** (punto) e dal campo della struttura a cui si vuole fare riferimento.
Esiste anche la possibilità di accedere alle strutture per mezzo dei puntatori (vedi capitolo PUNTATORI).

Esempio:
Variabili utilizzate:
val1 as long
val2 as long
val3 as long
misurazione as misura ***' dichiarazione della variabile di tipo struttura***

misurazione.larghezza=13
val1=misurazione.larghezza
misurazione.lunghezza=23
misurazione.altezza=54
val2=misurazione.lunghezza
val3=misurazione.altezza

7 OPERATORI

Essendo VTB un linguaggio completo, questo utilizza i comuni operatori logici e matematici.

7.1 Operatori Logici e Matematici

Di seguito vengono indicati tutti gli operatori logici matematici gestibili da VTB

OPERATORE	DESCRIZIONE	ESEMPIO
(	Parentesi aperta	Identifica l' inizio di un gruppo di calcoli oppure una funzione di sistema ***a=(c+b)/(x+y)***
+	Addizione	Addizione matematica ***a=b+c***
-	Sottrazione	Sottrazione matematica ***a=b-c***
*****	Moltiplicazione	Moltiplicazione matematica ***a=b*c***
/	Divisione	Divisione matematica ***a=b/c***
)	Parentesi chiusa	Identifica la fine di un gruppo di calcoli oppure una funzione di sistema ***a=(c+b)/(x+y)***
>	Maggiore	Condizione di maggiore ***if a>b***
<	Minore	Condizione di minore ***if a<b***
>=	Maggiore uguale	Condizione di maggiore o uguale ***if a>=b***
<=	Minore uguale	Condizione di minore o uguale ***if a<=b***
<>	Diverso	Condizione di diverso ***if a<>b***
=	Uguale	Condizione di uguale ***if a=b*** o assegnazione valore a variabile ***a=b***
||	OR logico	Condizione di OR logico ***if (a=b) || (b=c)*** la condizione è vera se una delle due espressioni o entrambi sono vere.
&&	AND logico	Condizione di AND logico ***if (a=b) && (b=c)*** la condizione è vera se entrambi le espressioni sono vere.
|	OR bit	Effettua l' OR dei bit indicati ***a=a|3*** Vengono settati i bit 1 e 2 della variabile ***a*** lasciando inalterati gli altri
&	AND bit	Effettua l' AND dei bit indicati ***a=a&3*** Vengono resettati tutti i bit della variabile ***a*** escluso il bit 1 e 2
!	Negazione espressione	Nega il contenuto dell' espressione i**f !(a=b)** L' espressione è vera se ***a*** è diverso da ***b***
~	Negazione bit	I bit della variabile vengono negati, cioè se sono allo stato logico 1 vengono portati allo stato logico zero e viceversa. **a=85 a=~a** Dopo la negazione la variabile ***a*** vale 170 **85 → 01010101** **170 → 10101010**
>>	Shift a destra dei bit	I bit della variabile vengono shiftati a destra n volte **a=8 a=a>>3** Dopo lo shift la variabile ***a*** vale 1
<<	Shift a sinistra dei bit	I bit della variabile vengono shiftati a sinistra n volte **a=1 a=a<<3** Dopo lo shift la variabile ***a*** vale 8

7.2 Note Su Espressioni matematiche

VTB gestisce le espressioni matematiche in modo completo. Tuttavia occorre prestare alcune attenzioni nei casi in cui nell'espressione siano utilizzate variabili di tipo INTERO o FLOAT.
In sostanza vengono seguite le regole sotto descritte:

1) Se nell'espressione viene utilizzata una variabile di tipo FLOAT, tutto il risultato viene convertito in FLOAT

2) Se il risultato dell'espressione, deve essere messo in una variabile FLOAT, occorre che per lo meno Un parametro dell'espressione sia di tipo FLOAT

Prendiamo ad esempio queste tre righe di codice:
A=10
B=4
R=A/B

A seconda di come vengono definite le variabili abbiamo i seguenti risultati:

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

Abilitando il livello di Warning sul compilatore, verranno generati degli avvisi in corrispondenza della possibilità di troncamento dei DATI.

8 FUNZIONI MATEMATICHE

VTB gestisce un ampio SET di funzioni matematiche.

8.1 SIN

Restituisce un valore FLOAT che specifica il seno di un angolo.
Hardware ***tutti***

Sintassi

Sin (***numero***) as float

L'argomento obbligatorio ***numero*** può essere un valore FLOAT o qualsiasi espressione numerica valida che esprime un **angolo in radianti**.

Osservazioni

La funzione **Sin** utilizza un angolo come argomento e restituisce il rapporto tra due lati di un triangolo rettangolo. Il rapporto è dato dalla lunghezza del lato opposto all'angolo divisa per la lunghezza dell'ipotenusa. Il risultato sarà compreso nell'intervallo tra -1 e 1. Per convertire i gradi in radianti, moltiplicarli per **PiGreco/180**. Per convertire i radianti in gradi, moltiplicarli per **180/PiGreco**

Esempio:
Variabili utilizzate:
angolo float
Cosec float
angolo = 1.3 ***' Definisce l'angolo in radianti.***
cosec = 1 / Sin (angolo) ***' Calcola la cosecante.***

8.2 COS

Restituisce un valore FLOAT che specifica il coseno di un angolo.
Hardware ***tutti***
Sintassi

Float Cos (***numero***) as float

L'argomento obbligatorio **numero** può essere un valore FLOAT o una qualsiasi espressione numerica valida che esprime un **angolo in radianti**.

Osservazioni
La funzione **Cos** accetta un angolo e restituisce il rapporto tra due lati di un triangolo rettangolo. Il rapporto è dato dalla lunghezza del lato adiacente all'angolo divisa per la lunghezza dell'ipotenusa. Il risultato sarà compreso nell'intervallo tra -1 e 1. Per convertire i gradi in radianti, moltiplicarli per **PiGreco/180**. Per convertire i radianti in gradi, moltiplicarli per **180/PiGreco**.

Esempio:
Variabili utilizzate:
angolo float
sec float
angolo = 1.3 ***' Definisce l'angolo in radianti.***
sec = 1 / Cos (angolo) ***' Calcola la secante.***

8.3 SQR

Restituisce un valore FLOAT che specifica la radice quadrata di un numero.
Hardware ***tutti***

Sintassi

Sqr (***numero***) as float

L'argomento obbligatorio ***numero*** può essere un valore FLOAT o qualsiasi espressione numerica valida maggiore o uguale a zero.
Esempio
Variabili utilizzate:
vsqr float
vsqr = sqr (4) ***' restituisce 2***

8.4 TAN

Restituisce un valore FLOAT che specifica la tangente di un angolo.
Hardware ***tutti***

Sintassi

Tan (***numero***) as float

L'argomento obbligatorio ***numero*** può essere un valore FLOAT o qualsiasi espressione numerica valida che esprime un **angolo in radianti**.

Osservazioni
La funzione **Tan** utilizza un angolo come argomento e restituisce il rapporto tra due lati di un triangolo rettangolo. Il rapporto è dato dalla lunghezza del lato opposto a un angolo divisa per la lunghezza del lato adiacente all'angolo. Per convertire i gradi in radianti, moltiplicarli per **PiGreco/180**. Per convertire i radianti in gradi moltiplicarli per **180/PiGreco**.

Esempio:
Variabili utilizzate:
angolo float
ctan float

angolo = 1.3 ***' Definisce l'angolo in radianti.***
ctan = 1 / Tan (angolo) ***' Calcola la cotangente.***

8.5 ATAN

Restituisce un valore di tipo FLOAT che specifica l'arcotangente di un numero.
Hardware ***tutti***

Sintassi

Atan (***numero***) as float

L'argomento obbligatorio ***numero*** può essere un valore FLOAT o una qualsiasi espressione numerica valida.

Osservazioni

La funzione **Atan** utilizza come argomento numero il rapporto tra due lati di un triangolo rettangolo e restituisce l'angolo corrispondente espresso in radianti.
Il rapporto è dato dalla lunghezza del lato opposto all'angolo divisa per la lunghezza del lato adiacente all'angolo. Il risultato viene espresso in radianti nell'intervallo da -PiGreco/2 a PiGreco/2. Per convertire i gradi in radianti, moltiplicarli per **PiGreco/180**. Per convertire i radianti in gradi moltiplicarli per **180/PiGreco**.

Nota

Atan *è la funzione trigonometrica inversa di* ***Tan****, che utilizza l'angolo come argomento e restituisce il rapporto tra due lati di un triangolo rettangolo.*
Atan *non deve essere confusa con la cotangente, che è semplicemente l'inverso di una tangente (1/tangente).*

8.6 ASIN

Restituisce un valore FLOAT che specifica il seno inverso di un angolo.
Hardware ***tutti***

Sintassi

Asin (***numero***) as float

L'argomento obbligatorio ***numero*** può essere un valore FLOAT o qualsiasi espressione numerica valida.

Esempio

Variabili utilizzate:
angolo float
var float

```
angolo = 1.3        ' Definisce l'angolo in radianti.
var = asin (angolo)' Calcola il seno inverso.
```

8.7 ACOS

Restituisce un valore FLOAT che specifica il coseno inverso di un angolo.
Hardware ***tutti***

Sintassi

Acos (***numero***) as float

L'argomento obbligatorio ***numero*** può essere un valore FLOAT o una qualsiasi espressione numerica valida.

Esempio

Variabili utilizzate:
angolo float
var float

```
angolo = 1.3        ' Definisce l'angolo in radianti.
var = acos (angolo)' Calcola il coseno inverso.
```

8.8 ATAN2

Restituisce un FLOAT che è l'angolo la cui tangente è il quoziente di due numeri specificati.
Hardware *tutti*

Sintassi

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

Gli argomenti obbligatori **y** e **x** sono di tipo FLOAT.

Valore restituito

Angolo θ, espresso in radianti, tale che -π≤θ≤π e tan(θ) = y / x, dove (x, y) è un punto del piano cartesiano. Osservare quanto segue:

- Per (x, y) nel quadrante 1, 0 < θ < π/2.
- Per (x, y) nel quadrante 2, π/2 < θ≤π.
- Per (x, y) nel quadrante 3, -π < θ < -π/2.
- Per (x, y) nel quadrante 4, -π/2 < θ < 0.

Per i punti sui limiti dei quadranti, il valore restituito è il seguente:

- Se y è 0 e x non è negativo, θ = 0.
- Se y è 0 e x è negativo, θ = π.
- Se y è positivo e x è 0, θ = π/2.
- Se y è negativo e x è 0, θ = -π/2.

Esempio

Variabili utilizzate:

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

PI= 3.141592 ' PI Greco
x=1.0
y=2.0
angle = 30
radians = angle * (PI/180)
result = Tan(radians) ' Calcola la tangente di 30 gradi
radians = Atan(result) ' Calcola Arcotangente della precedente tangente
angle = radians * (180/PI)
radians = Atan2(y, x) ' Calcola l' arcotangente
angle = radians * (180/PI);
```

8.9 ABS

Restituisce il valore assoluto di tipo INTERO
Hardware *tutti*

Sintassi

Abs (***numero***) as long

L'argomento obbligatorio ***numero*** può essere un valore LONG o una qualsiasi espressione numerica

Esempio

Variabili utilizzate:

```
Num long

Num = -3250
Num = Abs(Num)' restituisce 3250
```

8.10 FABS

Restituisce il valore assoluto di tipo FLOAT
Hardware ***tutti***

Sintassi

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

L'argomento obbligatorio ***numero*** può essere un valore FLOAT o una qualsiasi espressione numerica

Esempio
Variabili utilizzate:
Num float

Num = -3.250
Num = Abs(Num)' restituisce 3.250

9 ISTRUZIONI PER IL CONTROLLO DEL FLUSSO DEL PROGRAMMA

Come in tutti i linguaggi di programmazione anche in VTB sono presenti un ampio SET di istruzioni per il controllo del flusso di programma. **QUESTE SONO PRESENTI PER TUTTE LE PIATTAFORME HARDWARE**

9.1 IF-ELSE-ENDIF

Consente l'esecuzione condizionale di un gruppo di istruzioni in base al valore di un'espressione.

Sintassi

```
if condizione
        [istruzioni]
else
        [istruzionielse]
endif
```

La sintassi dell'istruzione **if... else** è composta dalle seguenti parti

condizione	Obbligatoria. Qualsiasi espressione numerica che può dare come risultato True (valore diverso da zero) o False (valore zero).
istruzioni	Elenco istruzioni da eseguire se la condizione **IF** è vera
istruzionielse	Facoltativa. Elenco istruzioni da eseguire se la condizione **IF** è FALSA.
endif	Fine del ciclo **IF ELSE**

Osservazioni
L'istruzione **Select Case** può risultare più utile per la valutazione di un'espressione poiché crea del codice più leggibile.

Esempio
Variabili utilizzate:
var1 int
var2 int

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

9.2 LABEL

Identifica un punto di salto per **GOSUB** o **GOTO.**

Sintassi

Label ***nomelabel***

Nomelabel nome alfanumerico che identifica la LABEL.
Non possono esistere nomi di LABEL uguali per ogni PAGINA o nel ciclo MAIN. Le label inserite nella SUB DI PAGINA MAIN sono visibili da tutte le PAGINE

ATTENZIONE: L'istruzione LABEL risulta OBSOLETA in quanto l' utilizzo delle FUNZIONI rende più pratico il controllo dei salti.

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

9.3 GOSUB-RETURN

Consente di passare il controllo ad una SOUBRUTINE e di ritornare alla successiva istruzione di programma.

Sintassi

GoSub nomelabel

L'argomento ***nomelabel*** può essere una qualsiasi LABEL di programma all'interno della pagina o del MAIN.

Osservazioni

GoSub e **Return** possono essere utilizzate in un punto qualsiasi del codice, ma devono essere entrambe incluse nella stessa PAGINA o nel MAIN. Una subroutine può includere più istruzioni **Return**, ma la prima istruzione **Return** incontrata comporterà il ritorno del programma all'istruzione che segue l'istruzione **GoSub** più recente.

ATTENZIONE: L' istruzione GOSUB risulta OBSOLETA in quanto l'utilizzo delle FUNZIONI rende più pratico il controllo dei salti

Esempio

```
if condizione
        gosub label1
else
        gosub label2
endif

Label Label1
.
Return
Label Label2
.
Return
```

9.4 GOTO

Consente di saltare ad una LABEL di programma.

Sintassi

Goto *nomelabel*

L'argomento ***nomelabel*** può essere una qualsiasi LABEL di programma all' interno della pagina o del MAIN.

Osservazioni

Goto passa il controllo ad un punto di programma identificato con una LABEL. A differenza di GOSUB non è necessario nessun **RETURN**

ATTENZIONE: L'istruzione GOTO risulta OBSOLETA in quanto l'utilizzo delle FUNZIONI rende più pratico il controllo dei salti

Esempio

```
if condizione
        goto label1
else
        goto label2
endif

Label Label1
.
Label Label2
.
```

9.5 INC

Incrementa una variabile di qualsiasi tipo

Sintassi

Inc *nomevar*

L'argomento ***nomevar*** può essere una qualsiasi variabile dichiarata nel programma.

Osservazioni

Inc equivale perfettamente all' istruzione **VAR=VAR+1** solamente che viene eseguita in modo più veloce.

Esempio

```
INC var1' Incrementa var1
```

9.6 DEC

Decrementa una variabile di qualsiasi tipo

Sintassi

Dec *nomevar*

L'argomento ***nomevar*** può essere una qualsiasi variabile dichiarata nel programma.

Osservazioni

Dec equivale perfettamente all' istruzione **VAR=VAR-1** solamente che viene eseguita in modo più veloce.

Esempio

```
DEC var1        ' Decrementa var1
```

9.7 SELECT-CASE-ENDSELECT

Esegue uno dei vari blocchi di istruzioni sulla base del valore di un'espressione.

Sintassi

```
Select espressione
    [Case elencoespressioni-n
        [istruzioni-n]] ...
    [Case Else
        [istruzionielse]]
EndSelect
```

La sintassi dell'istruzione **Select Case** è composta dalle seguenti parti:

espressione	Obbligatoria. Qualsiasi espressione numerica.
Elencoespressioni-n	Obbligatoria. Elenco delimitato in una delle seguenti forme: **espressione**, espressione **To** espressione**,** . La parola chiave **To** specifica un intervallo di valori. Se si utilizza la parola chiave **To** il valore minore deve apparire prima di **To**. Per indicare una serie di valore utilizzare espressione**,**.
Istruzioni-n	Facoltativa. Una o più istruzioni eseguite se espressione corrisponde a una qualsiasi parte di **elencoespressioni-n.**
Istruzionielse	Facoltativa. Una o più istruzioni eseguite se **espressione** non corrisponde a nessun elemento della proposizione **Case**.

Osservazioni

Se **espressione** corrisponde all'espressione **elencoespressioni**, associata a una proposizione **Case**, le istruzioni che seguono tale proposizione **Case** verranno eseguite fino alla proposizione **Case** successiva.
L'ultimo blocco di istruzioni verrà eseguito fino a **EndSelect**. Il controllo passerà quindi all'istruzione successiva a **EndSelect**.
Se **espressione** corrisponde a un'espressione elencoespressioni in più di una proposizione **Case**, verranno eseguite solo le istruzioni che seguono la prima corrispondenza. La proposizione **Case Else** viene utilizzata per indicare le **istruzionielse** da eseguire se non viene trovata corrispondenza tra **espressione** e **elencoespressioni** in una delle altre selezioni **Case**. Sebbene non sia obbligatorio, è consigliabile includere un'istruzione **Case Else** in un blocco **Select** per gestire valori di **espressione** non previsti.
Se nessuna **elencoespressioni Case** corrisponde a **espressione** e non è stata specificata un'istruzione **Case Else**, verrà eseguita l'istruzione successiva a **EndSelect**.
In ciascuna proposizione **Case**, non è possibile utilizzare più espressioni o intervalli. Ad esempio la riga: **Case** 1 **To** 4, 7 **To** 9 **NON È VALIDA.** Le istruzioni **Select Case** possono essere nidificate. A ciascuna istruzione **Select Case** nidificata deve corrispondere un'istruzione **EndSelect**.

Esempio

Variabili utilizzate:
var1 int
var2 int
var3 int

```
Select var1
    case 10          ' se var1=10
        ...
    case var2+var3   ' se var1=var2+var3
        ...
    case 5 TO 20             ' se var1 è compresa tra 5 e 20
        ...
    case 1,6,8               ' se var1=1 o var1=6 o var1=8
        ...
    case else                ' altri valori
        ...
Endselect
```

9.8 FOR-NEXT-STEP-EXITFOR

Ripete un gruppo di istruzioni per il numero di volte specificato. Il funzionamento è un mix tra linguaggio basic e C.

Sintassi

```
For contatore = inizio To fine [Step incremento]
        [istruzioni]
Next [contatore]
```

La sintassi dell'istruzione **For**...**Next** è composta dalle seguenti parti:

contatore	Obbligatoria. Variabile numerica utilizzata come contatore di ciclo. Escluso BIT
Inizio	Obbligatoria. Valore iniziale della variabile contatore.
Fine	Obbligatoria. Valore finale del contatore. Può essere qualsiasi condizione formata anche da espressioni sulla variabile contatore. Occorre sempre indicare il tipo di condizione da effettuare sulla variabile contatore (**> < <> =**)
incremento	Facoltativa. Quantità di cui viene incrementato il contatore al compimento di ciascun ciclo. Se non viene specificato, **incremento** assume per impostazione predefinita il valore 1. Può essere qualsiasi espressione numerica.
Istruzioni	Facoltativa. Una o più istruzioni comprese tra For e Next da eseguire

Osservazioni

L'argomento incremento può essere sia positivo che negativo. Il valore dell'argomento incremento determina l'esecuzione del ciclo come di seguito indicato:

L' istruzione **NEXT** incrementa il contatore in base a quanto indicato in **STEP**. È possibile inserire un qualsiasi numero di istruzioni **ExitFor** nel ciclo, come metodo d'uscita alternativo. **Exit For** viene spesso utilizzata dopo la valutazione di una condizione, ad esempio **If**..., per trasferire il controllo all'istruzione immediatamente successiva all'istruzione **Next**. È possibile nidificare cicli **For...Next** inserendo un ciclo **For...Next** all'interno di un altro. Assegnare come contatore a ciascun ciclo un nome di variabile univoco. La seguente costruzione è corretta:

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

Esempio

Variabili utilizzate:

```
Var1 int
Var2 int
Var3 int
Var4 int
For var1=0 to var1<8        ' Ripete 8 volte
        ...
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
```

9.9 WHILE-LOOP-EXITWHILE

Esegue una serie di istruzioni finché la valutazione di una determinata condizione è vera.

Sintassi

```
While condizione
        [istruzioni]
loop
```

La sintassi dell'istruzione **While...loop** è composta dalle seguenti parti:

condizione	Obbligatoria. Espressione numerica che può dare come risultato True (valore diverso da zero) o False (valore zero).
Istruzioni	Facoltativa. Una o più istruzioni eseguite finché la condizione è True.

Osservazioni

Se condizione è True, verranno eseguite tutte le istruzioni fino all'istruzione **loop**. Il controllo tornerà quindi all'istruzione **While** e condizione verrà analizzata nuovamente. Se condizione è ancora True, il processo verrà ripetuto. Se non è True, l'esecuzione riprenderà con l'istruzione successiva all'istruzione **loop**.
I cicli **While...loop** possono essere nidificati a qualsiasi livello. Ciascuna istruzione **loop** corrisponderà all'istruzione **While** più recente. Per uscire da un ciclo **while..loop**, oltre ad utilizzare l' istruzione **GOTO** LABEL (con label al di fuori del ciclo), può essere utilizzata l'istruzione **EXITWHILE** che porta il controllo del programma all'istruzione successiva il **LOOP**.

Esempio

Variabili utilizzate:
Var1 int

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

10 FUNZIONI

VTB gestisce le funzioni con la stessa sintassi di VISUAL BASIC. Esiste solamente una limitazione per quanto riguarda la dichiarazione delle variabili interne alla funzione. **Queste non possono essere VETTORI, STRUTTURE o BIT.**

10.1 Dichiarazione di una funzione

Sintassi

```
function nome_funzione (par1 as int, par2 as char, ….., par n as *long) as tipo_di_funzione
        dim var as int
        ....
        ....        'corpo della funzione
        ....
        nome_funzione = parametro di ritorno
endfunction
```

La sintassi di una funzione è composta dalle seguenti parti:

function	Obbligatoria. Parola chiave che identifica l'inizio della funzione.
nome_funzione	Obbligatoria. Nome che identifica univocamente la funzione, scelto dall'utente.
lista dei parametri	Facoltativa. Indica i parametri da passare alla funzione. Se la funzione è VOID non inserire nessun parametro.
tipo_di_funzione	Obbligatorio. Definisce il tipo di dato che ritorna dalla funzione. Se non ritorna alcun valore scrivere **as void.**
variabili locali	Facoltative. Variabili locali, vengono allocate alla chiamata della funzione e distrutte al temine della funzione stessa. **NON possono essere utilizzate variabili di tipo struttura, array o bit.**
corpo della funzione	Facoltativo. Sequenza di istruzioni eseguite dalla funzione.
nome_funzione=…	Facoltativo. Assegna il valore che deve ritornare dalla funzione.
endfunction	Obbligatorio. Parola chiave che indica il termine della funzione.

Osservazioni

Un funzione può essere richiamata semplicemente scrivendo l'identificatore che ne descrive il nome e passando alla funzione gli eventuali parametri presenti.
Per uscire dalla funzione in un punto qualsiasi può essere anche utilizzata l'istruzione **return**.
L'assegnazione **nome_funzione = ….** non provoca l'uscita dalla funzione ma solo l'attribuzione del valore di ritorno della funzione.

Esempio

Variabili utilizzate:

```
valore_medio as int
numero_a as int
numero_b as int

function media_intera(numero_1 as int, numero_2 as int) as int
        dim media_temp as int
        media_temp=(numero_1+numero_2)/2
        media_intera=media_temp
endfunction

'All'interno del progetto Vtb posso chiamare la funzione

numero_a=13
numero_b=33
valore_medio=media_intera(numero_a,numero_b)
```

10.2 Dichiarazione di variabili interne alle funzioni

Sintassi

Dim nomevar as tipo

La sintassi dell'istruzione **DIM** è composta dalle seguenti parti:

Nomevar	Obbligatoria. Nome della variabile
tipo	Obbligatoria. Tipo della variabile escluse: **STRUTTURE, ARRAY e BIT**

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

11 FUNZIONI DI SISTEMA

VTB mette a disposizione molte funzioni di LIBRERIA (API) per una gestione completa dei sistemi.
Queste funzioni possono essere dipendenti dall'hardware in uso.

11.1 FUNZIONI API PER IL CONTROLLO DELLA LINEA RS232

Le piattaforme Promax hanno normalmente 1 o 2 Canali seriali disponibili all'applicazione.
Esistono già oggetti che incorporano dei protocolli standard definiti, ad esempio il MODBUS sia MASTER che SLAVE. Tuttavia è possibile utilizzare il canale seriale in modo PROPRIETARIO.
Per far ciò vengono messe a disposizione delle API apposite.
Queste API fanno sempre riferimento al SECONDO CANALE SERIALE della PIATTAFORMA in USO.

11.1.1 SER_SETBAUD

Programma il BaudRate del secondo CANALE SERIALE
Hardware ***Tutti***

Sintassi

SER_SETBAUD (long Baud)

Parametri

Baud Valore del Baud Rate. Questo deve corrispondere ad un valore STANDARD:
1200-2400-4800-9600-19200-38400-57600-115200

11.1.2 SER_MODE

Programma la modalità del secondo CANALE SERIALE. Nel caso questa funzione non sia richiamata, di default la seriale di viene programmata con: No parity, 8 bit a carattere, 1 bit di stop.
Hardware ***Tutti***

Sintassi

SER_MODE(char par, char nbit, char nstop)

Parametri

par Parity (0=no parity, 1=odd parity, 2=even parity)
nbit Numero bit a carattere (7 o 8)
nstop Numero bit di stop (1 o 2)

Esempio

```
ser_mode(1,8,2) ' Programma la 2a seriale con ODD-PARITY, 8 BIT/CHAR 2 STOP-BIT
```

11.1.3 SER_GETCHAR

Ritorna un carattere presente nel buffer della linea seriale. Il sistema operativo si occupa della gestione del buffer di ricezione.
Hardware ***Tutti***

Sintassi

SER_GETCHAR () as int

Valore di ritorno

int ***-1*** Nessun carattere presente nel buffer
>=0 Codice del carattere recuperato dal buffer

11.1.4 SER_PUTCHAR

Invia un carattere sulla linea seriale. Il sistema operativo si occupa della gestione del buffer di trasmissione.
Hardware ***Tutti***

Sintassi

SER_PUTCHAR (int Car)

Parametri
Car Codice del Carattere da inviare

11.1.5 SER_PUTS

Invia una stringa di caratteri sulla linea seriale. La stringa deve terminare con il carattere 0 (NULL).
Hardware ***Tutti***
ATTENZIONE: Questa funzione non è adatta ad una trasmissione BINARIA dei caratteri ma solo ad una trasmissione ASCII.

Sintassi

SER_PUTS (char *str)

Parametri
***str** Puntatore alla stringa

Esempio

```
Ser_puts("PROVA TESTO")        ' trasmette la stringa PROVA TESTO
Strcpy(Vect(),"PROVA TESTO")   ' Copia la stringa PROVA TESTO in Vect
Ser_puts(Vect())               ' trasmette la stringa PROVA TESTO
```

11.1.6 SER_PRINTL

Stampa formattata di una variabile intera.
Hardware ***Tutti***

Sintassi

SER_PRINTL (const char *format, long val)

Parametri
Format Costante stringa che indica il formato da stampare
Val Qualsiasi valore, espressione o variabile intera

Formati disponibili

######	Indica il numero di caratteri da stampare	23456
###.###	Stampa con il punto decimale nella posizione inserita	123.456
+####	Stampa sempre con segno	+1234
#0.##	Forza uno ZERO nel punto inserito	0.12
X##	Stampa in formato ESADECIMALE	F1A3
B##	Stampa in formato BINARIO	10110011

Esempio

```
var=12345
ser_printl("###.##",var) ' Sarà stampato: "123.45"
var=2
ser_printl("###.##",var) ' Sarà stampato: "  . 2"
ser_printl("###.00",var) ' Sarà stampato: "  .02"
ser_printl("##0.00",var) ' Sarà stampato: "  0.02"
```

11.1.7 SER_PRINTF

Stampa formattata di una variabile float. E' come la precedente ma lavora con dei valori FLOAT.
Hardware ***Tutti***

Sintassi

SER_PRINTF (const char *format, float val)

Parametri

Format	Costante stringa che indica il formato da stampare
Val	Qualsiasi valore, espressione o variabile di tipo float

11.1.8 SER_PUTBLK

Invia un blocco di caratteri con lunghezza specificata. Rispetto alla funzione ***ser_puts*** permette di inviare anche il codice 0, inoltre avvia la trasmissione in background gestendo anche il segnale RTS utilizzato per abilitare il buffer RS485.
Hardware ***Tutti***

ATTENZIONE: Questa funzione è adatta ad una trasmissione BINARIA dei caratteri ed è l'unica che consente la trasmissione con RS485.

Sintassi

SER_PUTBLK (char *Buffer, int Len)

Parametri

***Buffer**	Puntatore al buffer da trasmettere
Len	Lunghezza dei Byte da trasmettere

Esempio

```
Ser_putblk(Vect(),11)          ' trasmette 11 caratteri dell'array vect
```

11.1.9 SER_PUTST

Ritorna lo stato della trasmissione in background avviata con ***ser_putblk.***
Hardware ***Tutti***

Sintassi

SER_PUTST () as int

Valore di ritorno

uint	***-1***	Errore di trasmissione
	>=0	Numero di caratteri ancora da inviare

Esempio

```
Ser_putblk(Vect(),11)      ' Invia 11 caratteri
while Ser_putst()          ' Attende che siano tutti trasmessi
loop
```

11.2 FUNZIONI API DI UTILIZZO GENERICO

Funzioni di utilizzo generico.

11.2.1 PAGINA

Carica la pagina indicata da page. Le pagine vengono numerate partendo da 1. La funzione PAGINA carica il CODICE DI PAGINA distruggendo quello della pagina precedente. Ad ogni cambiamento di pagina viene eseguito il codice in "INIT DI PAGINA"

Hardware ***Tutti***

Sintassi

PAGINA (int Page)

Parametri

Page Numero di pagina da caricare

11.2.2 GET_TIMER

Lettura del timer in unità di TASK PLC (scan time).

Hardware ***Tutti***

Sintassi

GET_TIMER () as long

Valore di ritorno

long Valore del timer di sistema in unità di campionamento

Alcune DEFINE sono automaticamente generate da VTB per adattare l'applicazione al tempo di scansione del TASK PLC:

TAU Tempo di scansione del TASK PLC in millisecondi (valore INTERO)
TAUFLOAT Tempo di scansione del TASK PLC in millisecondi (valore FLOAT)
TAUMICRO Tempo di scansione del TASK PLC in 0.1 millisecondi

Esempio

```
Tick long
Tick=Get_timer()                    ' Prendi il valore iniziale del timer
while Test_timer(Tick,1000/TAU)     ' Attendi per 1 secondo
Loop
```

11.2.3 TEST_TIMER

Test se trascorso un certo tempo. Ritorna 1 se il tempo è trascorso altrimenti 0.

Hardware ***Tutti***

Sintassi

TEST_TIMER (long Timer, long Tempo) as char

Parametri

Timer Valore iniziale del timer
Tempo Tempo di controllo

Valore di ritorno

char ***1*** Tempo trascorso
0 Tempo non trascorso

Esempio

```
Tick long
Tick=Get_timer()                    ' Prendi il valore iniziale del timer
while Test_timer(Tick,1000/TAU)     ' Attendi per 1 secondo
Loop
```

11.2.4 ALLOC

Alloca dinamicamente una area di memoria RAM
Hardware ***NG35***

Sintassi

ALLOC (Long Mem) as *char

Parametri

Mem		Quantità di memoria da allocare in byte

Valore di ritorno

*char	<>0	Puntatore alla memoria allocata
	0	Errore allocazione

Esempio

```
Punt As *Char
N as Long
Punt=Alloc(3000) ' Alloca 3000 byte di memoria
FOR N=0 to N<3000
        PUNT[N]=N
NEXT N
```

11.2.5 FREE

Libera un area di memoria precedentemente allocata con ***alloc***.
Hardware ***NG35***

Sintassi

FREE (Char *Punt)

Parametri

Punt	Puntatore alla memoria da liberare

Esempio

```
Pnt As *Char

Pnt=Alloc(3000)  ' Alloca 3000 byte di memoria
....
....
Free(pnt)                ' Libera la memoria
```

11.2.6 SYSTEM_RESET

Esegue un RESET software dell'hardware.
Hardware ***Tutti***

Sintassi

SYSTEM_RESET (Char mode)

Parametri

mode	**=0**	Esegue un normale RESET avviando l'applicazione
	=1	Esegue il RESET mettendo in BOOT il dispositivo

11.3 FUNZIONI API PER TRATTAMENTO DELLE STRINGHE

VTB non utilizza variabili STRINGA ma vengono trattate con delle apposite funzioni simili a quelle del **"C"** STANDARD.

11.3.1 STRCPY

Copia la stringa puntata da SOURCE in quella puntata da DEST. La stringa deve terminare con il carattere 0 (NULL).
Hardware ***Tutti***

Sintassi

STRCPY (Char *Dest, Char *Source)

Parametri
Dest Puntatore alla destinazione
Source Puntatore alla sorgente

Esempio
Variabili utilizzate:
Dest(10) char
Dest1(10) char
strcpy(Dest(),"prova testo") ***' copia la stringa "prova testo" nel vettore dest***
strcpy(Dest1(),Dest()) **'copia la stringa "prova testo" nel vettore dest1**

11.3.2 STRLEN

Ritorna la lunghezza della stringa.
Hardware ***Tutti***

Sintassi

STRLEN(Char *Str) as int

Parametri
Str Puntatore alla Stringa

Valore di ritorno
int Lunghezza della stringa

Esempio
Variabili utilizzate:
Len int
Len=StrLen("prova testo") ***' ritorna il valore 11***

11.3.3 STRCMP

Confronta due stringhe.
Hardware ***Tutti***

Sintassi

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

Parametri
Str1 Puntatore alla Prima Stringa
Str2 Puntatore alla Seconda Stringa

Valore di ritorno
char ***0*** Stringhe uguali
< Stringa Str1 minore di Str2
>0 Stringa Str1 maggiore di Str2

11.3.4 STRCAT

Concatena due stringhe.
Hardware ***Tutti***

Sintassi

STRCMP(Char *Dest,Char *Source)

Parametri

Dest	Puntatore alla Stringa di destinazione
Source	Puntatore alla Stringa da concatenare

Esempio
Variabili utilizzate:
Str(30) Char
Strcpy(Str(),"PROVA ")
StrCat(Str(),"TESTO")

11.3.5 STR_PRINTL

Converte una variabile INTERA in una STRINGA di caratteri.
Hardware ***Tutti***

Sintassi

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

Parametri

Dest	Puntatore alla Stringa di destinazione
Format	Formato (vedi PRINTL)
Var	Variabile Intera

Formati disponibili

######	Indica il numero di caratteri da stampare	23456
###.###	Stampa con il punto decimale nella posizione inserita	123.456
+####	Stampa sempre con segno	+1234
#0.##	Forza uno ZERO nel punto inserito	0.12
X##	Stampa in formato ESADECIMALE	F1A3
B##	Stampa in formato BINARIO	10110011

Per i formati vedi esempio della funzione ***print.***

11.3.6 STR_PRINTF

Converte una variabile FLOAT in una STRINGA di caratteri.
Hardware ***Tutti***

Sintassi

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

Parametri

Dest	Puntatore alla Stringa di destinazione
Format	Formato (vedi PRINTL)
Var	Variabile Float

Per i formati vedi esempio della funzione ***print.***

11.4 FUNZIONI DI INTERPOLAZIONE ASSI

Le funzioni di interpolazione ASSI si possono trovare a livello di API, ma anche sotto forma di OGGETTO nella CLASSE COBJINTERPOLA. Queste ultime sono sicuramente più aggiornate e quindi da utilizzare in modo preferenziale. Di seguito vengono descritte le funzioni GENERICHE interne all'oggetto. Ricordarsi di mettere sempre il prefisso del NOME_OGGETTO. Se per esempio l'oggetto si chiama ***obj*** la funzione ***moveto*** dovrà essere richiamata con ***obj.moveto.***

11.4.1 PROPRIETA'

Queste sono le proprietà di uso comune dell'oggetto COBJINTERPOLA.

N.assi Numero di assi da interpolare. Si può cambiare solo a livello di ambiente VTB. Viene automaticamente definita una **DEFINE** di nome **nomeobj**.***NASSI*** con questo valore.

N.tratti Numero di tratti del buffer di movimentazione. Si può cambiare solo a livello di ambiente VTB e **deve assumere il valore di una potenza di 2 (4, 8, 16, ecc.)**. Viene automaticamente definita una **DEFINE** di nome **nomeobj**.***NTRATTI*** con questo valore.

.vper Valore per il cambio di velocità al volo. Insieme a ***Div.vper*** forma un rapporto, quando vale 1 la velocità corrisponde a quella impostata.

Div.vper Divisore di ***vper***. Si può cambiare solo a livello di ambiente VTB.

Abilita arcto Normalmente viene impostato ad 1, se 0 le funzioni di interpolazione circolare non saranno disponibili. Utilizzato per diminuire la lunghezza del codice. Si può cambiare solo a livello di ambiente VTB.

.acc Accelerazione e decelerazione. Durante le rampe ad ogni campionamento (TASK PLC) la velocità espressa in unità/campionamento viene incrementata (o decrementata) di questo valore. Valore di default 10.

.sglr Soglia tolleranza errore sul raggio. Valore di default 10.

.sglp Soglia spigolo 2D. Utilizzata da moveto e lineto per il calcolo dello spigolo sul piano di lavoro. E' espresso in decimi di grado. Valore di default 200 (20 gradi).

.sgl3d(NASSI) Soglia spigolo 3D. Valore di default 0.2 (per tutti gli assi).

.pc(NASSI) Valore attuale delle quote interpolate.

.cmd Uscita dell'asse virtuale gestito da ***setcmd***.

11.4.2 MOVETO

Movimentazione con interpolazione lineare degli assi indicati. L' interpolazione viene eseguita alla velocità **vel**. Il parametro **ferma** definisce se gli assi devono fermarsi nel punto finale o continuare al movimento successivo. Questo prevede che ci siano più movimenti nel BUFFER.

Hardware ***Tutti***

Sintassi

.MOVETO(Long Vel, Char Ferma, Long *PuntAssi) as char

Parametri

Vel Velocità di interpolazione espressa nell'unità programmate dai parametri macchina

Ferma Flag per controllo se effettuare una fermata prima del tratto successivo

Ferma=0 nessuna fermata
Ferma=1 fermata prima del tratto successivo
Ferma=2 fermata su spigolo Tridimensionale (Soglia presa da ***sgl3d***)
Ferma=3 fermata su spigolo Bidimensionale (Soglia presa da ***sglp***)

PuntAssi Puntatore al vettore quote finale degli assi espresse nelle unità programmate

Valore di ritorno

Char ***0*** Non inserito nel buffer (buffer movimenti pieno)
1 Tratto inserito nel buffer

Note

Moveto viene generalmente utilizzata per interpolazioni su tutti gli ASSI.
Il vettore di velocità viene scomposto su tutti gli assi da interpolare. Se viene utilizzato lo spigolo Tridimensionale, FERMA=2, il CN controlla se fermarsi sul tratto in base al valore calcolato.

Con FERMA=3 lo spigolo viene calcolato solamente sugli assi del piano di LAVORO impostato in base al parametro SGLP espresso in decimi di grado. Se il tratto non viene inserito nel BUFFER, occorre attendere lo svuotamento di questo, altrimenti il tratto verrà perso.

Valori approssimativi di riferimento per SGL3D

SOGLIA SPIGOLO IN GRADI	VALORE SGL3D (min-max)
5	60-90
10	125-175
20	250-350
30	300-500
45	400-700

Esempio (nome oggetto = OBJ)
Variabili utilizzate:

```
VectAssi(4) long
Vel long
Test char
'*****************************************************************************
' interpolazione veloce di più segmenti di retta sugli assi X,Y tenendo fermi Z 'ed A effettuati alla velocità vel con fermata su spigolo 2D
'*****************************************************************************
vel=1000
VectAssi(0)=1000 'X
VectAssi(1)=2000 'Y
VectAssi(2)=OBJ.pc(2) 'Z rimane fermo
VectAssi(3)=OBJ.pc(3) 'A rimane fermo
muovi()
VectAssi(0)=4000 'X
VectAssi(1)=6000 'Y
VectAssi(2)=OBJ.pc(2) 'Z rimane fermo
VectAssi(3)=OBJ.pc(3) 'A rimane fermo
muovi()
VectAssi(0)=5000 'X
VectAssi(1)=2000 'Y
VectAssi(2)=OBJ.pc(2) 'Z rimane fermo
VectAssi(3)=OBJ.pc(3) 'A rimane fermo
muovi()
.
.
' *******************************************************
' Funzione per movimentazione, attende che il buffer sia vuoto
' *******************************************************
Function muovi() as Void
Dim test as Char
Label Move
test=Obj.moveto(vel,3,VectAssi())
if test=0
        goto Move        attesa se BUFFER pieno
endif
EndFunction
```

11.4.3 LINETO

Lineto interpola gli assi scomponendo la velocità SOLO SUGLI ASSI DEL PIANO DI LAVORO IMPOSTATO.

Pertanto gli altri assi vengono TRASPORTATI.
Questa funzione risulta utile per la gestione di ASSI TANGENZIALI, esempio plotter da TAGLIO dove la LAMA deve essere TRASPORTATA per avere una migliore FLUIDITA' di movimento. La fermata degli ASSI viene calcolata in BASE alla soglia spigolo ***sglp***. Se lo spigolo formato è minore o uguale a questa SOGLIA, gli assi non si fermano sul PUNTO, ma raccordano il tratto.
Hardware ***Tutti***

Sintassi

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

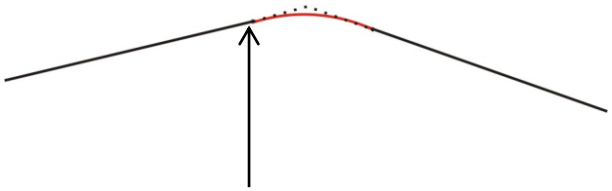
Parametri

Vel	Velocità di interpolazione espressa nell'unità programmate dai parametri macchina
PuntAssi	Puntatore al vettore quote finale degli assi espresse nelle unità programmate

Valore di ritorno

Char	***0***	Non inserito nel buffer (buffer movimenti pieno)
	1	Tratto inserito nel buffer

Note
Lineto, a differenza di Moveto, non scompone la velocità degli assi su tutti quelli attivi, ma solo su quelli del piano di lavoro selezionato. Pertanto tale funzione non è IDONEA per interpolazioni TRIDIMENSIONALI.



Se lo spigolo è minore o uguale a SGLP gli assi non si fermano

Esempio (nome oggetto = OBJ)
Variabili utilizzate:

```
VectAssi(4) long
Vel long
Test char
' interpolazione veloce di più segmenti con terzo asse trasportato
vel=1000
VectAssi(0)=1000 'X
VectAssi(1)=2000 'Y
VectAssi(2)=100 'Z trasportato
VectAssi(3)=OBJ.pc(3) 'A rimane fermo
muovi()
VectAssi(0)=4000 'X
VectAssi(1)=6000 'Y
VectAssi(2)=200 'Z trasportato
VectAssi(3)=OBJ.pc(3) 'A rimane fermo
muovi()
VectAssi(0)=5000 'X
VectAssi(1)=2000 'Y
VectAssi(2)=300 'Z trasportato
VectAssi(3)=OBJ.pc(3) 'A rimane fermo
muovi()

' Funzione per movimentazione, attende che il buffer sia vuoto
Function muovi() as Void
Dim test as Char
```

```
Label Move
test=Obj.lineto(vel,VectAssi())
if test=0
        goto Move        attesa se BUFFER pieno
endif
EndFunction
```

11.4.4 ARCTO

Movimentazione con INTERPOLAZIONE CIRCOLARE sugli ASSI del PIANO DI LAVORO SETTATO. Gli assi del PIANO DI LAVORO effettuano un'interpolazione di tipo CIRCOLARE, mentre gli altri ASSI di tipo LINEARE. In modo analogo a LINETO, la proprietà ***sglp*** identifica la fermata sul TRATTO SUCCESSIVO. Il senso dell'interpolazione circolare ORARIA o ANTIORARIA è determinato dal parametro **senso**.

Hardware ***Tutti***

Sintassi

.ARCTO(Long Vel, Char Senso, Long *PuntAssi, Long CX, Long CY) as char

Parametri

Vel	Velocità di interpolazione espressa nell'unità programmate dai parametri macchina
Senso	Senso di rotazione interpolazione circolare **Senso=2** Interpolazione ORARIA **Senso=3** Interpolazione NATIORARIA
PuntAssi	Puntatore al vettore quote finale degli assi espresse nelle unità programmate
Cx,CY	Coordinata X,Y (Intesa del piano di lavoro impostato) del CENTRO

Valore di ritorno

Char	***0***	Non inserito nel buffer (buffer movimenti pieno)
	1	Tratto inserito nel buffer
	-1	Arco impossibile (dipende da ***sglr***)

Note

Arcto effettua un interpolazione CIRCOLARE SIL PIANO DI LAVORO IMPOSTATO. Gli altri ASSI vengono interpolati comunque in MODO LINEARE

Esempio (nome oggetto = OBJ)

Variabili utilizzate:

```
VectAssi(4) long
Cx long
Cy long
Vel long
'*********************************************************
'interpolazione circolare oraria su X,Y Z ed A
'per poter realizzare il cerchio programmato occorre che
'gli assi X e Y siano inizialmente in posizioni ben precise,
'per esempio a 0,2000
'*********************************************************
vel=1000
VectAssi(4) long
VectAssi(0)=1000'punto finale X
VectAssi(1)=2000'punto finale Y
VectAssi(2)=5000'punto finale Z
VectAssi(3)=1000'punto finale A
Cx=500  'centro X
Cy=500  'centro 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 ' attesa se BUFFER pieno
endif
EndFunction
```

11.4.5 SETCMD

Questa funzione simula un asse virtuale e consente la sincronizzazione di comandi a piacere rispetto al movimento degli assi. Infatti a causa del BUFFER DI MOVIMENTAZIONE ASSI le funzioni di movimentazione non attendono l'esecuzione del comando ma lo inseriscono nel buffer. Questo comporta l'impossibilità di comandare, per esempio, delle uscite digitali in un punto ben preciso di una sequenza di movimentazione continua. Con questa funzione si abilita l'inserimento di un valore ad ogni movimentazione (***moveto, lineto, arcto***) che verrà inserito nel buffer e scritto in ***cmd*** al momento che questo viene eseguito.

Hardware ***Tutti***

Sintassi

.SETCMD(Long CMD)

Parametri

CMD Valore del comando

Esempio

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

Nel TASK PLC
if OBJ.cmd=10
        …
endif
if OBJ.cmd=20
        …
endif
```

11.4.6 SETPIANO

Setta il piano di lavoro sugli assi indicati. Di default il piano è settato sui primi due X, Y (ax1=0, ax2=1).
Ax1 non può essere uguale ad ax2.

Hardware ***Tutti***

Sintassi

.SETPIANO(Char Ax1, Char Ax2)

Parametri

Ax1 Indice del primo asse del piano
Ax2 Indice del secondo asse del piano

Note

Il PIANO DI LAVORO identifica gli assi per interpolazioni CIRCOLARI, per il calcolo della soglia spigolo 2D (***sglp***) e per il calcolo della VELOCITA' VETTORIALE con la funzione LINETO.

Esempio

```
Obj.setpiano(0,1)       'setta il piano sui primi due assi (X,Y)
Obj.setpiano(1,2)       'setta il piano sul secondo e terzo asse (Y,Z)
```

11.4.7 STOP

Ferma gli assi con rampa di decelerazione programmata (***acc***) attendendo che tutti quanti si siano fermati.
STOP viene utilizzata per fermare gli assi prima del punto di TARGET programmato da MOVETO, LINETO, ARCTO. **Il buffer dei movimenti viene svuotato**.

Hardware ***Tutti***

Sintassi

.STOP()

Note

Poiché STOP, a differenza di FSTOP, attende che gli assi si siano effettivamente fermati, questa funzione **NON DEVE ESSERE INSERITA NEL TASK PLC.**

11.4.8 FSTOP

Ferma gli assi con rampa di decelerazione programmata NON attendendo che gli assi siano FERMI.
FSTOP viene utilizzata per fermare gli assi prima del punto di TARGET programmato da MOVETO, LINETO, ARCTO. **Il buffer dei movimenti viene svuotato**.

Hardware ***Tutti***

Sintassi

FSTOP()

Note

FSTOP a differenza di STOP NON attende che gli assi si siano effettivamente fermati quindi **PUO' ANCHE ESSERE INSERITA NEL TASK PLC.**

11.4.9 MOVE

Ritorna lo stato di movimento in corso.
Hardware ***Tutti***

Sintassi

.MOVE() as char

Valore di ritorno

char	**0**	Nessun movimento in corso
	1	Movimento in corso

Note

MOVE torna 0 solo quando gli assi sono FERMI e il buffer dei movimenti è vuoto.
ATTENZIONE: MOVE torna assi fermi calcolando la posizione TEORICA DEGLI ASSI.

Esempio

```
Muovi()          'muove gli assi
while Obj.move()          'attende la fine del movimento
endif
```

11.4.10 PRESET

Preset delle quote ASSI senza muoverli. Gli assi assumono come valore di quota quella passata nei parametri.
Hardware ***Tutti***

Sintassi

.PRESET(long *Quote)

Parametri

Quote Puntatore alle quote assi da presettare

Note
Preset inizializza le quote assi al valore indicato. Occorre prestare attenzione alle seguenti condizioni:

- **GLI ASSI DEVONO ESSERE FERMI**
- **CAMBIANDO ISTANTANEAMENTE LA POSIZIONE DEGLI ASSI OCCORRE UNA SEQUENZA PARTICOLARE PER EVITARE CHE FISICAMENTE L'ASSE SI MUOVI BRUSCAMENTE**

Per esempio SE SI UTILIZZANO ASSI IN CANOPEN OCCORRE :
- **CHE QUESTI SIANO PRIMA TOLTI DALLA MODALITA' INTERPOLAZIONE**
- **FARE IL PRESET ALL' ASSE CANOPEN CON IL METODO .HOME**
- **FARE IL PRESET ALL' INTERPOLATORE CON PRESET**
- **COMMUTARE DI NUOVO L' ASSE NELLA MODALITA' INTERPOLATION MODE**

Esempio su asse in CanOpen asse X
variabili utilizzate:
Quote(3) as long

```
ASSECAN.modo=0              ' toglie da modalità interpolatore
ASSECAN.start=0         ' toglie anche lo start
ASSECAN.home=1000       ' HOME dell'asse a 1000
Quote(0)=1000               ' X stesso valore
Quote(1)=OBJ.pc(1)          ' non variata
Quote(2)=OBJ.pc(2)          ' non variata
OBJ.PRESET(Quote())         ' Preset interpolatore
ASSECAN.modo=2              ' Interpolation Mode
ASSECAN.start=1         ' start
```

In modo analogo lo stesso problema si presenta utilizzando gli assi STEP/DIR. **Vedere il capitolo relativo agli assi STEP/DIR per una corretto preset di questi**.

11.5 FUNZIONI API PER GESTIONE CANOPEN

Queste API riguardano la gestione della LINEA CANOPEN.
Premesso che gli OGGETTI rendono trasparente l'utilizzo della linea CANOPEN, in alcuni casi è necessario utilizzare le primitive per la gestione della comunicazione.

11.5.1 PXCO_SDODL

Questa funzione permette di inviare dati ad un nodo della rete utilizzando il protocollo SDO. E' supportato solo il protocollo SDO EXPEDITED consentendo quindi di inviare fino a 4byte di lunghezza dati.
Hardware ***Tutti***

Sintassi

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

Parametri

Node	Node ID dello SLAVE
Index, subindex	Indirizzo del dato da trasferire (Object-Dictionary)
Len	Numero di Byte da trasferire
***Dati**	Puntatore al buffer dati da trasferire (questo deve essere sempre un variabile)

Valore di ritorno

char	**0**	Nessun errore
	<>0	Errore di comunicazione:
	=2	Il nodo ha risposto con SDO ABORT CODE, chiamando la funzione ***read_sdoac*** nelle variabili di sistema _SYSTEM_SDOAC0 e _SYSTEM_SDOAC0 sarà disponibile il relativo codice di errore.

ATTENZIONE: A causa della diversa allocazione dei byte nelle variabili fare molta attenzione ad impostare la lunghezza corrispondente al tipo di variabile passata col puntatore.

Esempio
Variabili utilizzate:

```
Valore int
Ret char
Valore=100
Ret=pxco_sdodl(1,2000,0,2,valore())      'nodo=1, index=2000, subidx=0,
                                         'len=2 byte, valore=100
if Ret<>0                          'test se errore
        if Ret=2
                read_sdoac()'legge l'eventuale SDO ABORT CODE
        endif
endif
```

11.5.2 PXCO_SDOUL

Questa funzione permette di leggere dati da un nodo della rete utilizzando il protocollo SDO. E' supportato solo il protocollo SDO EXPEDITED consentendo quindi di leggere fino a 4byte di lunghezza dati.
Hardware ***Tutti***

Sintassi

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

Parametri

Node	Node ID dello SLAVE
Index,subindex	Indirizzo del dato da richiedere (Object-Dictionary)
***Dati**	Puntatore al buffer dati

Valore di ritorno

char **0** Nessun errore

<>0 Errore di comunicazione
=2 Il nodo ha risposto con SDO ABORT CODE, chiamando la funzione read_sdoac nelle variabili di sistema _SYSTEM_SDOAC0 e _SYSTEM_SDOAC0 sarà disponibile il relativo codice di errore.

ATTENZIONE: A causa della diversa allocazione dei byte nelle variabili fare molta attenzione ad utilizzare la variabile passata col puntatore della dimensione corrispondente al dato da leggere.

Esempio
Variabili utilizzate:

```
Valore int
Ret char
Ret=pxco_sdoul(1,2000,0,valore())'nodo=1, index=2000, subidx=0,
                                 'valore=dato letto
if Ret<>0                        'test se errore
        if Ret=2
                read_sdoac()'legge l'eventuale SDO ABORT CODE
                ...
        endif
        ...
endif
```

11.5.3 READ_SDOAC

Lettura dello SDO ABORT CODE inviato da un nodo della rete a seguito di una richiesta tramite le funzioni PXCO_SDODL e PXCO_SDOUL. Il codice letto viene scritto nelle variabili di sistema _SYSTEM_SDOAC0 e _SYSTEM_SDOAC1.
Per i codici di errore fare riferimento alle specifiche DS301 del CAN OPEN.
Hardware ***Tutti***

Sintassi

READ_SDOAC()

11.5.4 PXCO_SEND

Invio di un pacchetto CAN a basso livello. Questa funzione consente di inviare alla rete un frame CAN indicando COB-ID e DATI. Con questa funzione sarà possibile quindi inviare ad esempio PDO in modo manuale, pacchetti HEART-BEAT, ecc.
Occorre precisare che la gestione dei PDO viene eseguita in modo AUTOMATICO tramite il CONFIGURATORE CANOPEN.
Hardware ***Tutti***

Sintassi

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

Parametri
Id Valore del COB_ID
Len Numero di dati da trasferire
***Dati** Puntatore al buffer dati da trasferire

Valore di ritorno
char **0** Nessun errore
<>0 Errore di comunicazione

Esempio
Variabili utilizzate:

```
Valore int
Ret char
Valore=100
Ret=pxco_send(0x201,2,valore())  'Invia un PDO (cob-id=0x201) con 2 byte
if Ret<>0          'test se errore
        ...
endif
```

11.5.5 PXCO_NMT

Invio di un pacchetto NMT del CAN OPEN. I pacchetti NMT consentono di settare lo stato dei nodi nella rete. Occorre tenere presente che tutti i nodo presenti in configurazione (configuratore canopen) ed che l'hanno eseguita senza errori sono messi in stato START in modo automatico.

Hardware ***Tutti***

Sintassi

PXCO_NMT(char state, char node) as char

Parametri

state Stato da inviare:
1 = START NODE
2 = STOP NODE
128 = PRE-OPERATIONAL
129 = RESET NODE
130 = RESET COMUNICATION

node Numero del nodo

Valore di ritorno

char **0** Nessun errore
<>0 Errore di comunicazione

Esempio

Variabili utilizzate:

pxco_nmt(2,1) ***'Mette in STOP in nodo 1***

11.5.6 READ_EMCY

Legge l'ultimo pacchetto EMERGENCY OBJECT inviato da un eventuale nodo della linea CAN OPEN.
Il codice di emergenza viene inserito nella variabile vettore di sistema _SYSTEM_EMCY(8) che conterrà gli
8 byte relativi al pacchetto EMERGENCY OBJECT previsto dalle specifiche DS301 del CAN OPEN. Si consiglia di leggere questa funzione in modo ciclico. I codice di allarme sono dipendenti dal tipo di dispositivo collegato, occorrerà quindi riferirsi al manuale di tale dispositivo.

Hardware ***Tutti***

Sintassi

READ_EMCY() as char

Valore di ritorno

char **0** Nessun errore
<>0 Nodo che ha generato l'emergency object.

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

ATTENZIONE

Il sistema non bufferizza più errori, quindi nel caso in cui si verifichino più EMERGENCY OBJECT in contemporanea viene letto solo l' ultimo.

Un EMERGENCY OBJECT non significa che effettivamente ci sia un nodo in emergenza. Questo poiché le specifiche DS301 prevedono che tale pacchetto sia inviato anche al ripristino di una fase di emergenza. Alcuni dispositivi possono inviare all'accensione questo pacchetto.

Esempio
Variabili usate:
Err Long
ControllaErr Char

```
function Allarmi() as void
        ControllaErr=read_emcy()
        if ControllaErr=0                       ' nessun errore
                return
        endif
        err=(_SYSTEM_EMCY(7)&0xff)      ' Legge 4 byte del Manufactured specific
        err=err<<8                                      ' field e Maschera eventuali bit non
        err=err|(_SYSTEM_EMCY(6)&0xff)' interessati
        err=err<<8
        err=err|(_SYSTEM_EMCY(5)&0xff)
        err=err<<8
        err=err|(_SYSTEM_EMCY(4)&0xff)
endfunction
```

11.6 FUNZIONI API PER LA GESTIONE DELLA MEMORIA PERMANENTE

Nelle piattaforme Hardware possono trovarsi diversi dispositivi per il SALVATAGGIO DATI.
Dipendentemente dall'hardware questi possono essere di TIPO diverso tra di loro (Flash, Fram, ecc.) quindi il salvataggio dati deve rispettare la tipologia di costruzione di questi. Per esempio in una FLASH occorre seguire le regole del costruttore in termini di ***numero massimo di scritture, cancellazione a blocchi, ecc.***

11.6.1 IMS_WRITE

Scrive nella FLASH interna all'indirizzo indicato da ADDR, i dati indirizzati dal puntatore PUNT per un totale di NBYTE dati.
La FLASH viene gestita a BLOCCHI di 256 Bytes, pertanto si consiglia sempre di scrivere a MULTIPLI di 256 come lunghezza dati. Questo perché se vengono scritti meno di 256 Bytes, viene comunque cancellato l'intero BLOCCO, per non perdere i dati occorrerebbe quindi leggere prima tutto il blocco, salvare i dati interessati e sovrascrivere il tutto. Tuttavia i sistemi NG35, hanno sufficiente FLASH da poter essere gestita tranquillamente a blocchi di 256 anche se i dati da salvare sono minori.
Nel caso di schede tipo **NGM13,NGMEVO,NGQ,NGQx**, la scrittura viene in FRAM e questa può essere gestita a BYTE.
Hardware ***Tutti***

Sintassi

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

Parametri

Punt	Puntatore al buffer di dati
Addr	Indirizzo di partenza della MEMORIA
Nbyte	Numero di byte da scrivere

Valore di ritorno

Char	0	Nessun errore
	<>0	Errore scrittura

Esempio
Variabili utilizzate:
Vett(10) long
Ims_Write(Vett(),0,40) ' scrive 40 byte (10 long * 4) ad ADDR 0

ATTENZIONE: In questo caso viene comunque scritto un intero blocco di 256 Bytes nel caso di salvataggio in FLASH

11.6.2 IMS_READ

Legge da Memoria interna i dati a partire da ADDR, per una lunghezza di NBYTE e l' inserisce nel vettore puntato da PUNT.
Hardware ***Tutti***

Sintassi

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

Parametri

Punt	Puntatore al buffer di dati di lettura
Addr	Indirizzo di partenza della MEMORIA
Nbyte	Numero di byte da leggere

Valore di ritorno

Char	0	Nessun errore
	<>0	Errore lettura

Esempio
variabili utilizzate:
Vett(10) long
Ims_Read(Vett(),0,40) ' legge 40 byte (10 Long) da Addr 0

11.7 FUNZIONI API PER LA GESTIONE DI ETHERNET

Le piattaforme con ETHERNET a bordo gestiscono in modo AUTOMATICO lo STACK TCP/IP. La gestione di protocolli che si appoggiano a questo è demandata a livello di applicazione. Per esempio la gestione del protocollo MODBUS-TCP è gestita completamente da uno specifico oggetto ed usa questo gruppo di funzioni. Allo stesso modo è possibile creare protocolli personalizzati.

11.7.1 SET_IP

Setta i parametri del protocollo TCP/IP.
Hardware ***NG35,NGMEVO***

Sintassi

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

Parametri
ip indirizzo IP della scheda
sm subnet mask
gw gateway

Esempio

```
Set_ip("10.0.0.15","255.255.255.0",0)      'IP = 10.0.0.15
                                           'SUBNET = 255.255.255.0
                                           'GATEWAY = nessuno
```

ATTENZIONE: E' necessario inizializzare l' indirizzo IP su INIT della MAIN o del TASK PLC

11.7.2 PXETH_ADD_PROT

Aggiunge la gestione di un protocollo che si appoggia al TCP/IP. Occorre scrivere la funzione di gestione del protocollo e passare il puntatore a questa funzione.
Hardware ***NG35,NGMEVO***

Sintassi

PXETH_ADD_PROT(port as long, fun as delegate)

Parametri
port Porta TCP sulla quale aggiungere il protocollo
fun Puntatore alla funzione di gestione del protocollo

Esempio
Variabili utilizzate:
fun delegate

```
fun=my_protocol
pxeth_add_prot(502,fun) 'Aggiunge il protocollo my_protocol alla porta 502

'Funzione di gestione del protocollo
function my_protocol(len as long, buftx as *char) as long
...
endfunction
```

11.7.3 Funzione di GESTIONE PROTOCOLLO

Questa funzione non è definita dal sistema ma deve essere scritta nell'applicazione. Il sistema chiama questa funzione, tramite il puntatore passato in ***pxeth_add_prot,*** ogni volta che viene ricevuto un pacchetto dati sulla porta definita per questo protocollo. Per leggere i dati occorre utilizzare la funzione **pxeth_rx** mentre per inviare dei dati di risposta andranno scritti nel buffer di trasmissione (buftx) e uscire dalla funzione ritornando il numero di byte che si vogliono inviare.

Hardware ***NG35,NGMEVO***

Sintassi

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

Parametri

len Lunghezza del pacchetto dati ricevuto

buftx Puntatore al buffer di trasmissione

Valore di ritorno

long Numero di byte da inviare scritti nel buffer di trasmissione

Esempio

Variabili utilizzate:

bufrx(100) char

```
'Funzione di gestione del protocollo
function my_protocol(len as long, buftx as *char) as long
dim i as int

for i=0 to i<len            'Legge i dati ricevuti
        bufrx(i)=pxeth_rx()
next i
...                         'Processa i dati
buftx(0)=12
buftx(1)=34
my_protocol=2               'Comanda l'invio di 2 byte come risposta
endfunction
```

11.7.4 PXETH_RX

Legge un byte dal buffer di ricezione TCP/IP. Viene utilizzata nella funzione di gestione del protocollo per leggere i dati ricevuti.

Hardware ***NG35,NGMEVO***

Sintassi

PXETH_RX() as char

Valore di ritorno

Char Dato letto dal buffer di ricezione

11.8 FUNZIONI API PER LA GESTIONE FAT16

Alcune piattaforme, quali NG35 e possono gestire file tramite la formattazione standard FAT16 (o FAT32 su dispositivi tipo FLASH Le funzioni di libreria sono contenute nell'oggetto FATLIB che quindi dovrà essere caricato. Di seguito vengono descritte le funzioni GENERICHE interne all'oggetto. Ricordarsi di mettere sempre il prefisso del NOME_OGGETTO. Se per esempio l'oggetto si chiama ***disk*** la funzione ***OpenRead*** dovrà essere richiamata con ***disk.OpenRead***.

Hardware ***NG35 con opzione Flash Disk***

11.8.1 PROPRIETA'

Numero files Numero massimo di files apribili contemporaneamente. L'HANDLE del file dovrà essere un numero compreso tra 0 e questo valore meno uno. Si può cambiare solo a livello di ambiente VTB.

FAT Monitor Abilita il monitor dei comandi sulla seconda porta seriale. Si può cambiare solo a livello di ambiente VTB.

11.8.2 DRIVER

Il sistema può gestire più driver nel caso siano presenti nell'hardware. Il riferimento tramite la path avviene in modo standard (A:, B:, ecc.) mentre per alcune funzioni occorre passare l'indice. I riferimenti dei driver rispetto all'hardware è il seguente:

	A:	**B:**
NG35	Disco opzionale interno	Non presente
	Disco opzionale interno	USB Key

11.8.3 CODICI DI ERRORE

Tutte le funzioni di questo oggetto, eccetto **TestDrv**, **RTC.Read** and **RTC.Write**, ritornano un valore corrispondente al codice di errore.

Valore di ritorno

Char	0	OK Nessun errore
	1	DISK ERROR
	2	INTERNAL ERROR
	3	NOT READY
	4	NO FILE
	5	NO PATH
	6	INVALID NAME
	7	ACCESS DENIED
	8	FILE/DIR EXIST
	9	INVALID OBJECT
	10	WRITE PROTECTED
	11	INVALID DRIVE
	12	NOT ENABLED
	13	NO FILESYSTEM
	14	FORMAT ERROR
	15	TIMEOUT
	100	HANDLE OVERFLOW

11.8.4 OPENREAD, OPENWRITE, OPENCREATE

Queste funzioni aprono il file indicato associandogli un HANDLE al quale fare riferimento successivamente.

Sintassi

.OpenRead(handle as int, path as *char) as char
Apre il file in lettura, ritorna errore se non esiste.

.OpenWrite(handle as int, path as *char) as char
Apre il file in scrittura, ritorna errore se non esiste.

.OpenCreate(handle as int, path as *char) as char
Crea un nuovo file e lo apre in scrittura, se è già presente viene sovrascritto.

Parametri

handle	Numero da associare al file per tutti i riferimenti
path	Nome del file, può contenere anche la path

Esempio

Variabili utilizzate:
err char

```
err=disk.OpenRead(1,"\dati\tabella.dat") ' apre tabella.dat nella cartella dati
if err
    ...
endif
```

11.8.5 CLOSE

Chiude il file con l'HANDLE indicato liberandolo per successivi utilizzi.

Sintassi

.Close(handle as int) as char

Parametri

handle	Numero di riferimento del file

Esempio

Variabili utilizzate:
err char

```
err=disk.OpenRead(1,"\dati\tabella.dat") ' apre tabella.dat nella cartella dati
if err
    ...
endif
    ...
disk.Close(1) ' chiude il file
```

11.8.6 READ

Lettura dati dal file con l'HANDLE indicato. Saranno letti un numero di byte fino a quelli indicati, se la fine del file verrà incontrata prima la lettura terminerà. In NB verrà scritto l'effettivo numero di byte letti.

Sintassi

.Read(handle as int, dati as *char, len as long, nb as *long) as char

Parametri

handle	Numero di riferimento del file
dati	Puntatore al buffer dove saranno scritti i dati
len	Numero di byte da leggere del file
nb	Puntatore alla variabile dove sarà memorizzato il numero di byte effettivamente letti

Esempio

Variabili utilizzate:

```
err char
dati(100) char
nbyte long

err=disk.OpenRead(1,"\dati\tabella.dat") 'apre tabella.dat nella cartella dati
if err
        ...
endif
while 1
        err=disk.Read(1,dati(),10,nbyte()) 'legge 10 byte alla volta
        if err
                ...
        endif
        if nbyte<10                                     'fino alla fine del file
                exitwhile
        endif
loop
disk.Close(1) ' chiude il file
```

11.8.7 WRITE

Scrittura di dati nel file con l'HANDLE indicato.

Sintassi

.Write(handle as int, dati as *char, len as long, nb as *long) as char

Parametri

handle	Numero di riferimento del file
dati	Puntatore al buffer dati da scrivere
len	Numero di byte da scrivere
nb	Puntatore alla variabile dove sarà memorizzato il numero di byte effettivamente scritti

Esempio

Variabili utilizzate:

```
err char
dati(100) char
nbyte long
err=disk.OpenCreate(1,"\dati\tabella.dat") 'crea tabella.dat nella cartella dati
if err
        ...
endif
```

```
        ...                                     'prepara i dati da scrivere
err=disk.Write(1,dati(),50,nbyte())'scrive 50 byte
if err
        ...
endif
disk.Close(1) ' chiude il file
```

11.8.8 SEEK, SEEKEOF, SEEKREL

Imposta il puntatore corrente all'interno del file.

Sintassi

.Seek(handle as int, offset as long) as char
Setta l'offset rispetto all'inizio del file.

.SeekEof(handle as int, offset as long) as char
Setta l'offset rispetto la fine del file.

.SeekRel(handle as int, offs as long) as char
Setta l'offset rispetto alla posizione corrente del file.

Parametri

handle	Numero di riferimento del file
offs	Valore dell'offset in numero byte

Esempio

```
err=disk.OpenRead(1,"\dati\tabella.dat") ' apre il file
...
err=disk.Seek(1,200)        ' si posiziona per la lettura dal byte 200
```

11.8.9 CHDIR

Cambia la cartella corrente. Tutte le funzioni che seguiranno in mancanza di una path completa faranno riferimento a quella corrente.

Sintassi

.Chdir(path as *char) as char

Parametri

path	Nome della cartella completa di path

Esempio

```
err=disk.Chdir("programmi")
err=disk.OpenCreate(1,"file.txt")   ' crea il file file.txt nella cartella                    '
programmi
```

11.8.10 MKDIR

Crea una nuova cartella, ritorna errore se è già esistente.

Sintassi

.Mkdir(path as *char) as char

Parametri

path	Nome della cartella completa di path

Esempio

err=disk.Mkdir("\prova\text") ***' crea la cartella text dentro la cartella prova***

11.8.11 DELETE, ERASE, KILL

Cancella un file o una cartella. La stessa funzione può essere richiamata con tre nomi diversi.

Sintassi

.Delete(path as *char) as char
.Erase(path as *char) as char
.Kill(path as *char) as char

Parametri
path Nome del file completo di path da cancellare

Esempio
err=disk.kill("\prova\text") ***' cancella la cartella/file text dentro la cartella prova***

11.8.12 RENAME

Rinomina un file o una cartella. Ritorna errore se una cartella col nuovo nome è già esistente.

Sintassi

.Rename(oldpath as *char, newpath as *char) as char

Parametri
oldpath Nome del file/cartella da rinominare
newpath Nome nuovo del file/cartella da rinominare

Esempio
err=disk.Rename("text.txt","dati.dat") ***' rinomina il file text.txt con ' con dati.dat nella corrente cartella***

11.8.13 COPY

Copia di un file. Se un file col nome di destinazione è già presente questo viene sovrascritto.

Sintassi

.Copy(srcpath as *char, dstpath as *char) as char

Parametri
srcpath Nome del file da copiare
dstpath Nome del file duplicato.

ATTENZIONE: La path di destinazione deve sempre contenere il nome del file. Non può contenere solo il percorso.

Esempio
err=disk.Copy("text.txt","B:dati.dat") ***' copia il file text.txt nel driver B:***
err=disk.Copy("text.txt","\prova\dati.dat") ***' copia il file text.txt nella ' cartella prova***

11.8.14 OPENDIR

Apre una cartella. E' il punto di partenza per una ricerca dei file presenti nel disco. Usata insieme a ***ReadDir***.

Sintassi

.OpenDir(path as *char) as char

Parametri

path Nome della cartella. Se la stringa è vuota viene presa la cartella corrente.

11.8.15 READDIR

Legge le informazioni del primo file/cartella trovato nella FAT. Le informazioni sono salvate nella struttura ***NomeOggetto_finfo***.

Sintassi

.ReadDir() as char

Struttura NomeOggetto_finfo

.size	Dimensione del file	
.date	Data del file	bit 0-4 giorno (1-31) *bit 5-8 mese (1-12)* *bit 9-15 anno (0-99)*
.time	Ora del file	bit 5-10 minuti (0-59) *bit 11-15 ora (0-23)*
.attrib	Attributo	bit 0 read-only *bit 1 hidden* *bit 2 system* *bit 3 volume* *bit 4 directory* *bit 5 arch.*
.name(13)	Nome corto es. "abcdefgh.abc"	
.lname	Puntatore al nome esteso (max 255 caratteri)	

Esempio

```
' Funzione che stampa sulla porta seriale l'elenco dei file presenti
' nella cartella corrente
function list_dir() as void
dim res as char
dim pname as *char
dim flbyte as long

res=disk.OpenDir("")
if res
        ser_puts("No file")
        ser_putchar(10)
        ser_putchar(13)
        return
endif
while 1
        res = disk.ReadDir()
        if res || disk_finfo.name(0)=0
                return
        endif
        ser_printl("00",disk_finfo.date & 31)
        ser_printl("/00",(disk_finfo.date >> 5) & 15)
        ser_printl("/####",(disk_finfo.date >> 9) + 1980)
        ser_printl(" 00",disk_finfo.time >> 11)
```

```
        ser_printl(":00",(disk_finfo.time >> 5) & 63)
        if disk_finfo.attrib & ?p1?.ATTR_DIR
                ser_puts("  <DIR>  ")
        else
                ser_printl(" ######### bytes ",disk_finfo.size)
        endif
        ser_puts(" - ")
        ser_puts(disk_finfo.name())
        ser_puts(" - ")
        ser_puts(disk_finfo.lname)
        ser_putchar(10)
        ser_putchar(13)
loop
endfunction
```

11.8.16 GETFREE

Legge le caratteristiche di un driver: dimensione totale e numero di byte liberi. Le informazioni sono salvate nella struttura ***NomeOggetto_dinfo***

Sintassi

.GetFree(drv as char) as char

Parametri

drv Indice del driver da testare:
0 = A:
1 = B:

Struttura NomeOggetto_dinfo

.btot Dimensione del disco in byte
.bfree Numero di byte disponibili

Esempio

```
err=disk.GetFree(0)
ser_puts("bytes free: ")
ser_printl("#.###.###.### ",disk_dinfo.bfree)
ser_puts("su ")
ser_printl("#.###.###.### ",disk_dinfo.btot)
```

11.8.17 CHDRV

Setta il driver corrente. Tutte le funzioni che seguiranno in mancanza del nome driver nella path faranno riferimento a quello corrente.

Sintassi

.ChDrv(drv as char) as char

Parametri

drv Indice del driver da testare:
0 = A:
1 = B:

Esempio

```
err=disk.ChDrv("B:")
err=disk.OpenCreate(1,"file.txt")   ' crea il file file.txt nel driver B:
```

11.8.18 TESTDRV

Controlla se un driver è presente. Questa è l'unica funzione che **non ritorna il codice di errore come le altre**.

Sintassi

.TestDrv(drv as char) as char

Parametri

drv Indice del driver da testare:
0 = A:
1 = B:

Valore di ritorno

Char	0	Nessun driver trovato
	1	Driver presente

ATTENZIONE: Questa funzione controlla solo la presenza del disco ma non se in questo è presente una FAT.

11.8.19 REAL TIME CLOCK (RTC)

Quando vengono creati dei file nei relativi campi della FAT vengono scritte data e ora. Per questo nello stesso oggetto sono presenti le funzioni di lettura e scrittura dell'orologio. Tutte le informazioni passano da una struttura predefinita di nome **RTC**.

Sintassi

RTC.Read() as void
Legge il Real Time Clock

RTC.Write() as void
Scrive nel Real Time Clock

Struttura RTC

RTC.year Anno (0-99)
RTC.month Mese (1-12)
RTC.day Giorno (1-31)
RTC.dweek Giorno settimanale (0-6)
RTC.hour Ora (0-23)
RTC.min Minuti (0-59)
RTC.sec Secondi (0-59)

11.9 FUNZIONI API PER LA GESTIONE DELLA PIATTAFORMA NG35

Queste funzioni sono relative alla PIATTAFORMA HARDWARE NG35.
Hardware ***NG35***

11.9.1 NG_DI - LETTURA INGRESSI DIGITALI NG35

Questa funzione permette di leggere gli ingressi delle espansioni **I/O NGIO/NGPP** della serie NG35.
Le schede sono identificate da un indice progressivo partendo da 0 per la prima scheda. La prima scheda è quella più vicina alla CPU, mentre l'ultima è quella più lontana.

Sintassi

NG_DI(Char Card) as uint

Parametri
Card Numero della scheda di espansione (da 0 a 7)

Valore di ritorno
Uint Valore dei 16 ingressi gestiti a BIT
bit a 1 = ingresso ATTIVO

Ingresso	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Esempio
Variabili utilizzate:
input UINT

```
input = ng_di(0)  ' legge gli ingressi della prima scheda I/O
input = ng_di(2)  ' legge gli ingressi della terza scheda I/O
```

11.9.2 NG_DO - SCRITTURA USCITE DIGITALI NG35

Questa funzione permette di scrivere le uscite delle espansioni **I/O NGIO/NGPP** della serie NG35.
Le schede sono identificate da un indice progressivo partendo da 0 per la prima scheda. La prima scheda è quella più vicina alla CPU, mentre l'ultima è quella più lontana.

Sintassi

NG_DO(Char Card, Uint Out)

Parametri
Card Numero della scheda di espansione (da 0 a 7)
Out Stato delle uscite
bit a 1 = uscita ATTIVA

Uscita		14	13	12	11	10	9		8	7	6	5	4	3	2	1
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Esempio
Variabili utilizzate:

```
ng_Do(0,0x7)        ' Set ta le uscite 1, 2 e 3 della Card 0
ng_Do(1,0x31)       ' Set ta le uscite 1, 9 e 10 della Card 1
```

ATTENZIONE: I bit 8 e 15 non sono usati.

11.9.3 NOTE SULLA GESTIONE DELLE I/O DIGITALI NG35

Per una programmazione più pulita e fluida è consigliabile gestire le I/O nel TASK PLC. Pertanto nel TASK, leggere ciclicamente gli ingressi scrivendoli in una variabile GLOBALE (es. Input) e scrivere le uscite leggendole da un'altra variabile GLOBALE (es. Output). Su queste variabili potranno essere definiti i singoli bit associati ai canali digitali e quindi utilizzarli all'occorrenza.

Esempio
Variabili utilizzate:
Input1 UINT
Input2 UINT
Output1 UINT
Output2 UINT
PulsanteStart BIT Input1.3
PulsanteStop BIT Input1.6
PompaAcqua BIT Output2.12

Nel TASK PLC:
Input1=Ng_Di(0)
Input2=Ng_Di(1)
Ng_Do(0,Out1)
Ng_Do(1,Out2)

OVUNQUE:
```
if  PulsanteStart
        PompaAcqua=1
endif
if  PulsanteStop
        PompaAcqua=0
endif
```

11.9.4 NG_ADC - LETTURA INGRESSI ANALOGICI NG35

La scheda NG35 possiede a bordo 8 Canali analogici a **10 Bit**, questi possono essere letti tramite la funzione NG_ADC.

Sintassi

NG_ADC(Char Chan) as uint

Parametri
Chan Numero del canale (da 0 a 7)

Valore di ritorno
Ritorna il valore analogico letto da 0 a 1023.

11.9.5 NG_DAC - SCRITTURA USCITE ANALOGICHE NG35

Questa funzione permette di pilotare l'uscita analogica di ogni canale gestito dal controllo NG35 tramite l'espansioni **NG-IO** e **NG-PP** (opzionale).
Le schede dispongono di convertitori digitale/analogico a 12 bit, su un fondo scala di 10V. Per cui con +2047 si ottengono 10V in uscita, con -2047 -10V in uscita.
La selezione del canale avviene con un indice che va da 0 a 7, ogni espansione gestisce due canali quindi avremo:

Indice Canale	Espansione
0	**Scheda 0 (più vicina alla NG35)**
1	
2	**Scheda 1**
3	
4	**Scheda 2**
5	
6	**Scheda 3**
7	

Sintassi

NG_DAC(Char Chan, Long Val)

Parametri

Chan Numero Canale (da 0 a 7)
val Valore dell'uscita (da -2047 a +2047)

Esempio

Variabili utilizzate:
val LONG
channel CHAR

```
channel = 0
val = 1024
ng_dac(channel, val)      'Scrive 1024 (~5V) su uscita analogica 0
ng_dac(1,512)             'Scrive 512 (~2,5V )su uscita analogica asse 1
```

11.9.6 CALIBRAZIONE OFFSET USCITE ANALOGICHE NG35

Questa funzione permette di calibrare l' OFFSET dell'uscita analogica di ogni CANALE gestito dal controllo NG35. Vari fattori dovuti all'hardware fanno sì che, anche se viene pilotata con valori nulli, l'uscita risulti diversa da 0 nell'ordine di pochi mV (offset). Questa funzione permette di azzerare questo effetto.
Con questa funzione si sposta in maniera software lo 0V dell'uscita che comunque non potrà mai essere esattamente nulla. Considerando che per ogni unità si ottengono circa 4mV in uscita, i valori da passare dovrebbero essere nell'ordine di poche unità, in senso opposta all'uscita rilevata.

Sintassi

NG_DAC_CAL(Char Chan, Long Offset)

Parametri

Chan Numero Canale (da 0 a 7)
Offset Valore dell'OFFSET

ATTENZIONE: IL VALORE DELL'OFFSET NON RIMANE MEMORIZZATO, PERTANTO AD OGNI ACCENSIONE OCCORRE IMPOSTARLO DI NUOVO.

11.9.7 NG_ENC - LETTURA CANALI ENCODER NG35

Questa funzione permette di leggere l'ingresso encoder (quota incrementale attuale) di ogni canale gestito sulla scheda di espansione **NG-IO** con una risoluzione di 32 bit. Questa funzione legge soltanto l'incremento che sarà aggiunto alla variabile passata con un puntatore. Il contatore vero e proprio quindi sarà contenuto in una variabile definita dall'applicazione e che potrà essere azzerata in qualunque momento. Per una corretta gestione degli encoder raccomandiamo di utilizzare questa funzione esclusivamente nel TASK PLC e quindi utilizzare la variabile di appoggio all'occorrenza. La selezione del canale avviene con un indice che va da 0 a 15, ogni espansione gestisce due canali quindi avremo:

Indice Canale	Espansione
0	Scheda 0 (più vicina alla NG35)
1	
2	Scheda 1
3	
...	...
...	
14	Scheda 7
15	

Sintassi

NG_ENC(Char Chan, Long *Quota)

Parametri

Chan Numero canale encoder (da 0 a 15)

Quota Puntatore alla variabile di tipo long in cui viene memorizzato il conteggio

Esempio

Variabili utilizzate:

```
posx LONG              ' Contatore encoder canale 0
posy LONG              ' Contatore encoder canale 1
```

Nel TASK PLC:

```
ng_enc(0,posx())
ng_enc(1,posy())
```

OVUNQUE:

```
if  posx>25000         ' Lettura encoder canale 0
        . . .
        posx=0         ' Azzeramento contatore canale 0
endif
if  posy>200000   ' Lettura encoder canale 1
        . . .
        posy=1000      ' Preset contatore canale 1
endif
```

11.9.8 NG-T0 - LETTURA INDICE DI ZERO ENCODER NG35

Questa funzione permette di leggere lo stato dell'ingresso tacca di zero di ogni canale encoder gestito sulla scheda di espansione **NG-IO**. Per la selezione del canale vale quanto detto per la lettura encoder.

Sintassi

NG_T0(Char Chan) as char

Parametri

Chan Numero canale encoder (da 0 a 15)

Valore di ritorno

Stato della tacca:

0 OFF

1 ON

ATTENZIONE: L'INGRESSO E' DIFFERENZIALE, LO STATO ON CORRISPONDE AD UNA TENSIONE SUL CH+ MAGGIORE DI QUELLA SUL CH-.

Esempio

```
if ng_t0(0)
    . . .
endif
```

11.9.9 NG_RELE - GESTIONE DEI RELE' NG35-NGIO

Questa funzione permette la gestione dei 2 RELE' presenti su ogni espansione **NG-IO**.
Normalmente questi RELE' servono per l' ABILITAZIONE DEL DRIVER del relativo ASSE ma possono essere utilizzate anche per scopi generici. Per la selezione del canale vale quanto detto per la lettura encoder.

Sintassi

NG_RELE(Char Chan, char Stato)

Parametri

CH Canale relativo all'attivazione del rele' (da 0 a 16)
Stato Stato del rele':
0 OFF(contatto aperto)
1 ON (contatto chiuso)

Esempio

Variabili utilizzate:

```
channel UINT
stato  UINT

channel = 1
stato = 1
Ng_rele(channel,stato)    'setta il relè del secondo canale
channel = 2
stato = 0
ng_rele(channel,stato)    'resetta il relè del terzo canale
ng_rele(0,1)                    'setta il relè del primo canale
```

11.9.10 LETTURA TEMPERATURA NG35

La Scheda NG35 presenta al suo interno un SENSORE DI TEMPERATURA.
Questo può risultare utile per monitorare la temperatura interna. Il sensore è collegato sul un CANALE ANALOGICO che precisamente è il **Nr. 9.** Quindi per la lettura viene utilizzata la stessa funzione NG_ADC per la gestione dei canali analogici. Sul valore letto occorre comunque effettuare una trasformazione per riportare la temperatura in Gradi Centigradi.

Esempio

```
Function Read_Temp() as Long
Dim Gradi as Long

Gradi=NG_ADC(8)                          ' Legge il canale 9
Gradi= Gradi*3300/1024-600       ' Riporta in decimi di grado 375=37,5 gradi
Read_Temp=Gradi
EndFunction
```

11.10 FUNZIONI API PER LA GESTIONE DELLA espansione NGMsX per NGMEVO

Queste funzioni sono relative alla PIATTAFORMA HARDWARE NGMEVO e NGMsX

Hardware ***NGMEVO***

11.10.1 NG_DAC - SCRITTURA USCITE ANALOGICHE NGMsX

Questa funzione permette di pilotare l'uscita analogica di ogni canale gestito dal controllo NGMEVO e scheda NGMsX
Le schede dispongono di convertitori digitale/analogico a 12 bit, su un fondo scala di 10V. Per cui con +2047 si ottengono 10V in uscita, con -2047 -10V in uscita.
La selezione del canale avviene con un indice che va da 0 a 2, ogni espansione gestisce due canali quindi avremo:

Indice Canale	Espansione
0	**Scheda 0 (più vicina alla NGMEVO)**
1	
2	**Scheda 1**
3	
4	**Scheda 2**
5	

Sintassi

NG_DAC(Char Chan, Long Val)

Parametri

Chan Numero Canale (da 0 a 5)
val Valore dell'uscita (da -2047 a +2047)

Esempio

Variabili utilizzate:
val LONG
channel CHAR

```
channel = 0
val = 1024
ng_dac(channel, val)      'Scrive 1024 (~5V) su uscita analogica 0
ng_dac(1,512)             'Scrive 512 (~2,5V )su uscita analogica asse 1
```

11.10.2 CALIBRAZIONE OFFSET USCITE ANALOGICHE NGMsX

Questa funzione permette di calibrare l' OFFSET dell'uscita analogica di ogni CANALE gestito dal controllo NGMsX

Sintassi

NG_DAC_CAL(Char Chan, Long Offset)

Parametri

Chan Numero Canale (da 0 a 5)
Offset Valore dell'OFFSET

ATTENZIONE: IL VALORE DELL'OFFSET NON RIMANE MEMORIZZATO, PERTANTO AD OGNI ACCENSIONE OCCORRE IMPOSTARLO DI NUOVO.

11.10.3 NG_ENC - LETTURA CANALI ENCODER NGMsX

Questa funzione permette di leggere l'ingresso encoder (quota incrementale attuale) di ogni canale gestito sulla scheda di espansione **NGMsX** con una risoluzione di 32 bit. Questa funzione legge soltanto l'incremento che sarà aggiunto alla variabile passata con un puntatore. Il contatore vero e proprio quindi sarà contenuto in una variabile definita dall'applicazione e che potrà essere azzerata in qualunque momento. Per una corretta gestione degli encoder raccomandiamo di utilizzare questa funzione esclusivamente nel TASK PLC e quindi utilizzare la variabile di appoggio all'occorrenza. La selezione del canale avviene con un indice che va da 0 a 15, ogni espansione gestisce due canali quindi avremo:

Indice Canale	Espansione
0	Scheda 0 (più vicina alla NGMEVO)
1	
2	Scheda 1
3	
4	Scheda 2
5	

Sintassi

NG_ENC(Char Chan, Long *Quota)

Parametri

Chan Numero canale encoder (da 0 a 5)
val Puntatore alla variabile di tipo long in cui viene memorizzato il conteggio

Esempio

Variabili utilizzate:

```
posx LONG          ' Contatore encoder canale 0
posy LONG          ' Contatore encoder canale 1
```

Nel TASK PLC:

```
ng_enc(0,posx)
ng_enc(1,posy)
```

OVUNQUE:

```
if  posx>25000          ' Lettura encoder canale 0
        . . .
        posx=0          ' Azzeramento contatore canale 0
endif
if  posy>200000   ' Lettura encoder canale 1
        . . .
        posy=1000       ' Preset contatore canale 1
endif
```

11.10.4 NG-T0 - LETTURA INDICE DI ZERO ENCODER NGMsX

Questa funzione permette di leggere lo stato dell'ingresso tacca di zero di ogni canale encoder gestito sulla scheda di espansione **NGMsX**. Per la selezione del canale vale quanto detto per la lettura encoder.

Sintassi

NG_T0(Char Chan) as char

Parametri

Chan Numero canale encoder (da 0 a 5)

Valore di ritorno
Stato della tacca:
0 OFF
1 ON

ATTENZIONE: L'INGRESSO E' DIFFERENZIALE, LO STATO ON CORRISPONDE AD UNA TENSIONE SUL CH+ MAGGIORE DI QUELLA SUL CH-.

Esempio
```
if ng_t0(0)
        . . .
endif
```

11.10.5 NG_RELE - GESTIONE DEI RELE' NGMsX

Questa funzione permette la gestione dei 2 RELE' presenti su ogni espansione **NGMsX**.

Sintassi
NG_RELE(Char Chan, char Stato)

Parametri
CH Canale relativo all'attivazione del rele' (da 0 a 5)
Stato Stato del rele':
0 OFF(contatto aperto)
1 ON (contatto chiuso)

Esempio
Variabili utilizzate:
channel UINT
stato UINT

```
channel = 1
stato = 1
Ng_rele(channel,stato)    'setta il relè del secondo canale
channel = 2
stato = 0
ng_rele(channel,stato)    'resetta il relè del terzo canale
ng_rele(0,1)                    'setta il relè del primo canale
```

11.11 FUNZIONI API PER LA GESTIONE DELLA uscita analogica su NGQ

Queste funzioni sono relative alla PIATTAFORMA HARDWARE NGQ

Hardware ***NGQ***

11.11.1 NG_DAC - SCRITTURA USCITE ANALOGICHE NGQ

Questa funzione permette di pilotare l'uscita analogica di ogni canale gestito dal controllo NGQ

Le schede dispongono di convertitori digitale/analogico a 12 bit, su un fondo scala di 10V. Per cui con +2047 si ottengono 10V in uscita, con -2047 -10V in uscita.

La selezione del canale avviene con un indice che va da 0 a 1

ATTENZIONE: Per abilitare le uscite analogiche sulla scheda NGQ, è necessario impostare la proprietà ENCODER ENABLE=true dell' oggetto NGQ INIT

Sintassi

NG_DAC(Char Chan, Long Val)

Parametri

Chan Numero Canale (da 0 a 1)

val Valore dell'uscita (da -2047 a +2047)

Esempio

Variabili utilizzate:

val LONG

channel CHAR

```
channel = 0
val = 1024
ng_dac(channel, val)      'Scrive 1024 (~5V) su uscita analogica 0
ng_dac(1,512)             'Scrive 512 (~2,5V )su uscita analogica asse 1
```

11.11.2 CALIBRAZIONE OFFSET USCITE ANALOGICHE NGQ

Questa funzione permette di calibrare l' OFFSET dell'uscita analogica di ogni CANALE gestito dal controllo NGQ

Sintassi

NG_DAC_CAL(Char Chan, Long Offset)

Parametri

Chan Numero Canale (da 0 a 1)

Offset Valore dell'OFFSET

ATTENZIONE: IL VALORE DELL'OFFSET NON RIMANE MEMORIZZATO, PERTANTO AD OGNI ACCENSIONE OCCORRE IMPOSTARLO DI NUOVO.

11.12 FUNZIONI API PER LA GESTIONE DELLA sxcheda NGQx (encoder e uscite analogiche)

Queste funzioni sono relative alla PIATTAFORMA HARDWARE NGQx gestione encoder e uscite analogiche
Hardware ***NGQx***

11.12.1 NG_DAC - SCRITTURA USCITE ANALOGICHE NGQx

Questa funzione permette di pilotare l'uscita analogica di ogni canale gestito dal controllo NGQx
Le schede dispongono di convertitori digitale/analogico a 12 bit, su un fondo scala di 10V. Per cui con +2047 si ottengono 10V in uscita, con -2047 -10V in uscita.
La selezione del canale avviene con un indice che va da 0 a 1

ATTENZIONE: Per abilitare le uscite analogiche sulla scheda NGQx, è necessario impostare la proprietà ENCODER ENABLE=true dell' oggetto NGQ INIT

Sintassi

NG_DAC(Char Chan, Long Val)

Parametri
Chan Numero Canale (da 0 a 1)
val Valore dell'uscita (da -2047 a +2047)

Esempio
Variabili utilizzate:
val LONG
channel CHAR
channel = 0
val = 1024

```
ng_dac(channel, val)     'Scrive 1024 (~5V) su uscita analogica 0
ng_dac(1,512)            'Scrive 512 (~2,5V )su uscita analogica asse 1
```

11.12.2 CALIBRAZIONE OFFSET USCITE ANALOGICHE NGQx

Questa funzione permette di calibrare l' OFFSET dell'uscita analogica di ogni CANALE gestito dal controllo NGQ

Sintassi

NG_DAC_CAL(Char Chan, Long Offset)

Parametri
Chan Numero Canale (da 0 a 1)
Offset Valore dell'OFFSET

ATTENZIONE: IL VALORE DELL'OFFSET NON RIMANE MEMORIZZATO, PERTANTO AD OGNI ACCENSIONE OCCORRE IMPOSTARLO DI NUOVO.

11.12.3 NG_ENC - LETTURA CANALI ENCODER NGQx

Questa funzione permette di leggere l'ingresso encoder (quota incrementale attuale) di ogni canale gestito sulla scheda di espansione **NGQx** con una risoluzione di 32 bit. Questa funzione legge soltanto l'incremento che sarà aggiunto alla variabile passata con un puntatore. Il contatore vero e proprio quindi sarà contenuto in una variabile definita dall'applicazione e che potrà essere azzerata in qualunque momento. Per una corretta gestione degli encoder raccomandiamo di utilizzare questa funzione esclusivamente nel TASK PLC e quindi utilizzare la variabile di appoggio all'occorrenza. La selezione del canale avviene con un indice che va da 0 a 1
ATTENZIONE: Per abilitare la lettura encoder sulla scheda NGQx, è necessario impostare la proprietà ENCODER ENABLE=true dell' oggetto NGQ INIT
Sintassi

NG_ENC(Char Chan, Long *Quota)

Parametri
Chan Numero canale encoder (da 0 a 1)
val Puntatore alla variabile di tipo long in cui viene memorizzato il conteggio

Esempio
Variabili utilizzate:
```
posx LONG                ' Contatore encoder canale 0
posy LONG                ' Contatore encoder canale 1
```
Nel TASK PLC:
```
ng_enc(0,posx)
ng_enc(1,posy)
```
OVUNQUE:
```
if  posx>25000           ' Lettura encoder canale 0
        . . .
        posx=0           ' Azzeramento contatore canale 0
endif
if  posy>200000  ' Lettura encoder canale 1
        . . .
        posy=1000        ' Preset contatore canale 1
endif
```

11.12.4 NG-T0 - LETTURA INDICE DI ZERO ENCODER NGQx

Questa funzione permette di leggere lo stato dell'ingresso tacca di zero di ogni canale encoder gestito sulla scheda di espansione **NGQx**. Per la selezione del canale vale quanto detto per la lettura encoder.

Sintassi
NG_T0(Char Chan) as char

Parametri
Chan Numero canale encoder (da 0 a 1)

Valore di ritorno
Stato della tacca:
0 OFF
1 ON

ATTENZIONE: L'INGRESSO E' DIFFERENZIALE, LO STATO ON CORRISPONDE AD UNA TENSIONE SUL CH+ MAGGIORE DI QUELLA SUL CH-.

Esempio
```
if ng_t0(0)
        . . .
endif
```

11.12.5 NG_RELE - GESTIONE DEI RELE' NGQx

Questa funzione permette la gestione dei 2 RELE' presenti su ogni espansione **NGQx**

Sintassi
NG_RELE(Char Chan, char Stato)

Parametri
CH Canale relativo all'attivazione del rele' (da 0 a 1)
Stato Stato del rele':
0 OFF(contatto aperto)
1 ON (contatto chiuso)

Esempio
Variabili utilizzate:
channel UINT
stato UINT
channel = 1
stato = 1
Ng_rele(channel,stato) ***'setta il relè del secondo canale***
channel = 2
stato = 0
ng_rele(channel,stato) ***'resetta il relè del terzo canale***
ng_rele(0,1) ***'setta il relè del primo canale***

11.13 FUNZIONI API PER LA GESTIONE DELLA PIATTAFORMA NGM13-NGMEVO-NGQ-NGQx

Queste funzioni sono relative alla PIATTAFORMA HARDWARE NGM13. Quando viene selezionata questa scheda, VTB inserisce automaticamente l' OGGETTO **NGM13_INIT NGMEVO_INIT** relativo alla gestione dell' HARDWARE.
Hardware ***NGM13-NGMEVO***

11.13.1 NGM13_INIT PROPERTY-NGMEVO_INIT PROPERTY

L'oggetto fornisce una visione completa di tutte quelle che sono le opzioni software da configurare per utilizzare correttamente la ***NGM13-NGMEVO***.
In particolare permette di impostare:

- L'attivazione del protocollo di comunicazione proprietario PROMAX RPC, con il relativo baudrate
- Quanti e quali sono i canali analogici in ingresso presenti
- Gli assi passo passo da usare e quelli gestiti dall'interpolatore
- Numero schede I/O presenti

Ovviamente, per ogni singolo progetto esisterà un solo oggetto NGM init, visto che uno solo è il CN su cui gira il programma.

Proprietà principali

Link RPC port Porta RS232 su cui abilitare Il protocollo RPC per gestione HOST PC.
Valori:
0 Nessun RPC Link
1 Porta seriale SER1/PROG (in questo caso viene disabilitato il DEBUG e il download applicazione deve essere effettuato in modo manuale con i tasti BOOT/RESET della scheda ***NGM13-NGMEVO***
2 Porta seriale SER2
La scheda NGMEVO ha sempre attivo il link RPC sulla porta ETHERNET

Link RPC baud Baud rate da utilizzare per la comunicazione RPC

ADC enable mask Maschera di attivazione canali analogici in ingresso. Gestita a bit
Bit 0 canale analogico 1 abilitato (viene escluso input digitale 9)
Bit 1 canale analogico 2 abilitato (viene escluso input digitale 10)
. . .
Bit 7 canale analogico 8 abilitato (viene escluso input digitale 16)

P-P enable mask Maschera di attivazione assi Passo-Passo
Bit 0 Canale 0 abilitato
Bit 1 Canale 1 abilitato (vengono escluse uscite digitali 9 e 12)
Bit 2 Canale 2 abilitato (vengono escluse uscite digitali 10 e 13)
Bit 3 Canale 3 abilitato (vengono escluse uscite digitali 11 e 14)

P-P Interp. Mask Maschera di gestione assi Passo-Passo da interpolatore. Gestita a bit
Bit 0 Canale 0 in interpolation mode
Bit 1 Canale 1 in interpolation mode
Bit 2 Canale 2 in interpolation mode
Bit 3 Canale 3 in interpolation mode

Num. NGM-IO Numero di schede di espansione ingressi/uscite NGMIO presenti. Si ricorda che 16 in e 14 out sono già disponibili nella configurazione standard quindi questa non deve essere

considerata

L-Sync enable mask Maschera di attivazione canali L-SYNC
Bit 0 Canale 0 abilitato (viene esclusa l'uscita 1)
Bit 1 Canale 1 abilitato (viene esclusa l'uscita 2)
Bit 2 Canale 2 abilitato (viene esclusa l'uscita 3)
Bit 3 Canale 3 abilitato (viene esclusa l'uscita 4)

L-Sync Prescaler Valore del prescaler per i canali L-SYNC

11.13.2 NGQ_INIT PROPERTY NGQ e NGQx

L'oggetto fornisce una visione completa di tutte quelle che sono le opzioni software da configurare per utilizzare correttamente la ***NGQ-NGQx***
In particolare permette di impostare:

- L'attivazione del protocollo di comunicazione proprietario PROMAX RPC, con il relativo baudrate
- Gli assi passo passo gestiti dall'interpolatore

Proprietà principali

Link RPC port Porta RS232 su cui abilitare Il protocollo RPC per gestione HOST PC.
Valori:
0 Nessun RPC Link
1 Porta seriale SER1/PROG (in questo caso viene disabilitato il DEBUG e il download applicazione deve essere effettuato in modo manuale con i tasti BOOT/RESET della scheda ***NGQ NGQx***
2 Porta seriale SER2

Link RPC baud Baud rate da utilizzare per la comunicazione RPC

P-P Interp. Mask Maschera di gestione assi Passo-Passo da interpolatore. Gestita a bit
Bit 0 Canale 0 in interpolation mode
Bit 1 Canale 1 in interpolation mode
Bit 2 Canale 2 in interpolation mode
Bit 3 Canale 3 in interpolation mode

STEP Level Se ad 1 inverte il clock del PULSE per avere un ritardo maggiore tra il DIR e PULSE
Alcuni driver STEP/DIR tollerano un tempo molto alto tra uno STEP e la variazione el DIR

ENCODER Enable Se ad 1 Abilita la lettuar encoder nella scheda NGQx o le gestione dell' uscita analogica sulla scheda NGQ

11.13.3 NG_DI - LETTURA INGRESSI DIGITALI NGM13 NGMEVO

Questa funzione permette di leggere gli ingressi della ***NGM13-NGMEVO*** e delle eventuali espansioni **NGM-IO e NGM-PS**
Le schede sono identificate da un indice progressivo partendo da 0 per la prima scheda. La prima scheda è considerata la CPU (indice 0), quella più vicina a questa avrà l'indice 1 e a seguire le altre.

Sintassi

NG_DI(Char Card) as uint

Parametri
Card Numero della scheda di espansione (da 0 a 7)
Valore di ritorno
Uint Valore dei 16 ingressi gestiti a BIT
bit a 1 = ingresso ATTIVO

Ingresso	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Esempio

Variabili utilizzate:
input UINT

input = ng_di(0) ' legge gli ingressi della prima scheda I/O
input = ng_di(2) ' legge gli ingressi della terza scheda I/O

11.13.4 NG_DI - LETTURA INGRESSI DIGITALI NGQ-NGQx

Questa funzione permette di leggere gli ingressi della ***NGQ-NGQx***
L' indice della scheda di espansione è sempre 0

Sintassi

NG_DI(Char Card) as uint

Parametri
Card Numero della scheda di espansione sempre 0
Valore di ritorno
Uint Valore dei 16 ingressi gestiti a BIT
bit a 1 = ingresso ATTIVO

Ingresso	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

11.13.5 NG_DO - SCRITTURA USCITE DIGITALI NGM13-NGMEVO

Questa funzione permette di scrivere le uscite della ***NGM13-NGMEVO*** e delle sue espansioni **NGM-IO e NGM-PS**.
Le schede sono identificate da un indice progressivo partendo da 0 per la prima scheda. La prima scheda è considerata la CPU (indice 0), quella più vicina a questa avrà l'indice 1 e a seguire le altre.

Sintassi

NG_DO(Char Card, Uint Out)

Parametri
Card Numero della scheda di espansione (da 0 a 7)
Out Stato delle uscite
bit a 1 = uscita ATTIVA

Uscita			14	13	12	11	10	9	8	7	6	5	4	3	2	1
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Esempio
Variabili utilizzate:
ng_Do(0,0x7) ***' Set ta le uscite 1, 2 e 3 della NGM13***
ng_Do(1,0x31) ***' Set ta le uscite 1, 8 e 9 della prima espansione***

ATTENZIONE: **I bit 14 e 15 non sono usati.**
Le uscite da 9 a 14 della NGM13 (NON QUELLE DELLA NGMEVO) sono condivise con i canali STEP/DIR 1, 2 e 3.

11.13.6 NG_DO - SCRITTURA USCITE DIGITALI NGQ-NGQx

Questa funzione permette di scrivere le uscite della ***NGQ-NGQx*** .
L' indice della scheda di espansione è sempre 0

Sintassi

NG_DO(Char Card, Uint Out)

Parametri

Card Numero della scheda di espansione (deve essere 0)
Out Stato delle uscite
bit a 1 = uscita ATTIVA

Uscita			14	13	12	11	10	9	8	7	6	5	4	3	2	1
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

11.13.7 NOTE SULLA GESTIONE DELLE I/O

Per una programmazione più pulita e fluida è consigliabile gestire le I/O nel TASK PLC. Pertanto nel TASK, leggere ciclicamente gli ingressi scrivendoli in una variabile GLOBALE (es. Input) e scrivere le uscite leggendole da un'altra variabile GLOBALE (es. Output). Su queste variabili potranno essere definiti i singoli bit associati ai canali digitali e quindi utilizzarli all'occorrenza.

Esempio
Variabili utilizzate:
Input1 UINT
Input2 UINT
Output1 UINT
Output2 UINT
PulsanteStart BIT Input1.3
PulsanteStop BIT Input1.6
PompaAcqua BIT Output2.12

Nel TASK PLC:

```
Input1=Ng_Di(0)
Input2=Ng_Di(1)
Ng_Do(0,Out1)
Ng_Do(1,Out2)
```

OVUNQUE:

```
if  PulsanteStart
        PompaAcqua=1
endif
if  PulsanteStop
        PompaAcqua=0
endif
```

11.13.8 LETTURA CANALI ANALOGICI NGM13-NGMEVO

La scheda NGM13 possiede a bordo 8 Canali analogici **12 Bit**. In questa scheda, **l' attivazione di ogni canale analogico, esclude un ingressi DIGITALE**. L' attivazione dei canali analogici deve essere effettuata sia da software tramite l' OGGETTO **NGM13_INIT-NGMEVO_INIT** (caricato automaticamente) che da hardware.
Questa è la relazione tra ingressi digitali e analogici (il primo ingresso è il numero 1):

Analog input	Digital input
1	9
2	10
3	11
4	12
5	13
6	14
7	15
8	16

Sintassi

NG_ADC(Char Ch) as uint

Parametri

CH Numero del canale partendo da 0

Valore di ritorno

Ritorna il valore analogico letto da 0 a 4095

11.13.9 LETTURA CANALI ANALOGICI NGQ-NGQx

Le schede NGQ e NGQx mettono a disposizione da 1 a 4 Canali Analogici (1 per la scheda NGQx) a **12 Bit**.
Se vengono attivati i canali STEP/DIR sulla scheda NGQ, rimane un solo canale analogico disponibile.

Sintassi

NG_ADC(Char Ch) as uint

Parametri

CH Numero del canale partendo da 0

Valore di ritorno

Ritorna il valore analogico letto da 0 a 4095

11.14 FUNZIONI API PER LA GESTIONE CANALI STEP/DIR NGM13-NGMEVO-NGPP_NGQ

La scheda **NGM13,NGMEVO,NGQ,NG-PP** per NG35 sono equipaggiate con 4 CANALI STEP/DIR tramite i quali è possibile gestire degli assi anche in interpolazione lineare, circolare, elicoidale.
Normalmente per il loro utilizzo viene utilizzato un oggetto che ne semplifica l'implementazione. Possono infatti essere associate a oggetti INTERPOLATORI, POSIZIONATORI, CAMME, ASSE ELETTRICI, ecc.
In questo capitolo non vengono descritte le caratteristiche di tutti gli oggetti (per i quali si rimanda al manuale degli oggetti) ma solo delle funzioni necessarie per interfacciare questi oggetti alle uscite STEP/DIR. Infine alcuni esempi sono riportati per chiarire meglio come creare un'applicazione che utilizza questa risorsa hardware.
La scheda NGMEVO può utilizzare le espansioni NGMsX che portano fino a due Canali STEP/DIR aggiuntivi

ATTENZIONE
I CANANALI STEP/DIR della scheda NGPP e della scheda NGMsX possono essere usati nel modo INTERPOLAZIONE

11.14.1 PP_STEP – GENERAZIONE DEI SEGNALI STEP/DIR

Questa funzione **PP_STEP** è la PRIMITIVA che permette la generazione dei PASSI STEP sul canale indicato. Generalmente il suo utilizzo è abbinato a generatori di RAMPA e POSIZIONE derivati dai relativi OGGETTI.
Hardware ***NGM13,NG35+NG-PP-NGMEVO-NGQ-NGMsX***

Sintassi

PP_STEP(Char Chan, Long Quota)

Parametri
Chan Numero del canale STEP/DIR
NGM13 da 0 a 3 canali on board
NGQ da 0 a 3 canali on board
NGMEVO da 0 a 3 canali on board
NGMEVO da 4 a 5 canali su prima espansione NGMsX
NGMEVO da 6 a 7 canali su seconda espansione NGMsX
NGMEVO da 8 a 9 canali su terza espansione NGMsX
NGPP da 0 a 3 canali prima espansione NGPP
NGPP da 4 a 7 canali seconda espansione NGPP
.
.
NGPP da 28 a 31 canali ultima espansione NGPP
Quota Valore assoluto della posizione dell'asse step/dir

ATTENZIONE: La funzione PP_STEP và necessariamente inserita nel TASK PLC.

11.14.2 PP_PRESET – PRESET DEL GENERATORE STEP/DIR

Questa funzione consente di cambiare la posizione corrente di un generatore step/dir.
Hardware ***NGM13,NG35+NG-PP-NGMEVO-NGQ***

Sintassi

PP_PRESET(Char Chan, Long Quota)

Parametri
Chan Numero del canale STEP/DIR (vedi PP_STEP per riferimento canali)
Quota Valore della posizione che assumerà il l'asse step/dir

ATTENZIONE: PER UN CORRETTO PRESET DEGLI ASSI SEGUIRE LE INDICAZIONI RIPORTATE NELLE NOTE PIU' AVANTI

11.14.3 PP_GETPOS – LETTURA POSIZIONE ATTUALE NGPP-NGMEVO

Questa funzione consente di leggere la posizione corrente di un generatore step/dir. **Il valore letto corrisponderà al DOPPIO dei passi generati**. Questa funzione non è presente in NGM13 e NGQ dove per leggere le quote attuali ci sono 4 variabili di sistema.
Hardware ***NG35+NG-PP,NGMEVO***

Sintassi

PP_GETPOS(Char Chan) as long

Parametri
Chan Numero del canale STEP/DIR (vedi PP_STEP per riferimento canali)

Valore di ritorno
Long Posizione attuale x 2

11.14.4 LETTURA POSIZIONE ATTUALE NGM13-NGQ

Sono presenti 4 variabili di sistema contenenti la posizione attuale degli assi step/dir. **Il valore letto corrisponderà al DOPPIO dei passi generati**.

Hardware ***NGM13,-NGQ***

_SYSTEM_PXC as long Posizione attuale canale 0
_SYSTEM_PYC as long Posizione attuale canale 1
_SYSTEM_PZC as long Posizione attuale canale 2
_SYSTEM_PAC as long Posizione attuale canale 3

11.14.5 ESEMPIO DI UTILIZZO CON OGGETTO MONOAX

L'oggetto MONOAX, in pratica è un POSIZIONATORE MONOASSE molto sofisticato in grado di generare RAMPE di ACCELERAZIONE, DECELERAZIONE, di controllare la posizione di un asse, la velocità ecc.
Per rendere l'oggetto indipendente dall'hardware utilizzato questo agisce su una VARIABILE che in definitiva sarà la quota dell'asse.
Servirà quindi scrivere qualche riga di codice per interfacciare l'oggetto al tipo di hardware utilizzato ridirezionando la suddetta variabile verso un filtro **PID** per gestire assi analogici, un **PDO** per gestire assi **CANOPEN**, oppure alla funzione ***pp_step*** per gestire assi **STEP/DIR**.

Passi da eseguire:

1) Nell'oggetto **NGM13_INIT** abilitare l'interpolation mode sul canale step/dir utilizzato

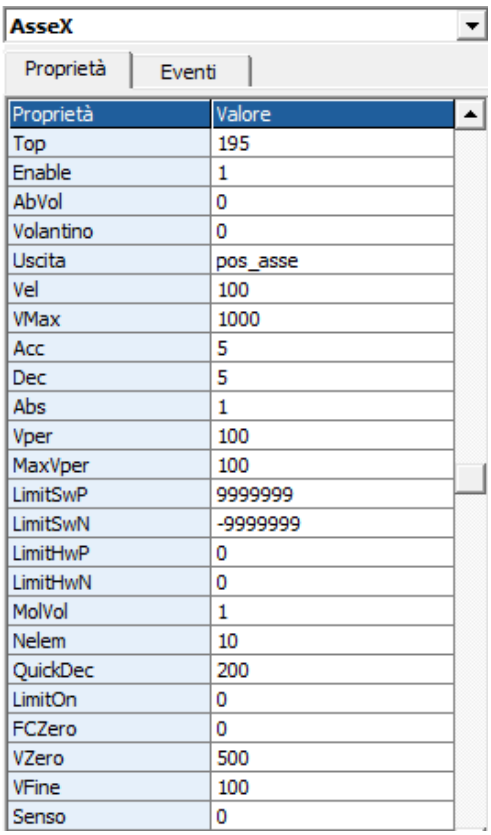
2) Inserire un OGGETTO **MONOAX** da **MOTORCONTROL → CINTERPPOS** nella **PAGINA MAIN**

3) Chiamarlo per esempio col NOME **ASSEX**

4) *Dichiarare le seguenti Variabili GLOBALI:*
Pos_Asse *Long - posizione dell'asse*
RappX *Float - rapporto tra passi generati e spostamento effettivo dell'asse*

5) *Inizializzare nella sezione INIT della MAIN la variabile RAPPX al valore desiderato (comunque non uguale 0). Un valore negativo consentirà di scambiare la direzione dell'asse.*

6) *Settare le seguenti PROPRIETA' dell'OGGETTO MONAX (esempio):*



Proprietà	Valore
Top	195
Enable	1
AbVol	0
Volantino	0
Uscita	pos_asse
Vel	100
VMax	1000
Acc	5
Dec	5
Abs	1
Vper	100
MaxVper	100
LimitSwP	9999999
LimitSwN	-9999999
LimitHwP	0
LimitHwN	0
MolVol	1
Nelem	10
QuickDec	200
LimitOn	0
FCZero	0
VZero	500
VFine	100
Senso	0

7) *Inserire nel TASK PLC il seguente CODICE VTB:*
pp_step(0, pos_asse * RappX)

8) *Codice di esempio per eseguire uno spostamento di TEST:*

```
Nel TASK MAIN:
if CONDIZIONE_DI_START
        AsseX.Vel=1000
        AsseX.quota=100000
        AsseX.start=true
        CONDIZIONE_DI_START=false       ' Per evitare start ricorsivi
endif
```

Con questo esempio la variabile pos_asse raggiungerà il valore 100000 seguendo le RAMPE programmate nell'oggetto. Nel TASK PLC il valore viene inviato tramite la funzione PP_STEP al canale 0 STEP/DIR ottenendo così un movimento dell'asse controllato in posizione e in velocità. La funzione ***pp_step*** *genera i PASSI ad ogni campionamento, pertanto a seconda del campionamento del TASK PLC, si avranno velocità diverse. Un campionamento tipico per gli assi STEP va da 2 Millisecondi a 5 Millisecondi.*

11.14.6 ESEMPIO DI UTILIZZO CON OGGETTO INTERPOLATORE

L'oggetto INTERPOLATORE genera delle traiettorie su più ASSI contemporaneamente in base al tipo di interpolazione che viene effettuata. In modo analogo all'oggetto MONAX, lavora con una variabile di appoggio che inviata in modo opportuno alla funzione ***pp_step*** *consentirà di effettuare interpolazioni sugli assi STEP/DIR.*

Passi da eseguire:

1) Nell'oggetto NGM13_INIT abilitare l'interpolation mode sui canali step/dir utilizzati

2) Inserire un OGGETTO ***INTERPOLATORE*** da ***MOTORCONTROL--> COBJINTERPOLA*** nella **MAIN**

3) Chiamarlo per esempio col NOME ***INTERPOLA1***

4) Dichiarare le seguenti Variabili GLOBALI:
PosAssi(2) *long - posizione degli assi*
Rapp(2) *Float - rapporto tra passi generati e spostamento effettivo degli assi*

5) Inizializzare nella sezione INIT della MAIN la variabile RAPP(0) e RAPP(1) al valore desiderato (comunque non uguale 0). Un valore negativo consentirà di scambiare la direzione dell'asse.

6) *Settare le seguenti PROPRIETA' dell'OGGETTO INTERPOLA1 (esempio)*

7) Inserire nel TASK PLC il seguente codice VTB:

```
pp_step(0, Interpola1.pc(0) * Rapp(0))      ' Asse X
pp_step(1, Interpola1.pc(1) * Rapp(1))      ' Axxe Y
```

8) Codice di esempio per eseguire uno spostamento di TEST:

```
Nel TASK MAIN sezione funzioni di pagina:
function MuoviAssi(Qx as Long, Qy as Long, Vel as Long)
PosAssi(0)=Qx
PosAssi(1)=Qy
Interpola1.moveto(Vel, 1, PosAssi())
endfunction
```

9) Chiamare infine la funzione con i parametri desiderati.

11.14.7 NOTE PER IL PRESET DEI CANALI STEP/DIR

Occorre prestare particolare attenzione quando si lavora con gli assi STEP/DIR o CAN OPEN in interpolation mode. Nel capitolo relativo alle funzioni di interpolazione è già stato indicato un esempio per gestire il preset con assi CAN OPEN. Di seguito saranno trattate le problematiche relative agli assi STEP/DIR.
La funzione PP_STEP lavora in modo asincrono rispetto a quelle che generano le traiettorie tipo MONOAX o INTERPOLATORE. Occorre sempre che le quote di questi oggetti siano concordi con la quota interna al generatore di passi. I numero di passi generati tramite la PP_STEP corrisponderà alla differenza tra il valore della VARIABILE GLOBALE che assume ad un dato campionamenti (TASK PLC) e quello successivo. Azzerando in modo immediato il valore di questa variabile, la funzione PP_STEP genererà quindi un numero di passi uguale al valore che aveva la variabile, e tutti in un campionamento.
Per esempio supponendo che la variabile abbia un valore di 10000, nel momento che questa viene azzerata saranno generati 10000 PASSI in un campionamento. Considerando un campionamento di 2 mSec risulterà una frequenza di 5MHz !

Per evitare che questo succeda occorre ad ogni PRESET dell'asse (quindi assegnazione di un valore alla variabile di appoggio) interrompere il generatore di STEP di PP_STEP e riattivarlo quando le quote sono concordi.
Pertanto è sempre meglio mettere sotto una condizione nel TASK PLC la generazione di PP_STEP nel seguente modo:

```
if DisableStep=false
        pp_step(0,Interpola1.pc(0) * Rapp(0))       ' ASSE X
        pp_step(1,Interpola1.pc(1) * Rapp(1))       ' ASSE Y
endif
```

Il flag DisableStep se posto a true bloccherà la generazione dei PASSI. Quindi al momento che servirà eseguire un preset degli assi, rifacendosi agli esempi di utilizzo precedenti, richiamare il seguente codice:

PRESET ASSI INTERPOLATORE:

```
DisableStep=true
pos_vect(0)=qpresetX            ' quota di preset X
pos_vect(1)=qpresetY            ' quota di preset Y
obj.preset(pos_vect())          ' preset interpolatore
pp_preset(0,qpresetX*Rapp(0))   ' preset canale step/dir 0
pp_preset(1,qpresetY*Rapp(1))   ' preset canale step/dir 1
DisableStep=false
```

PRESET ASSE MONOAX:

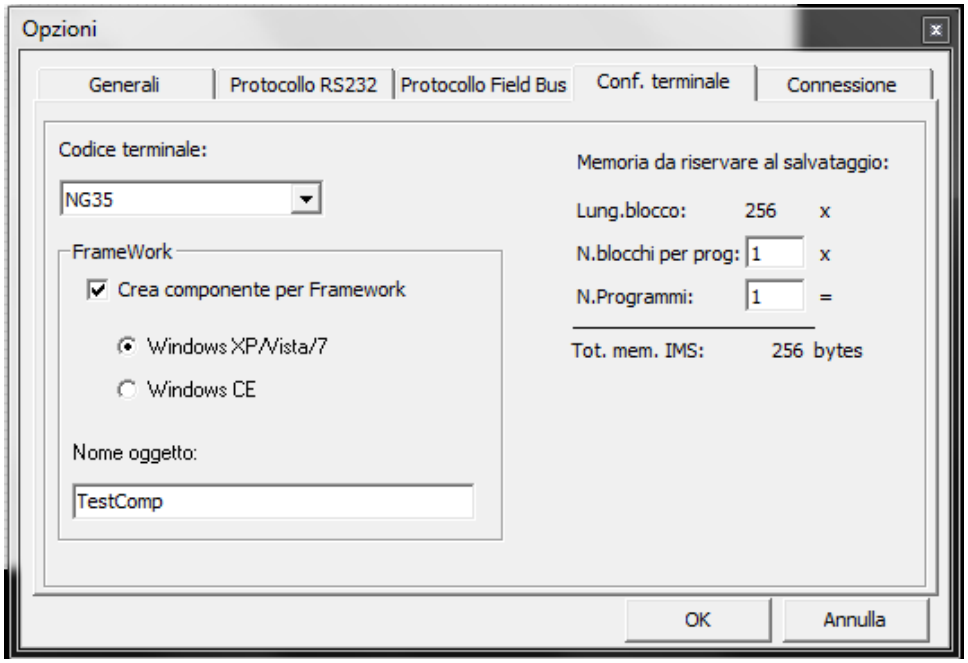
```
DisableStep=true
MONOAX.HOME= qpresetX                ' quota di preset
pp_preset(0,qpresetX*RappX)     ' preset canale step/dir 0
DisableStep=false
```

12 COMPONENTE PER FRAMEWORK

Il compilatore VTB può creare una DLL COMPONENT MODEL, che può essere importata nei PROGETTI .NET. Questo permette da parte di UN PC di avere un pieno controllo sul hardware in USO:
LETTURA/SCRITTURA VARIABILI, CHIAMATA A FUNZIONI IN REMOTE PROCEDURE CALL.
Per maggiori dettagli, si rimanda al manuale del NG Framework

12.1 Abilitazione alla creazione del COMPONENTE NGFRAMEWORK

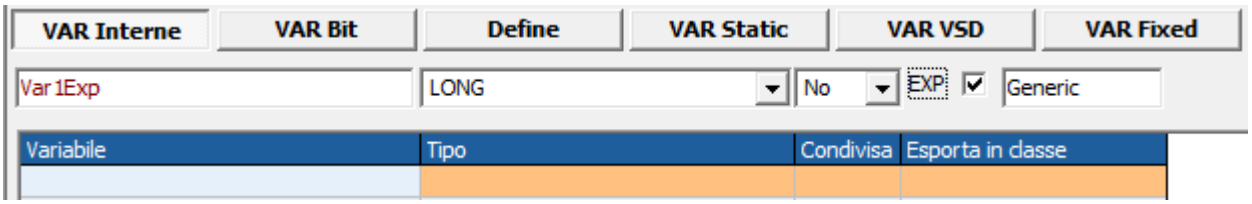
Per utilizzare il componente occorre abilitare da Opzioni di VTB la compilazione della DLL per .NET



Il componente può essere creato per sistemi WINDOWS XP/VISTA/7 oppure per sistemi Windows CE.
Il nome della DLL generata è quello che viene riportata nel nome OGGETTO.
Quindi alla fine della compilazione verrà generato una DLL NOMEOGGETTO:DLL da importare come componente nel progetto .NET

12.2 Esportazione delle VARIABILI

Si possono esportare le variabili desiderate al FRAMEWORK e quindi da parte PC leggere e scrivere queste variabili come fossero sul PC stesso.



Per esportare la variabile è necessario al momento della dichiarazione, abilitare il CHECK EXP e inserire il nome della classe di esportazione (default Generic). La classe serve esclusivamente ad avere una logica di raccolta delle variabili esportate in modo da rendere più pratico la ricerca di queste dall'applicazione PC.
La variabile in questo caso è contenuta in Generic.VAR1EXP. Questa può essere letta o scritta da parte PC in modo naturale come una normale variabile.
Si rende noto comunque che il tempo per l'operazione di SCRITTURA/LETTURA dipende dal LINK abilitato RS232 o ETHERNET.
Possono essere esportate solo le VARIABILI INTERNE anche se queste fanno riferimento ad una struttura.
In quest'ultimo caso non viene considerata la classe di esportazione, ma questa viene presa da nome della variabile (questo poiché una struttura è comunque simile ad una classe).

12.3 Esportazione di una FUNZIONE

In modo analogo alle variabili possono essere esportate anche le funzioni.
Queste vanno dichiarate in modo particolare con il seguente POSTFISSO:

function NomeFunzione(parametri) as Tipo **$_EXPORT_$ CLASSE**

endfunction

$_EXPORT_$ **Parola chiave per abilitare l' esportazione della funzione**
CLASSE **Nome della classe di esportazione dove viene trovata la funzione Es: FunzSistem**

Esempio:
function MiaFunc(Val1 As Long,Val2 As Long) as Long **$_EXPORT_$ FunzSistem**

endfunction

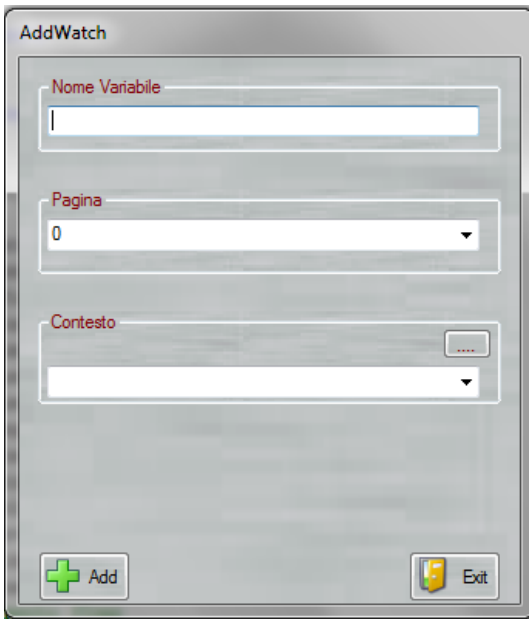
13 DEBUG APPLICAZIONE

L'applicazione VTB può essere meglio controllata utilizzando il DEBUG.NET. Questo permette di effettuare un controllo sia in lettura che scrittura di tutte le variabili VTB, di inserire BREAK POINT, ed eseguire STEP by STEP il codice. Di conseguenza rende più semplice la messa a punto dell'applicazione. Nei vecchi sistemi a 16 bit per effettuare il debug è necessario avere compilato l' intera applicazione con la Modalità DEBUG (***Strumenti-->Opzioni-->Generali***).
Il DEBUG può essere effettuato sia da linea RS232 che ETHERNET.
Quando viene utilizzata la linea seriale RS232 questa deve essere connessa sulla **SER PROG** del sistema.
ATTENZIONE: se vengono attivati protocolli seriali che occupano questa PORTA, (es MODBUS ecc.) il DEBUG non è possibile.

13.1 Barra dei pulsanti

Tramite la barra dei pulsanti è possibile effettuare il debug completo dell' applicazione.

Aggiunge una variabile alla finestra WATCH.
Questo permette di inserire una variabile che viene controllata in tempo REALE e può anche essere scritta.



Di seguito si apre la finestra sopra. Scrivendo il nome Variabile nel campo Nome VARIABILE, appare l'elenco delle variabili in ordine alfabetico presenti nel progetto, rendendo più semplice la ricerca.
La variabile può altresì essere inserita nella finestra WATCH nei seguenti modi:

Drag&Drop dal codice. Selezionare la variabile desiderata e TRSCINARLA nella finestra WATCH

```
118  if AsseX_flagb=1 && AsseX.move=0
119      AsseX_flagb=0
120      gosub AsseX_OnEndMove
121  endif
```

Click con tasto ***DESTRO del Mouse*** sulla selezione Variabile e Invia a Debug

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

Invia a Debug

Vai a Definizione

In tutti i casi la variabile viene inviata nella lista WATCH.

Pagina
Indica la pagina di appartenenza della VARIABILE (se questa è una variabile di pagina) PAGINA 0 è riferita alle variabili GLOBALI.

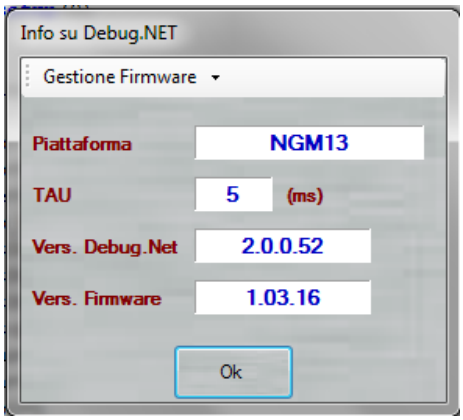
Contesto
Indica se la VARIABILE è locale in una FUNZIONE. Da questo COMBO è possibile scegliere la FUNZIONE.
Questo tipo di VARIABILI sono visibili solamente quando viene raggiunto un BREAK POINT e siamo nel giusto contesto.

Rimuove la Variabile selezionata.
La variabile selezionata nella finestra WATCH viene rimossa

Rimuove tutte le variabili dalla finestra WATCH

Rimuove tutti i BreakPoint inseriti nel progetto

Informazioni su DEBUG.NET
Oltre a visualizzare le informazioni relative a DEBUG.NET e il tipo di piattaforma in uso è possibile accedere all'aggiornamento del FIRMWARE delle varie schede. (Vedi capitolo gestione Firmware)



Stop della lettura dei vettori.
Quando si leggono vettori di GROSSE DIMENSIONI e per alcuni motivi il sistema è in TIME OUT, con questo pulsante è possibile interrompere la lettura.

Reset Scheda
Simula un RESET HARDWARE della scheda
ATTENZIONE: L' applicazione viene inizializzata

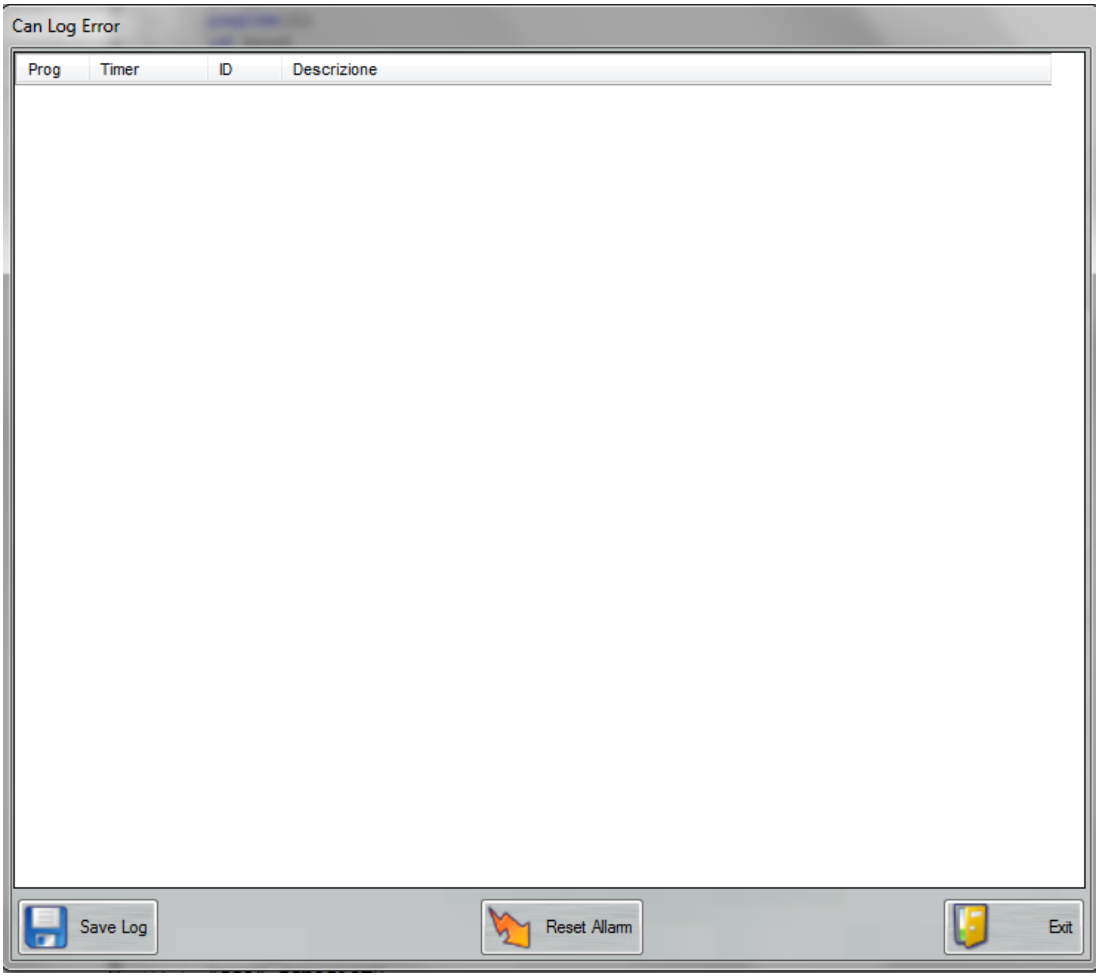
Salva la sessione delle variabili su file
E' possibile creare dei file personalizzati che contengono tutte le variabili in WATCH.
Questo permette di riprenderle successivamente in modo veloce.

Carica da File la sessione delle variabili
Permette di recuperare da file una sessione di variabili precedentemente salvata.
Il contenuto delle variabili NON VIENE INIZIALIZZATO.

Carica da File la sessione delle variabili con ripristino dei valori
Permette di recuperare da file una sessione di variabili precedentemente salvata.
Il contenuto delle variabili VIENE INIZIALIZZATO AGLI ULTIMI VALORI SALVATI.

Carica l' ultima sessione di variabili utilizzata
DEBUG:NET alla chiusura salva sempre in modo automatico l'ultima sessione di variabili in uso.
Con questo pulsante è possibile ripristinarla.

Visualizza il LOG ERRORI CANOPEN
Questo risulta molto utile quando operiamo in applicazioni CanOpen. Tramite questa utility è possibile verificare se la linea CANBUS è corretta oppure se si presentano errori.



Gli errori vengono visualizzati in ordine TEMPORALE con codifica del CanOpen.
E' possibile salvare il file LOG per poterlo analizzare in seguito. Gli errori vengono campionati dalla piattaforma HARDWARE in tempo REALE.

Oscilloscopio

Attiva l' oscilloscopio digitale (vedi capitolo oscilloscopio)

Opzioni di DEBUG.NET

Permette di settare alcune opzioni del DEBUG

Ritardo Lettura Pacchetti (Ms)

Questa opzione se > di ZERO inserisce il ritardo corrispondente tra lettura di un pacchetto e l'altro. Quando si utilizza un LINK RS232 questo NON E' NECESSARIO.

Può essere utile in ETHERNET poiché data l'elevata velocità di questo una lettura continua potrebbe creare problemi all'applicazione VTB (rallentarne il flusso)

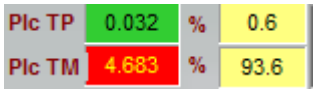
Pertanto si consiglia con ETHERNET di inserire perlomeno 1 Ms di ritardo

Visualizzazione ESADECIMALE/DECIMALE

Se questo CHECK è attivato tutti i valori numerici delle variabili vengono visualizzati nel formato ESDECIMALE.

Visualizzazione Carattere ASCII

Se questo check è attivato e la variabile contiene un numero che rientra nella tabella ASCII, viene visualizzato anche il carattere corrispondente (utile per vettori che contengono STRINGHE alfanumeriche).



Tempi in Millisecondi di esecuzione del TASK PLC e relativa percentuale di utilizzo della CPU

Se il sistema rileva che la percentuale di utilizzo della CPU si avvicina a valori CRITICI, effettua dei LAMPEGGI ROSSI di avvertimento.

Run programma da BreakPoint (da tasto F5)

Quando raggiunto un BreakPoint, permette di riprendere l' esecuzione normale del Programma.

Esecuzione Istruzione/Routine (da tasto F10)

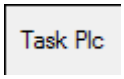
Quando raggiunto un BreakPoint questo tasto esegue un ISTRUZIONE o Funzione alla volta, STEP By STEP. Eventuali chiamate a funzioni vengono eseguite ***senza entrare in queste.***

Esecuzione Istruzione (da tasto F11)

Quando raggiunto un BreakPoint questo tasto esegue un ISTRUZIONE alla volta, STEP By STEP. Eventuali chiamate a funzioni vengono eseguite ***entrando all' interno di queste.***

Cerca nel file

Cerca un testo nel file attivo

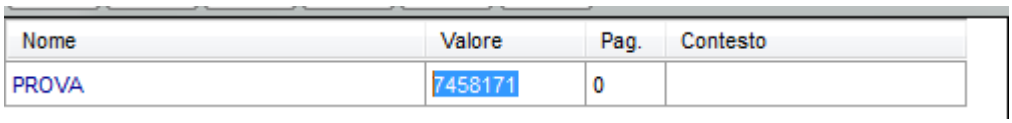


Visualizza contenuto del TASK PLC

ATTENZIONE: ***nel TASK PLC non è possibile inserire BreakPoint.***

13.2 Scrittura di una variabile

Per scrivere una variabile è necessario che questa si trovi nella finestra WATCH.
Fare doppio click sul campo VALORE e scrivere il valore desiderato seguito da INVIO

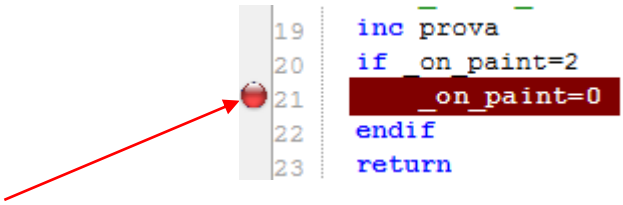


La variabile viene immediatamente scritta sul sistema.
Se la variabile è di tipo BIT fare doppio click per commutare il valore da TRUE a FALSE e viceversa.

13.3 Inserimento/Rimozione di un BreakPoint

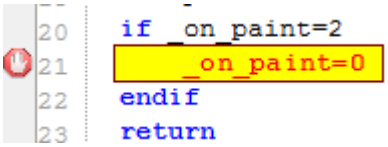
L'inserimento di un BreakPoint permette di interrompere il programma in un punto desiderato.
Da questo punto una volta raggiunto il breakpoint è possibile eseguire STEP by STEP il programma stesso verificandone la correttezza.
ATTENZIONE: I BreakPoint non possono essere inseriti nell'hardware NGM13.

Tramite il COMBO Select File, selezionare la pagina di codice desiderata.
Portarsi con il Mouse sulla linea desiderata all' estrema sinistra di questa e fare click con il tasto sinistro del mouse.



Punto di click con il mouse

Quando il il programma passa dal quel punto, la barra da MARRONE diventa GIALLA e l' esecuzione si INTERROMPE. A questo punto, è possibile gestire un nuovo RUN con il pulsante ***Run programma da BreakPoint (F5),*** oppure un esecuzione Step by Step.



Per rimuovere un BreakPoint cliccare nuovamente sul BreakPoint inserito

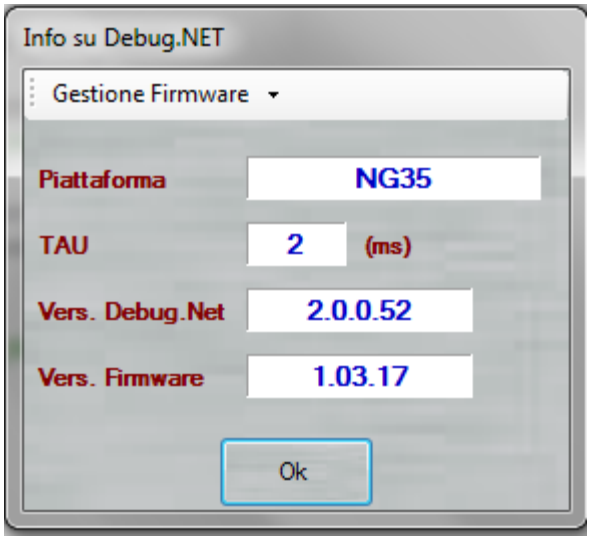
ATTENZIONE: Anche se raggiunto un BreakPoint il TASK PLC non viene mai interrotto, permettendo quindi l' esecuzione del DETERMINISMO. Tuttavia è possibile interrompere il programma in punti CRITICI, soprattutto quando si opera su macchine. In questo caso occorre prestare molta attenzione ad eventuali danni collaterali.

13.4 Aggiornamento del Firmware della piattaforma tramite DEBUG.NET

Oltre al debug dell'applicazione, tramite DEBUG.NET, è possibile aggiornare il FIRMWARE (sistema operativo) della piattaforma in USO. Ovviamente la piattaforma deve essere connessa al PC.

ATTENZIONE: L' aggiornamento del FIRMWARE è possibile solamente da canale RS232.

Tramite il pulsante INFORMAZIONI viene attivata la seguente finestra



Dal Menù Gestione Firmware abbiamo le seguenti Opzioni:

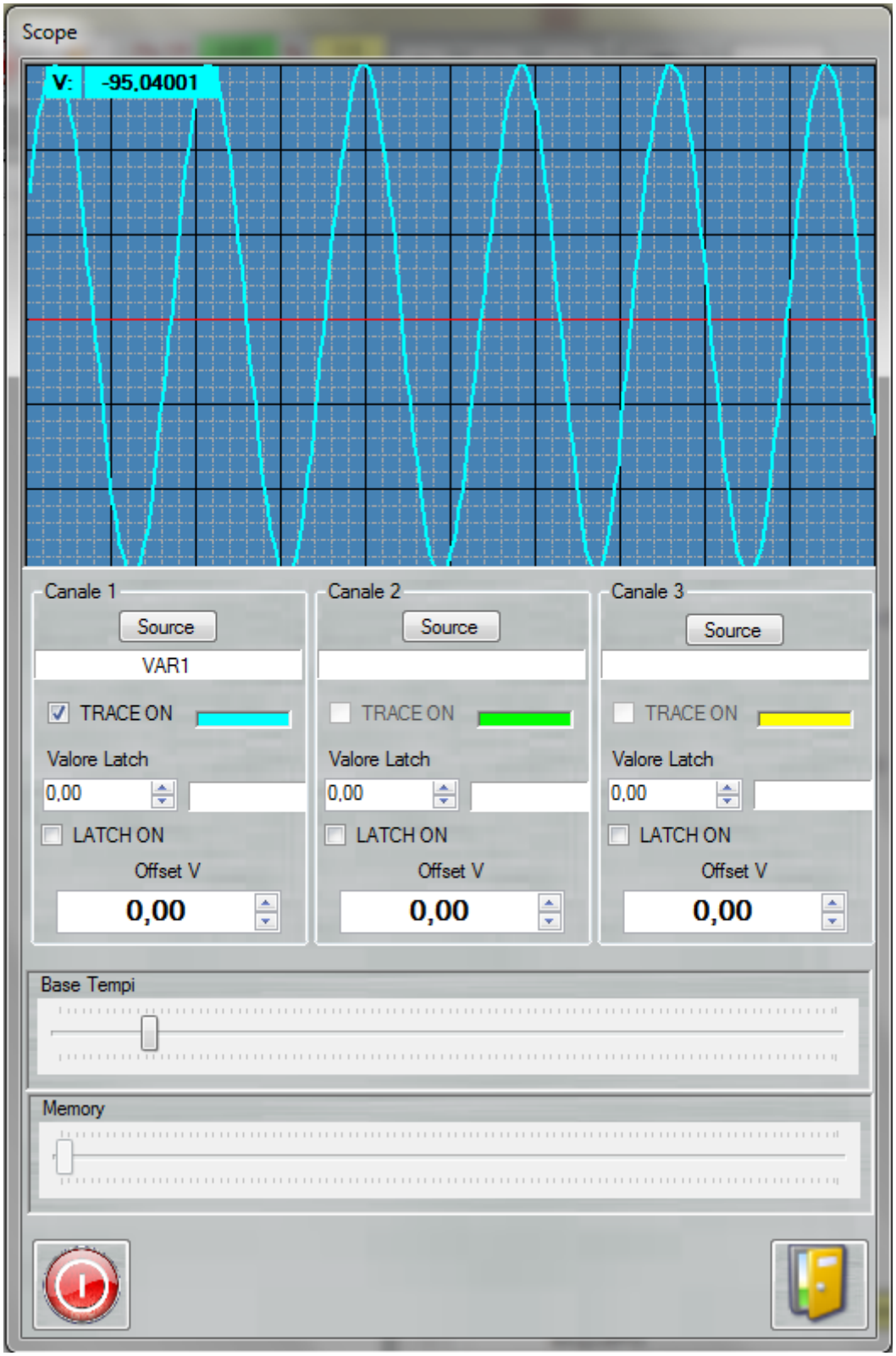
Update dal server
In questo caso occorre avere una connessione INTERNET attiva. L' applicazione controlla se sul SERVER PROMAX è presente un versione più recente del FIRMWARE e ne propone l' aggiornamento.

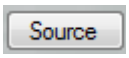
Update da File
Permette di caricare un file .SREC del firmware relativo alla scheda.
ATTENZIONE: In questo caso non viene fatta nessun controllo sulla revisione del firmware e sulla sua compatibilità con la piattaforma in uso.

ATTENZIONE: Durante la fase di aggiornamento del FIRMWARE l'applicazione viene BLOCCATA ma NON PERSA.

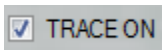
13.5 Oscilloscopio Digitale

DEBUG.NET mette a disposizione un applicazione OSCILLISCOPIO che può risultare utile al DEBUG delle applicazioni. L'OSCILLISCOPIO DIGITALE è in grado di monitorare variabili che sono presenti nella ***FINESTRA DI WATCH****.*
L' oscilloscopio mette a disposizione 3 CANALI.





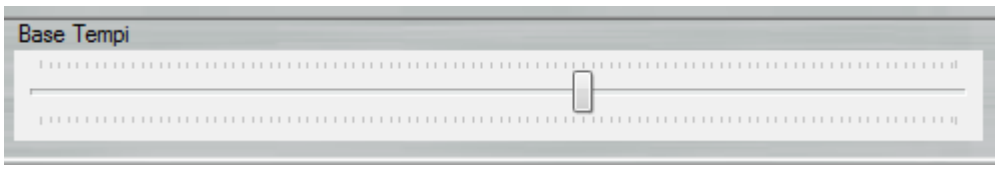
Sceglie la variabile da monitorare sul relativo Canale.
La variabile deve essere presente nella finestra WATCH.



Abilita o disabilita la TRACCIA del relativo canale.

Inserisce un OFFSET sulla traccia relativa.

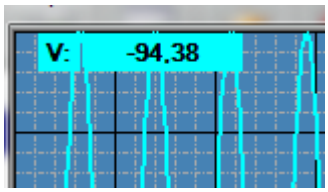
Abilitando il LATCH, quando la variabile supera il valore Latch impostato, la traccia viene CONGELATA.



Imposta la base dei TEMPI per tutte le tracce.

Quando l' oscilloscopio è in stato di OFF, permette di scorrere la memoria campionata.

ON/OFF oscilloscopio.



Posizionando il mouse su un punto qualsiasi della traccia, viene visualizzata l'ampiezza in quel punto (valore della VARIABILE MONITORATA).

14 CONFIGURATORE CANOPEN

Il Configuratore CanOpen, è un tool che è incluso nel pacchetto di VTB IDE e permette di configurare la rete CanOpen collegata ad un CN Promax.
Con questo software, lo sviluppatore può facilmente descrivere il numero di nodi slave ei vari PDO presenti nella rete.

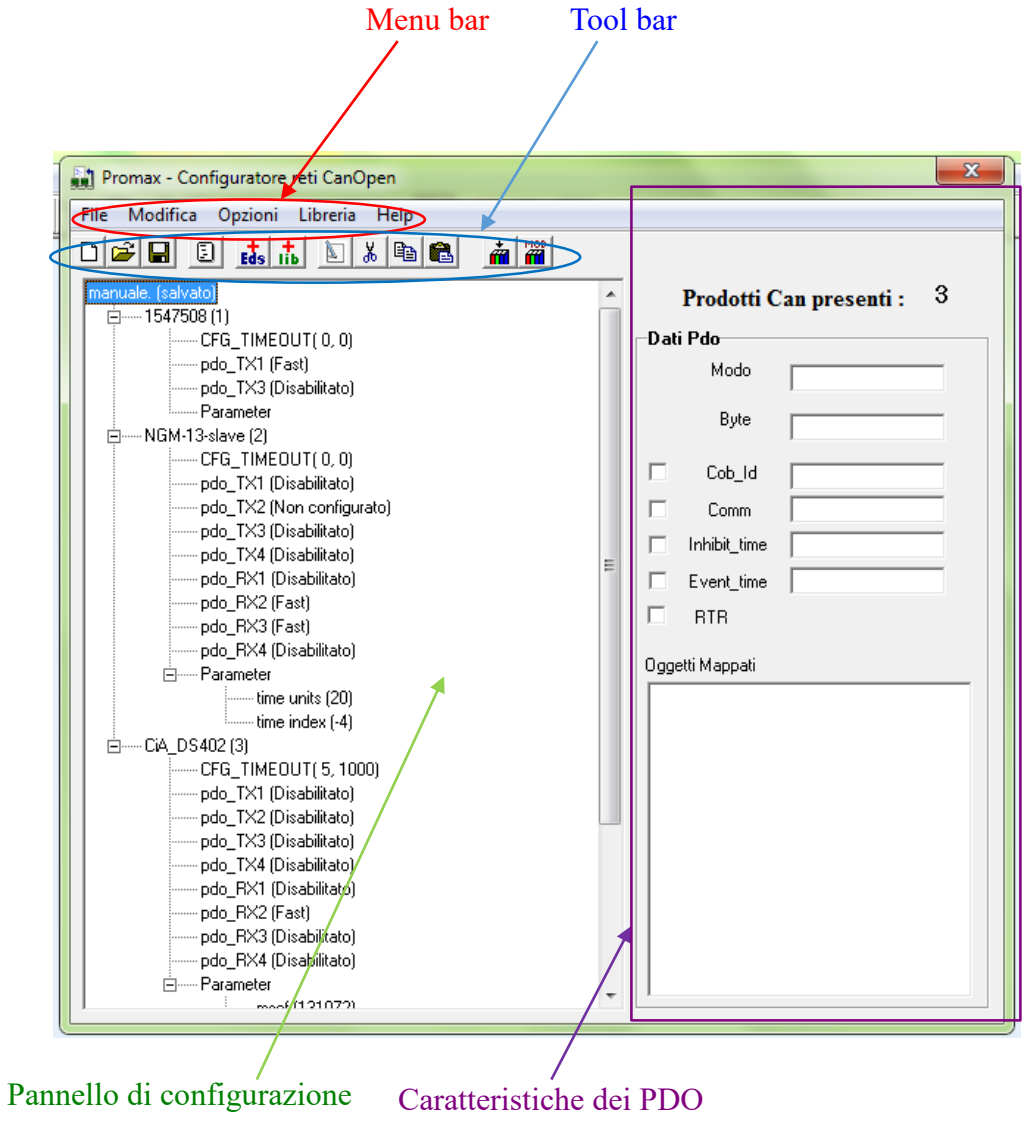
14.1 Note sui dispositivi utilizzati

Questo software può lavorare con le schede Promax CanOpen Slaves oppure con Slaves di terze parti.
Il configuratore può caricare un file EDS standard per utilizzare tutte le proprietà del dispositivo.

14.2 Note sul significato della configurazione CanOpen

La configurazione di una rete CanOpen serve per inizializzare i vari Slaves programmando i PDO (Process Data Object) necessari, istruendo il Master ad un corretto scambio dati con i dispositivi Slaves.
I dati e le strutture che possono essere configurati all'interno di ogni Salve, dipendono dal firmware del dispositivo, non dal software di configurazione CANopen e sono descritti nel file EDS del dispositivo.
Una Slave sconosciuto può essere anche inizializzato con file EDS di un altro dispositivo simile.

14.3 Interfaccia



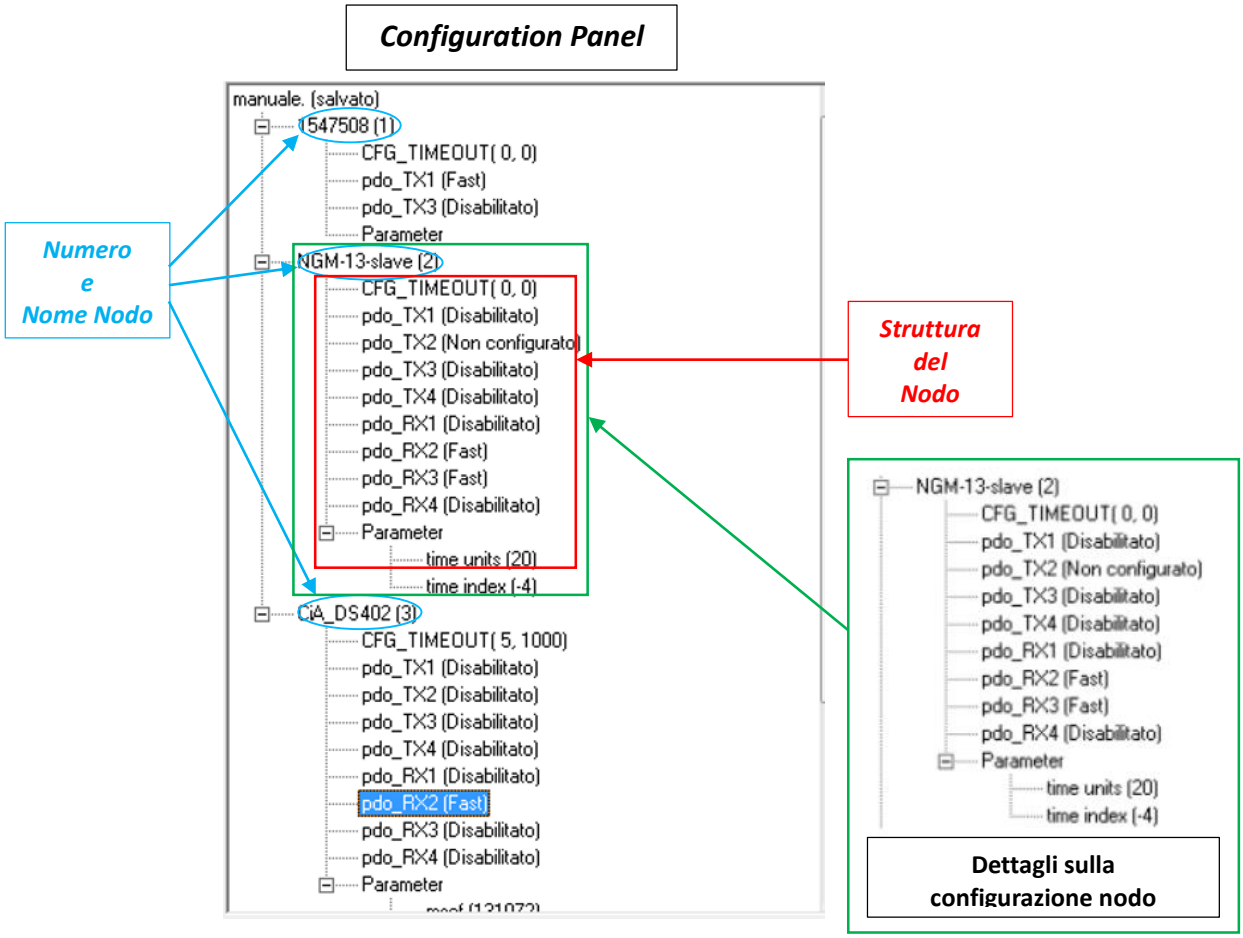
Il Configuratore CanOpen , appare come applicazione standard stile di Windows, con menù classici e barra degli strumenti.
Poi vi è un pannello di configurazione, che mostra la reale configurazione con il nome del file e vari nodi inclusi.
L'ultima sezione può mostrare una sintesi delle caratteristiche del PDO selezionato

14.4 Tool bar

	**Nuova Configurazione** - Da menu ***File → Nuova*** Crea una nuova configurazione.
	**Apri Configurazione** - Da menu ***File → Apri*** Apre una configurazione esistente
	**Salva Configurazione** - Da menu ***File → Salva*** Salva il progetto corrente e crea il fle .cop linkabile in nella' applicazione VTB. Vengono analizzati anche il numero dei PDO configurati e visualizzato il tempo della BANDA PASSANTE. **Attenzione questo tempo non deve eccedere il campinonamento impostato nel TASK PLC (meglio se un 10% inferiore)**
	**Guarda file di Configurazione –** Da menu **Opzioni → Mostra file finale** Visualizza il file di configurazione che viene generato per il progetto VTB
	**Aggiungi nodo da EDS –** Da menu **Modifica → Aggiungi nodo** Aggiunge un nodo CanOpen prendendo la configurazione dal file EDS
	**Aggiungi nodo da libreria –** Da menu **Libreria → Prendi nodo da libreria** Aggiunge un nodo CanOpen da libreria (oggetti già preconfigurati)
	**Modifica nodo** – Da menu **Modifica → Modifica nodo** Modifica il nodo
	**Taglia nodo** – Da menu **Modifica → Rimuovi nodo** Elimina il nodo selezionato
	**Copia nodo** – Da menu **Modifica → Copia nodo** Copia il nodo selezionato negli appunti. Viene reso disponibile alla funzione Incolla
	**Incolla nodo** – Da menu **Modifica → Incolla nodo** Aggiunge il nodo precedentemente copiato chiedendo un nuovo indirizzo
	**Esporta in libreria** – Da menu Libreria → **Salva nodo in libreria** Esporta in libreria la configurazione del nodo selezionato rendendola disponibile e giaà configurata
	**Modifica libreria** – Da menu Libreria → **Modifica nodo in libreria** Modifica un nodo in libreria

14.5 Pannello di configurazione

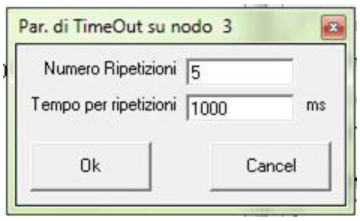
Il pannello di configurazione è la vera area di lavoro dell'applicazione. Essa mostra la configurazione attuale e rende possibile modificarla.



La struttura ad albero mostra la composizione della rete CanOpen configurata.
La radice è il nome del progetto e ogni ramo rappresenta un nodo
Nell'esempio, tratto da una configurazione reale, si ha:
NGM13 è il tipo di slave;
ID nodo è 2;
Dopo, vi è una funzione molto utile: la configurazione time-out (vedi Errore: sorgente del riferimento non trovata).
Poi possiamo vedere la struttura definita del dispositivo che stiamo usando, presa dal suo file EDS.
Qui abbiamo un dispositivo che dispone di quattro PDO TX (trasmissione) e quattro PDO RX (ricezione).
La direzione della comunicazione è definita dal punto di vista dello slave, significa che TX va da slave a master mentre RX dal master allo slave.
Ogni PDO viene mostrato con la sua modalità attiva (vedi Errore: sorgente del riferimento non trovata).
L'ultima informazione che possiamo vedere, sono parametri che la funzione imposta dopo aver effettuato la configurazione dello slave.

14.5.1 Time-out di configurazione

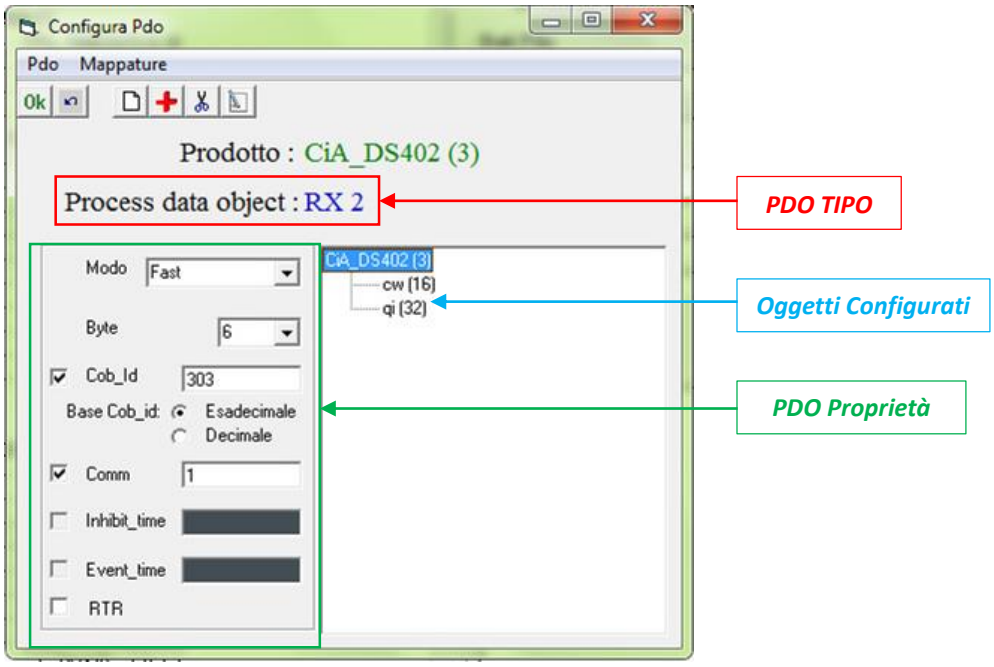
In situazioni normali, quando la macchina si avvia, è usuale avere il CNC che ha un tempo di START UP molto breve, mentre per gli slaves il tempo di inizializzazione può essere più lungo.
Il master quindi è costretto ad aspettare che gli slaves siano effettivamente pronti a ricevere la configurazione. Con il TIME OUT di CONFIGURAZIONE è possibile ovviare a questo inconveniente, aspettando che lo slave sia effettivamente pronto.
Il time out è composto da tempo di attesa e dal numero di tentativi da effettuare tra l' intervallo di tempo. Se alla fine lo slave non risponde e non è pronto a ricevere la configurazione, viene considerato come NON PRESENTE NELLA RETE.



"Numero Ripetizioni" *Indica il numero di tentativo di connessione con lo slave*
"Tempo per ripetizioni" Indica il tempo in Ms tra una ripetizione e l' altra

14.5.2 Cambiare la configurazione di un PDO

Ora vediamo come possiamo cambiare o aggiungere una configurazione PDO in un dispositivo slave



Per modificare la configurazione PDO, possiamo fare un doppio clic sul PDO desiderato, oppure selezionare ***"modifica → Configura PDO"*** sul menu.
Nella prima riga del modulo che apparirà, possiamo leggere il nome del dispositivo, quindi la PDO tipo-id.
La prima proprietà che può essere settato, è il ***"Modo"***, che definisce la modalità PDO di lavoro.
Settato come ***"fast"*** della PDO sarà mandato o ricevuto ad ogni tempo di campionamento.
Lavorare come "lento", può essere utile per scaricare il traffico di rete. Quando c'è più di un PDO lento, in ogni tempo di campionamento sarà inviare, da Master a Slave o viceversa, solo uno di questo. Significa che se abbiamo 3 lento DOP, avremo tutti i dati aggiornati solo dopo 3 volte campione. Chiaramente, questa modalità può essere utilizzata solo con dati che cambiano basso tasso.
"Disabilitato", disabilitato, significa che il PDO viene disabilitata. è

Sommario