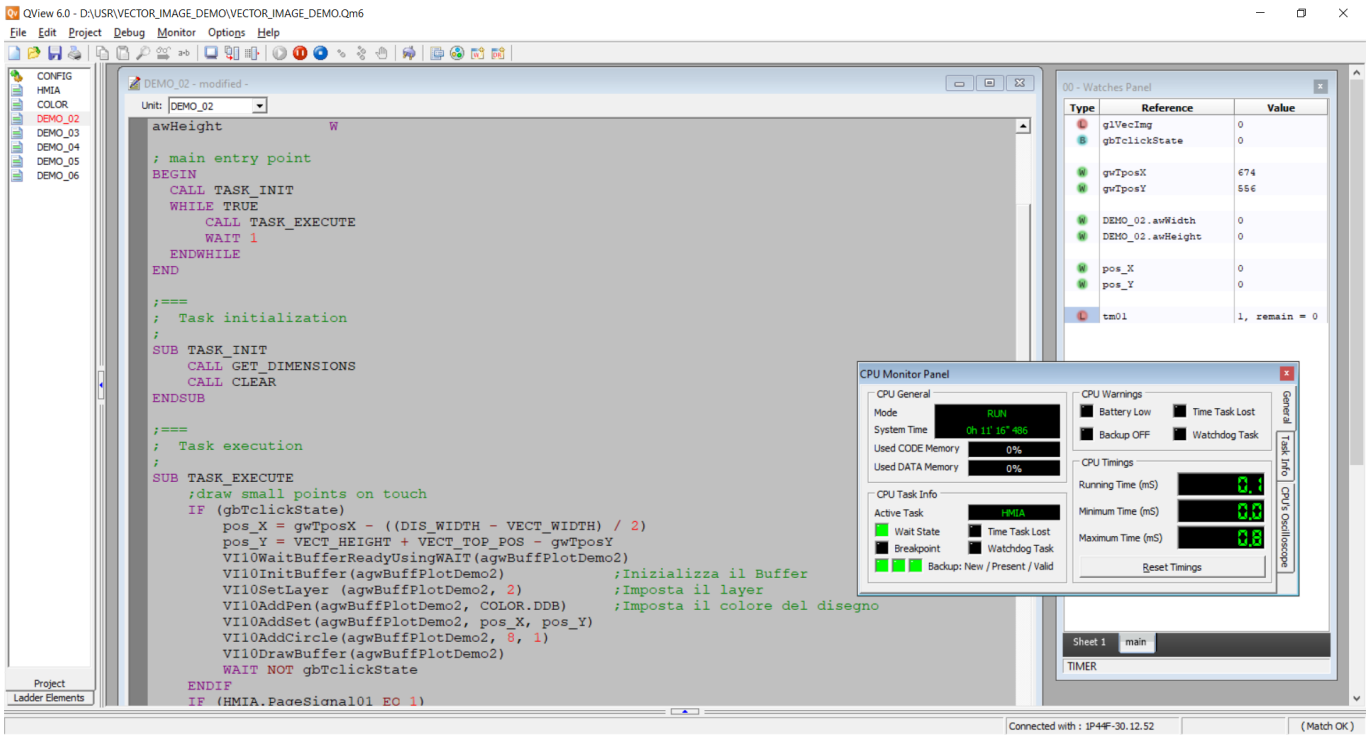
Sommario

QVIEW 6.0



1. Introduzione

L'ambiente di sviluppo QVIEW (QMOVE Viewer), il linguaggio QCL (QEM Control Language) e il linguaggio LADDER (logica contatto), sono gli strumenti necessari per lo sviluppo, il debug e la modifica dei progetti per il sistema QMOVE. La potenzialità di questa combinazione di hardware e software permette di sviluppare controlli per l'automazione industriale, con particolare riguardo alla gestione della movimentazione (elettrica, idraulica, CNC o ON / OFF).

1.1 QVIEW: QMOVE VIEWer

Come ogni linguaggio strutturato e compilato, anche il QCL ed il Ladder necessita di un editor per digitare le righe di codice e di un compilatore che le traduca in linguaggio macchina controllandone la sintassi; QVIEW è l'ambiente di sviluppo che, oltre a supportare il linguaggio QCL ed il linguaggio Ladder, permette di trasferire il programma nella CPU del sistema QMOVE, di monitorare e modificare tutte le variabili ed i parametri del programma ed eventualmente interromperne l'esecuzione per comprendere e correggere eventuali funzionamenti non desiderati (DEBUG).

1.2 QCL: QEM Control Language

Il *QCL* è un linguaggio di tipo strutturato la cui sintassi e i cui comandi risulteranno familiari a chi si occupa già di programmazione o che comunque ha avuto modo di fare esperienza con un linguaggio ad alto livello. Chi invece si addentra per la prima volta nel mondo dei linguaggi strutturati potrà trovare in questo manuale un valido supporto di apprendimento.

1.3 Ladder

Il *Ladder* è un linguaggio a contatti logici il cui uso sarà familiare a chi si occupa di programmazione PLC. Chi invece si addentra per la prima volta nel mondo ladder dovrà provvedere ad integrare il presente manuale con un corso PLC generico.

1.4 I Device

I *device* sono delle funzionalità messe a disposizione del programmatore. Essi svolgono le funzioni tipiche dell'automazione industriale, dalla lettura di un ingresso analogico, alla lettura di un segnale proveniente da un traduttore di posizione, fino ad eseguire dei veri e propri posizionamenti di assi. Ogni device è dotato di un manuale specifico, si rimanda perciò alla documentazione relativa.

1.5 QPaint

Una applicazione di prodotti QEM in una automazione industriale può aver bisogno sia di un progetto sviluppato in Qview (per il controllo e l'automazione) e sia di un progetto per tutto quello che riguarda l'interfaccia con l'operatore. QPaint è l'ambiente per lo sviluppo dell'interfaccia grafica per i terminali operatore QEM. I due progetti sono legati tra di loro perché fanno riferimento alla stessa struttura dati (variabili, array, ecc.).

2. Introduzione all’ambiente

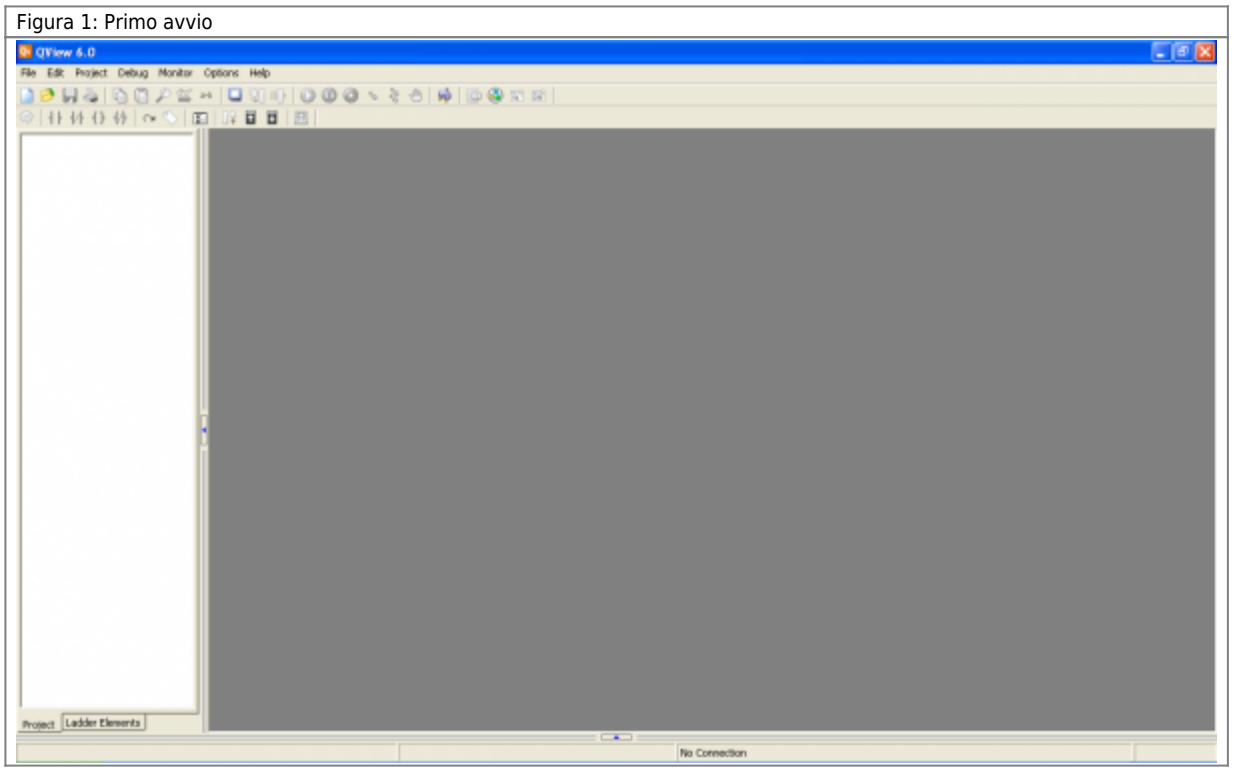
In questa sezione verrà introdotto l’ambiente di sviluppo (QVIEW) presentando un semplice esempio. L'esempio è un supporto alla sperimentazione pratica durante l’apprendimento.
Le fasi fondamentali che verranno sviluppate possono essere riassunte in:

- Apertura e creazione del progetto di esempio.
- Compilazione del progetto di esempio.
- Connessione del PC al QMOVE.
- Trasferimento del progetto compilato nella CPU (Download).
- Monitoraggio del funzionamento.

2.1 Apertura e creazione del progetto di esempio

QVIEW 6.0 è uno dei software del pacchetto QWorkbench. Una volta installato e avviato il QVIEW 6.0, la schermata iniziale appare come in Figura 1.

Figura 1: Primo avvio



- Nella parte superiore dello schermo sono posizionati i vari menu da cui è possibile effettuare tutte le operazioni consentite.
- Appena sotto vi è la barra degli strumenti dove si può avere accesso a tutte le operazioni di base più usate.
- Nella parte inferiore dello schermo si trova una barra di stato per un riepilogo delle informazioni importanti.
- Sulla zona grigia sarai in grado di mettere tutte le finestre necessarie per sviluppare, eseguire il debug e monitorare (Area di lavoro).
- Nella zona bianca è possibile elencare le unit che compongono il progetto o l'elenco dei elementi ladder è possibile utilizzare.

2.1.1 Creare un nuovo progetto

Per creare un nuovo progetto devi selezionare:

- File - New Project

e scrivere il nome del progetto come “FirstApp”. Viene visualizzata la finestra per avere alcune informazioni sul progetto. Per ora è possibile lasciare perdere. Sarà possibile compilare questo modulo in un secondo momento. Ora è possibile salvare il progetto per creare il file di progetto. Seleziona:

- File - Save

e scegliere una directory dove vuoi salvare il progetto. Ora bisogna creare le due *units* fondamentali per creare un progetto minimale che possa essere compilato. Selezionare:

- File - Add Unit - Configuration Unit

lasciare il *Unit Name* come CONFIG e premere OK. Come si vede una nuova unit è apparsa nell'area bianca e una finestra di editor è apparsa nella *Work Area*. In questa nuova finestra sarà possibile vedere le linee di codice che compongono la unit. Andiamo ora a modificare questa unit. La Configuration Unit è unica nel progetto (non ne possono essere aggiunte altre di questo tipo) ed è utilizzata per dichiarare le variabili o le strutture dati del progetto, per dichiarare quali device si vogliono usare e per dichiarare per quale tipo di hardware si sta sviluppando il progetto.

2.1.1.1 Configuration unit

In questo momento è necessario sapere che la unit di configurazione è dove:

- si dichiarano le costanti, variabili, matrici, ingressi digitali e uscite digitali. Ognuno di questi simboli è leggibile e scrivibile in tutto il progetto (in seguito vedremo come si possa dichiarare le variabili con visibilità locale);
- si dichiara l'hardware che si sta utilizzando;
- si dichiarano tutti i *devices* che si desidera utilizzare nel progetto.

Ora è necessario inserire alcuni cambiamenti della unit di configurazione. Cercare la parola chiave *SYSTEM* nella unit (è possibile usare lo strumento di ricerca con CTRL+F). Verrà trovato questo codice:

```
;-------------------------------------------------------------------
; SYSTEM variables definitions
;-------------------------------------------------------------------
SYSTEM
```

cambiare nel modo seguente:

```
;-------------------------------------------------------------------
; SYSTEM variables definitions
;-------------------------------------------------------------------
SYSTEM
   count   L   ;This is a variable use as a counter
```

Con questa sintassi è stata dichiarata la variabile *count*, il suo tipo è *L (Long)* e essa appartiene al gruppo *SYSTEM*.

- *SYSTEM*: è un gruppo di variabili che memorizzano il loro valore allo spegnimento del sistema. Alla riaccensione tale valore viene mantenuto.
- *Long*: è un tipo di variabile che occupa 4 bytes in memoria. Variabili di questo tipo possono contenere valori compresi tra -2147483648 and 2147483647;
- *count*: è il nome assegnato alla variabile.

Tutte le linee di codice che iniziano con “;” sono commenti del programmatore. Esse non hanno effetto esecutivo sul progetto.

Un'altra operazione è l'indicazione della composizione del QMOVE sistema. Prima di tutto bisogna cercare la parola chiave *BUS*.

```
;-------------------------------------------------------------------
; BUS configuration
;-------------------------------------------------------------------
BUS
```

Cambiare come segue:

```
;-------------------------------------------------------------------
; BUS configuration
;-------------------------------------------------------------------
BUS
   1  1P51  30
   2  .     .
   3  IMG8F .
   4  .     .
```

Queste righe di codice dipendono dal tipo di hardware per cui sta sviluppando il progetto. Nella documentazione di installazione del controllo qmove in uso, è possibile ottenere le informazioni relative alla configurazione della sezione BUS.

2.1.1.2 QCL unit

Aggiungere una nuova unit al progetto. Seleziona:

- File - Add Unit - QCL Unit

lasciare il nome della unit a UNIT1 e premere OK. Otterremo una seconda unit con il seguente codice già scritto:

```
MAIN:
  WAIT 1
  JUMP MAIN
END
```

cambiare in questo modo:

```
MAIN:
  count = count + 1
  WAIT 1
  JUMP MAIN
END
```

Le unit di questo tipo possono essere più di una in un progetto. Il codice compreso tra l'etichetta *MAIN* e l'istruzione *JUMP MAIN* è ripetuto continuamente come in un ciclo infinito. La linea di codice *count = count + 1* incrementa di 1 il valore della variabile *count* ad ogni ciclo.

2.1.2 Compilazione del progetto di esempio

Prima di trasferire il progetto nella CPU, bisogna sempre compilarlo. E' possibile avviare la compilazione dal menù:

- Project - Compile

Alla fine della compilazione, alcuni messaggi ti informeranno dei risultati ottenuti. Nell'esempio che abbiamo appena creato il risultato non dovrebbe avere errori e quindi otterremo *Project compiling OK!*.

2.2 Connessione del PC al QMOVE

Prima di poter aprire il collegamento è necessario che sia attivo il gestore delle risorse QRM (Qem Resurces Manager). Si trova tra le icone in basso a sinistra della barra degli strumenti di Windows. La sua icona è  . Se questa icona è presente significa che il QRM è attivo. Altrimenti sarà necessario avviarlo trovandolo nel pacchetto QWorkbench.
La connessione può avvenire sia via seriale tramite il convertitore IQ009 convertitore IQ009, sia tramite il collegamento tra la porta Ethernet del PC e la porta Ethernet del Qmove (per i modelli che ne sono provvisti).

Per poter trasferire il progetto compilato alla CPU bisogna connettere il PC al sistema QMOVE. Selezionare:

- Options - Open Connection

Appare la finestra “Connection Resources”. Cliccare con il tasto destro su *Serial Com* e selezionare *Edit Resource Property…*.
Per un semplice collegamento seriale è necessario solo assicurarsi che il numero della porta COM sia corretto. Per quanto riguarda la velocità di comunicazione, lasciare per ora *AUTODETECT*.
Per un collegamento Ethernet invece è necessario selezionare *TCP/IP* e specificare il *TCP/IP Host & Port* con il suo indirizzo IP e il numero della porta a 5001. Per configurare il QMOVE con uno specifico indirizzo IP si deve usare una delle utility installate nel QWorkbench: QConfigurator-1 oppure QConfigurator-2.

Ora fare un doppio click sulla risorsa da usare per il collegamento e il messaggio *No connection* sulla barra in basso dovrebbe cambiare in *Connected with...* seguito dal codice del modello hardware utilizzato.

2.3 Trasferimento del progetto compilato nella CPU (Download)

In questo momento si è pronti per trasferire il progetto compilato nella CPU. Selezionare:

- Project - Download

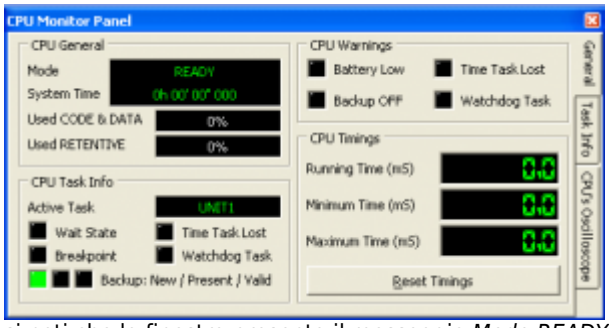
e il progetto verrà trasferito. Durante il trasferimento una finestra informa sullo stato del trasferimento.

2.4 Monitoraggio del funzionamento

Il sistema ora è pronto per mettere in esecuzione l’applicativo appena scaricato. Seleziona:

- Monitor - CPU

Appare la finestra

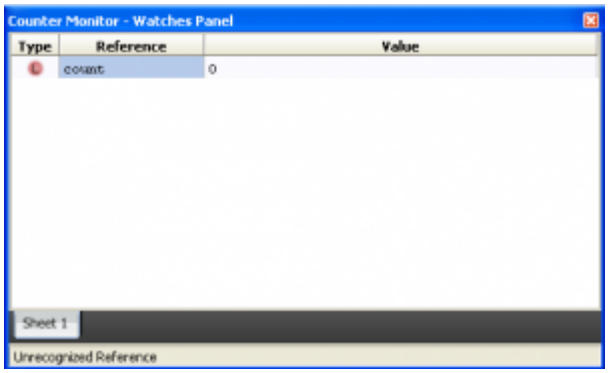


si noti che la finestra presenta il messaggio *Mode READY*.

Selezionando il comando

- Debug - Run,

l'applicativo scaricato viene messo in esecuzione; lo stato della CPU diventa RUN. Cliccare ora sul tasto  e inserire un nuovo *Watches Panel* con il nome *Counter Monitor* (oppure un nome a scelta del programmatore). Nella prima riga del *Watches Panel* inserire il nome dell'unica variabile del nostro progetto *count*.



Quando la CPU è in RUN vedremo il valore della variabile che si incrementa.

3. Introduzione alla programmazione

3.1 Dichiarazioni

Le dichiarazioni possibili in un progetto per QMOVE possono essere le seguenti:

- dichiarazione dei simboli;
- dichiarazione dei device;
- dichiarazione del hardware utilizzato.

Queste dichiarazioni possono avvenire

- nella unit di configurazione, oppure
- nelle altre unit che compongono il progetto e che contengono anche il codice operativo del progetto.

I *simboli* sono i nomi assegnati alle variabili, array, datagroup, costanti, input o output. Per distinguere tra loro le varie dichiarazioni si utilizzano delle parole chiavi poste prima di ogni gruppo di dichiarazioni. Le parole chiavi sono:

- CONST
- SYSTEM
- GLOBAL
- ARRSYS
- ARRGBL
- TIMER
- INPUT
- OUTPUT
- INTDEVICE
- DATAGROUP (solo nella unit di configurazione)
- BUS (solo nella unit di configurazione)
- REFERENCES (solo nella unit di configurazione)

3.1.1 Unit di configurazione

La unit di configurazione è una componente di un progetto per QMOVE; c'è un'unica unit di configurazione in un progetto.
La unit di configurazione serve come contenitore di dichiarazioni di alcune variabili e di alcune le costanti utilizzate nell'applicativo; inoltre viene anche definita la composizione hardware del sistema QMOVE adottato, specificando il tipo di scheda CPU e il modello delle schede presenti.
Le variabili, array, datagroup, device e costanti dichiarati nella unit di configurazione sono disponibili per tutto il progetto. Possono essere visibili (e quindi leggibili e scrivibili) da tutte le altre unit che compongono il progetto.

3.1.2 Unit operative

Le altre unit che compongono un progetto per QMOVE possono essere più di una e hanno lo scopo di contenere il codice operativo che determina la logica di funzionamento del progetto stesso. Inoltre all'inizio di queste unit è possibile fare lo stesso tipo di dichiarazioni che si possono fare nella unit di configurazione (tranne la dichiarazione della composizione dell'hardware e la dichiarazione degli ingressi e uscite digitali).
Le variabili, array, datagroup dichiarate nelle unit operative possono essere dotate di una proprietà di accessibilità aggiuntiva. Normalmente quando si dichiara una variabile in una unit operativa essa può essere utilizzata solamente in quella unit. In fase di dichiarazione è possibile, però, assegnare ad essa una proprietà aggiuntiva:

- **IN**, il simbolo è accessibile in scrittura/lettura dalle altre unit, mentre è accessibile solamente in lettura dalla unit che dichiara quel simbolo;
- **OUT**, il quel simbolo è accessibile in scrittura/lettura dalla unit dichiarante, mentre le altre unit possono solamente usarlo in lettura
- **INOUT**, il simbolo è accessibile in scrittura/lettura sia dalla unit dichiarante che da tutte le altre unit.

3.2 Tipi di variabili

In seguito descriveremo tutte le informazioni che è necessario conoscere per poter dichiarare le variabili.

Nella descrizione della sintassi le parti tra <…> sono obbligatorie. Le parti tra […] sono opzionali, mentre le parti separate da "/" sono da considerarsi "usare uno di questi".

Il carattere punto e virgola (";") ha lo scopo di identificare una riga di commento. Tutto il testo inserito dopo questo carattere non verrà considerato. Il carattere punto e virgola vale solo per la linea in cui è inserito.

QCL prevede cinque tipi principali di dato:

3.2.1 Flag

Il dato di tipo FLAG è utilizzato per la definizione di variabili booleane che hanno un range di valori interi compreso tra 0 e 1. L'occupazione della memoria da parte delle variabili di questo tipo dipende dal loro numero.
La sintassi per la definizione di una variabile di tipo FLAG è la seguente:

```
<variable name>  F  [OUT/IN/INOUT] [;comment]
```

3.2.2 Byte

Il dato di tipo BYTE è utilizzato per la definizione di variabili che hanno un range di valori interi compreso tra -128 e +127; ogni variabile occupa un byte di memoria.
La sintassi per la definizione di una variabile BYTE è la seguente:

```
<variable name>  B  [OUT/IN/INOUT] [;comment]
```

3.2.3 Word

Il dato di tipo WORD è utilizzato per la definizione di variabili che hanno un range di valori interi compreso tra -32768 e +32767; ogni variabile occupa due byte di memoria.
La sintassi per la definizione di una variabile WORD è la seguente:

```
<variable name>  W  [OUT/IN/INOUT] [;comment]
```

3.2.4 Long

Il dato di tipo LONG è utilizzato per la definizione di variabili che hanno un range di valori interi compreso tra -2147483648 e +2147483647; ogni variabile occupa quattro byte di memoria. La sintassi per la definizione di una variabile LONG è la seguente:

```
<variable name>  L  [OUT/IN/INOUT] [;comment]
```

3.2.5 Single

Il dato di tipo SINGLE è un valore a virgola mobile come definito dalla specifica *IEEE 754 single precision*.
La sintassi per la definizione di una variabile SINGLE è la seguente:

```
<variable name>  S  [OUT/IN/INOUT] [;comment]
```

3.2.6 Double

Il dato di tipo DOUBLE è un valore a virgola mobile come definito dalla specifica *IEEE 754 double precision*.
La sintassi per la definizione di una variabile DOUBLE è la seguente:

```
<variable name>  D  [OUT/IN/INOUT] [;comment]
```

3.2.7 Tabella riassuntiva dei tipi di variabili utilizzabili

Assegnando ad una variabile un valore esterno al range consentito si verifica la condizione di overflow.

Tipo dato	Codice	Spazio occupato memoria (Bit)	Intervallo
FLAG	F	1	0 ÷ 1
BYTE	B	8	-128 ÷ 127
WORD	W	16	-32768 ÷ 32767
LONG	L	32	-2147483648 ÷ 2147483647
SINGLE	S	32	Vedi specifica *IEEE 754 single precision*
DOUBLE	D	64	Vedi specifica *IEEE 754 double precision*

3.3 Gli Identificatori

Gli identificatori sono dei nomi tramite i quali è possibile fare dei riferimenti ad oggetti o ad etichette. Ogni identificatore è formato da uno o più caratteri alfanumerici tenendo presente che il primo deve essere una lettera. I caratteri alfanumerici possono essere riassunti in:
- dalla “A” alla “Z”
- dalla “a” alla “z”
- da “0” a “9”
- “_”

3.3.1 I nomi

I nomi vengono usati per agevolare l'identificazione di un oggetto all'interno del sistema QMOVE. Per oggetto si intende qualsiasi entità gestibile a livello di linguaggio e provvista di caratteristiche fisiche proprie come ad esempio: variabili, ingressi, uscite e devices. In Qview 6.0 i nomi sono limitati ad una lunghezza massima di 64 caratteri.

3.4 Le costanti

Una costante è un carattere o una stringa di caratteri utilizzabile come valore in un progetto. Le costanti vengono dichiarate in seguito alla parola chiave *CONST*. Viene riportata la sintassi per la definizione di costanti:

```
CONST
<nome costante> <valore> [OUT] [;comment]
```

dove:

CONST	Parola chiave che indica la definizione delle costanti.
<nome costante>	Nome della costante.
<valore>	valore associato alla costante.
[OUT]	Si può usare in una dichiarazione all'interno di una unit operativa quando si vuole rendere accessibile in lettura la costante anche alle altre unit operative.
[;comment]	Il commento non è obbligatorio ma consigliato per una miglior lettura del software

Un esempio di dichiarazione di alcune costanti è:

```
CONST
  TM_SECOND  1000  OUT  ;These are the number of milliseconds in a second.
  DIM_ARRAY  7     OUT  ;Used as array dimension.
  DIM_PROG   10         ;Used as number of recipe.
  DIM_STEP   5          ;Used as number of step for each recipe.
```

3.5 Le variabili SYSTEM

Le variabili SYSTEM mantengono il loro valore anche allo spegnimento del sistema.

Con il nome SYSTEM vengono raggruppate tutte le variabili ritentive; possono essere di tipo FLAG, BYTE, WORD, LONG o SINGLE. Devono essere poste in seguito alla parola chiave *SYSTEM*. La sintassi per la definizione di variabili SYSTEM.

```
SYSTEM
  <nome variabile>    <F/B/W/L/S>  [OUT/IN/INOUT] [;comment]
```

dove:

SYSTEM	Parola chiave che indica la definizione di variabili SYSTEM.
<nome variabile>	Nome della variabile SYSTEM.
<F/B/W/L/S>	Tipo della variabile e può essere.
[OUT/IN/INOUT]	Tipo di accessibilità.
[;comment]	Il commento è consigliato per una miglior lettura del software.

Esempio:

```
SYSTEM
  count      L       ;This is used as a counter.
  sbSeconds  B  OUT  ;Seconds
  sbMinutes  B  OUT  ;Minutes
  swHours    W       ;Hours
```

3.6 Le variabili GLOBAL

Le variabili *GLOBAL* hanno la caratteristica di venire azzerate ad ogni accensione del sistema

Con il nome GLOBAL vengono raggruppate tutte le variabili non ritentive; possono essere di tipo FLAG, BYTE, WORD, LONG e

SINGLE. Vengono dichiarate in seguito alla parola chiave *GLOBAL*. Viene riportata la sintassi per la definizione di variabili GLOBAL.

```
GLOBAL
  <nome variabile>   <F/B/W/L/S>  [OUT/IN/INOUT] [;comment]
```

dove:

GLOBAL	Parola chiave che indica per la definizione di variabili GLOBAL.
<nome variabile>	Nome della variabile GLOBAL.
<F/B/W/L/S>	Tipo della variabile.
[OUT/IN/INOUT]	Tipo di accessibilità.
[;comment]	Il commento è consigliato per una miglior lettura del software.

Esempio:

```
GLOBAL
  gfMyFlag       F   OUT   ;Non retentive flag variable
  gbYourByte     B   IN    ;Non retentive byte variable
  gwHisWord      W   INOUT ;Non retentive word variable
  glYourLong     L   IN    ;Non retentive long variable
  gsTheirSingle  S   OUT   ;Non retentive single precision variable
```

3.7 Le variabili ARRAY SYSTEM

Una variabile Array System è un insieme di variabili ritentive dello stesso tipo. Esse hanno la stessa dimensione ed è possibile accedere ad esse tramite un nome comune e riferendosi ad uno specifico elemento tramite un indice. Non sono previsti array di variabili tipo FLAG.
Come le SYSTEM sono variabili ritentive e devono essere dichiarate di seguito alla parola chiave *ARRSYS*.
Viene riportata la sintassi per la definizione di variabili ARRAY SYSTEM.

```
ARRSYS
  <nome variabile>   <B/W/L/S>   <numero_elementi>  [OUT/IN/INOUT] [;comment]
```

dove:

GLOBAL	Parola chiave che per la definizione di variabili ARRAY SYSTEM.
<nome variabile>	Nome della variabile ARRAY SYSTEM.
<B/W/L/S>	Tipo della variabile e può essere
<numero_elementi>	Numero di elementi che compongono la variabile.
[OUT/IN/INOUT]	Tipo di accessibilità.
[;comment]	Il commento è consigliato per una miglior lettura del software.

Il numero massimo di elementi consentiti in un array è di 65535. Non si possono indicare dimensioni negative o dimensioni con parte decimale. Come dimensione dell'array è possibile usare delle costanti già definite nella sezione CONST; se questa avesse valore decimale verrà troncato: ad esempio il valore 200.34 viene forzato a 200.
Esempio:

```
ARRSYS
  asbMyArray    B 10         INOUT  ;Array system declaration of 10 bytes in size.
  aslYourArray  L DIM_ARRAY  IN     ;Array system declaration of DIM_ARRAY bytes in size.
```

Si osservi che nel secondo array si è usata una costante per la dichiarazione della dimensione.

3.8 Le variabili ARRAY GLOBAL

Una variabile Array Global è un insieme di variabili non ritentive dello stesso tipo. Esse hanno la stessa dimensione ed è possibile accedere ad esse tramite un nome comune, riferendosi ad uno specifico elemento tramite un indice. Non sono previsti array di variabili tipo FLAG.
Come le GLOBAL sono variabili non ritentive e devono essere dichiarate di seguito alla parola chiave "ARRGBL".
Viene riportata la sintassi per la definizione di variabili ARRAY GLOBAL:

```
ARRGBL
  <nome variabile>   <B/W/L/S>   <numero_elementi>  [OUT/IN/INOUT] [;comment]
```

dove:

ARRGBL	Parola chiave per la definizione di variabili ARRAY GLOBAL.
<nome variabile>	Nome della variabile ARRAY GLOBAL.
<B/W/L/S>	Tipo della variabile e può essere.
<numero_elementi>	Numero di elementi che compongono la variabile.
[OUT/IN/INOUT]	Tipo di accessibilità.

[;comment]	Il commento è consigliato per una miglior lettura del software.

Esempio:

```
ARRGBL
arwMyArray  W  15 OUT  ;Array global declaration of byte of 15 dimension.
```

3.9 Le variabili TIMER

Sono variabili utilizzate per realizzare temporizzazioni alle quali può essere assegnato un valore intero (espresso in ms) che rappresenta il tempo che deve trascorrere (dal momento dell'assegnazione); in lettura è disponibile lo stato di "temporizzazione terminata" (1) o "temporizzazione in corso" (0).
Le variabili timer vengono dichiarate in seguito alla parola chiave "TIMER".
La sintassi per la definizione di variabili TIMER è:

```
TIMER
  <timer name> [OUT/IN/INOUT]  [;comments]
```

dove:

TIMER	Parola chiave per la definizione di variabili TIMER.
<timer name>	Nome della variabile TIMER.
[OUT/IN/INOUT]	Tipo di accessibilità.
[;comment]	Il commento è consigliato per una miglior lettura del software.

Esempio:

```
TIMER
  tTimer1   INOUT  ;First timer.
  tTimer2   OUT    ;Second timer.
  tSeconds  OUT    ;3rd timer.
  tDelay    INOUT  ;4th timer.
```

Quando la variabile timer si trova a sinistra dell'assegnazione si definisce il caricamento di un valore sul timer (valore espresso in millisecondi):

```
tMyTimer = 1000  ;Set timer tMyTimer to 1 second.
```

Quando la variabile timer si trova a destra dell'assegnazione o all'interno di una espressione ne viene letto lo stato (0 = Temporizzazione in corso, 1 = Temporizzazione terminata):

```
gfIsTimerEnd = tMyTimer ;Assign to gfIsTimerEnd variable the timer state.
```

oppure

```
IF(tMyTimer)  ;If timer tMyTimer has passed exec the code
  ;Put here the code
  .....
ENDIF
```

È, inoltre, possibile leggere il valore del tempo rimanente prima dello scadere del timer (il valore restituito è espresso in millisecondi):

```
< Timer Name >.remain
```

Esempio:

```
glRemainTime = tMyTimer.remain
```

3.10 Le variabili INPUT ed OUTPUT

Sono tutte le variabili che fanno riferimento ad ingressi od uscite digitali. La loro dichiarazione deve essere posta in seguito alle parole chiave *INPUT* per gli ingressi o *OUTPUT* per le uscite.

Gli ingressi e uscite digitali possono essere dichiarati solamente nella unit di configurazione.

La sintassi per la definizione di variabili INPUT ed OUTPUT è:

```
INPUT
  <variable name>   <F/B/W>   <io address>
OUTPUT
  <variable name>   <F/B/W>   <io address>
```

Gli I/O address sono disponibili sulle schede tecniche hardware delle schede utilizzate.

dove:

INPUT	Parola chiave per la definizione di variabili INPUT.
OUTPUT	Parola chiave per la definizione di variabili OUTPUT.
<variable name>	Nome della variabile, cioè il simbolo associato all'ingresso o uscita.
<F/B/W>	Tipo della variabile
<io address>	Indirizzo dell'INPUT o OUTPUT così composto: *Slot number.name: Slot number* è il numero dello slot nel quale è posta la scheda con la risorsa hardware. *name*: è il nome che fa riferimento all'indirizzo fisico dell'I/O (definito nei riferimenti hardware).

Un'applicazione interessante degli ingressi e delle uscite digitali è quello del raggruppamento degli stessi in un unico identificatore. Questo identificatore è analogo ad una variabile di otto o sedici bit dove ogni ingresso o uscita digitale rappresenta un bit.
Esempio:

```
INPUT
  ibInput  B  3.INPB     ;8 digital input grupped in a single byte
```

```
OUTPUT
  obOutput  B  3.OUTB      ;8 digital output grupped in a single byte
```

Se si dispone di schede con più di otto ingressi o uscite digitali Esempio:

```
INPUT
  ibInput1  B  3.INPB1   ;First 8 digital input grupped in a byte (1÷8).
  ibInput2  B  3.INPB2   ;Second 8 digital input grupped in a byte (9÷16).
  iwInput   W  3.INPW    ;A word of 16 digital input grupped in a word (1÷16).
```

```
OUTPUT
  obOutput1  B  3.OUTB1   ;First 8 digital output grupped in a byte 8 (1÷8).
  obOutput2  B  3.OUTB2   ;Second 8 digital output grupped in a byte 8 (9÷16).
  owOutput   B  3.OUTW    ;A word of 16 digital output grupped in a word (1÷16).
```

3.11 Le variabili DATAGROUP

Le variabili Datagroup sono una particolare struttura di dati, memorizzate in una zona ritentiva. Esse vengono dichiarate nella unit di configurazione e sono idonee per rappresentare un archivio di ricette.
Le variabili Datagroup contengono due tipologie di variabili:

3.11.1 Variabili Statiche

```
DATAGROUP
  <DataGroup Name>
  DATAPROGRAM
    <number of recipes>
;Static variables declaration
    <variable name> <F/B/W/L/S>
    <variable name> <F/B/W/L/S>
    <variable name> <F/B/W/L/S>
```

Le variabili statiche risiedono nel sottosettore di DATAGROUP nominato come DATAPROGRAM. Il primo valore di questo settore è un numero o una costante intera, e indica il numero di ricette dell'archivio. Le variabili statiche sono da considerarsi come contenitori di un determinato valore per ogni ricetta, e sono accessibili come un array, dove l'indice è il numero di ricetta desiderata. Esempio:

```
DATAGROUP
  Name
  DATAPROGRAM
    100
;Static variables declaration
    Variable1     L
    Variable2     S
    Variable3     F
```

Per far riferimento alla variabile *Variable3* della ricetta 5, viene usato il codice:

```
Variable3[5]
```

Possiamo immaginare la struttura della memoria di una variabile DATAGROUP, nel nostro esempio con variabili statiche, come la seguente tabella:

	Variable1	**Variable2**	**Variable3**
Ricetta 1			
Ricetta 2			
Ricetta 3			
...			
Ricetta 100			

3.11.2 Variabili Indicizzate

```
DATAGROUP
  <DataGroup Name>
  DATAPROGRAM
```

```
    <number of recipes>
  STEP
    <number of steps>
;Indexed variables declaration
    <variable name> <F/B/W/L/S>
    <variable name> <F/B/W/L/S>
    <variable name> <F/B/W/L/S>
```

Le variabili indicizzate risiedono nel sottosettore di DATAGROUP nominato come STEP. La loro funzione è quella di introdurre per ogni ricetta, il concetto di passo. Il primo valore di questo settore è un numero o una costante intera, e indica il numero di passi per ogni ricetta dell'archivio. Le variabili indicizzate sono da considerarsi come contenitori di una lista di valori per ogni ricetta, e sono accessibili come un array a 2 dimensioni, dove il primo valore è il numero di ricetta, il secondo il passo desiserato. Esempio:

```
DATAGROUP
  Name
  DATAPROGRAM
    100
;Static variables declaration
    Variable1     L
    Variable2     S
    Variable3     F
  STEP
    10
;Indexed variables declaration
    Variable4     W
    Variable5     B
```

Per far riferimento alla variabile *Variable4* della ricetta 5 al passo 9, viene usato il codice:

```
Variable4[5, 9]
```

Possiamo immaginare la struttura della memoria di una variabile DATAGROUP, nel nostro esempio con variabili statiche e indicizzate, come la seguente tabella:

	Variable1	Variable2	Variable3	Variable4	Variable5
Ricetta 1					
Ricetta 2					
Ricetta 3					
...					
Ricetta 100					

Altre informazioni:

- In un DATAGROUP tutte le variabili, sia statiche che indicizzate, sono ritentive (mantengono il valore allo spegnimento).
- Le variabili Datagroup possono essere più di una, in questo caso è necessario inserire più parole chiave DATAGROUP.
- La sottosezione DATAPROGRAM è obbligatoria, mentre quella STEP è opzionale.
- Il numero massimo di ricette impostabile è 65534.
- Il numero massimo di passi impostabile è 65534.
- Rispetto ad un comune array, le variabili statiche possono supportare anche un dato di dimensione Flag (F).

3.12 Sezione BUS

La sezione BUS nell'unità di configurazione è indispensabile per dichiarare quale modello hardware di QMOVE il programmatore ha a disposizione.
In base al modello hardware dichiarato è possibile anche avere a disposizione le sue risorse hardware come per esempio il numero di ingressi e uscite digitali, il numero di ingressi e di uscite analogiche oppure il numero di ingressi di conteggio. Il QMOVE può inoltre essere dotato anche di altre risorse come interfacce seriali o lettori di memoria removibile.
La dichiarazione da inserire nella sezione *BUS* si suddivide in *slot*. Nello *slot 1* si indica quale è la CPU utilizzata mentre negli *slot* successivi servono per dichiarare quali schede elettroniche sono installate nel modello. Ogni scheda è individuata da una parola chiave che ne identifica tipologia di hardware. Le parole chiavi sono reperibili nei manuali installazione e manutenzione dell'hardware utilizzato.

La sintassi da utilizzare è:

```
BUS
   1  <ID_CPU>   <FW>
  [2  <ID_CARD>   .]
  [3  <ID_CARD>   .]
  [4  <ID_CARD>   .]
```

Esempio:

```
BUS
  1 1P51F   30      ;QMOVE J1-P51F with firmware 30
  2 .        .      ;No card is installed in this slot
```

```
3 1MG8F    .      ;This card has 32 digital input, 32 digital output, 6 analog output, ecc.
```

4. Visibilità delle variabili

L'argomento relativo alla visibilità dei simboli dichiarati all'interno del progetto è molto importante. Vale la pena di riprenderlo con cura in questo capitolo dedicato ad esso.
Come abbiamo detto in precedenza, è possibile dichiarare i seguenti gruppi

```
CONST
SYSTEM
GLOBAL
ARRSYS
ARRGBL
TIMER
INPUT
OUTPUT
INTDEVICE
```

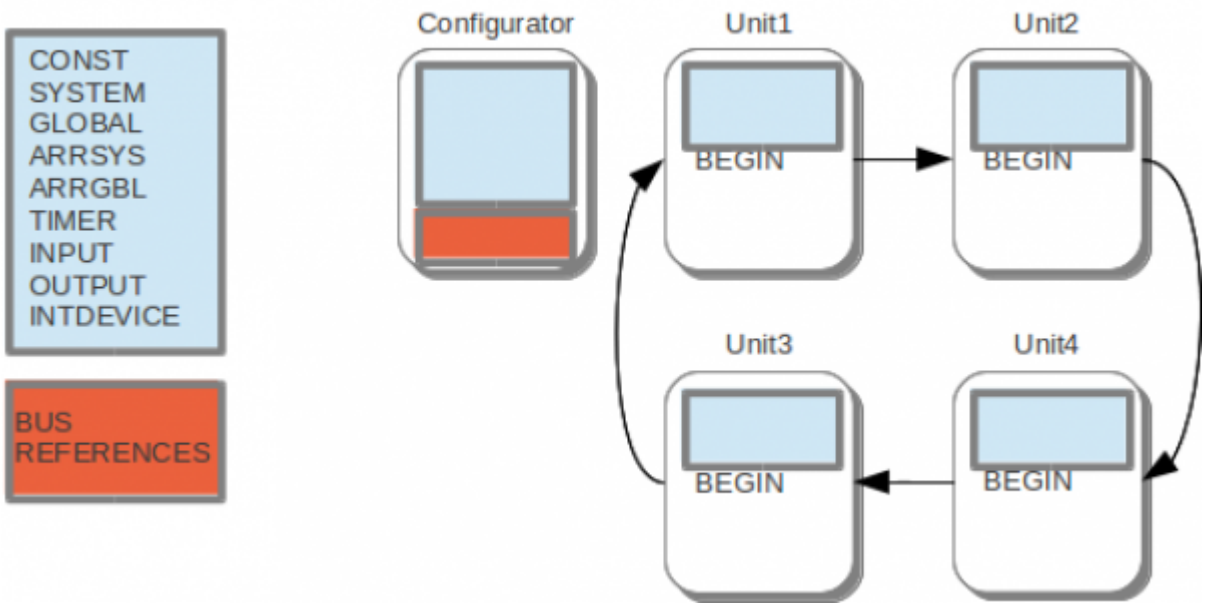
sia nella unit di configurazione che nelle altre unit che compongono il progetto. Gli altri gruppi di dichiarazioni

```
BUS
REFERENCES
```

Possono essere presenti solamente nella unit di configurazione.

Come vedete è apparsa la parola chiave *REFERENCES* che merita una spiegazione approfondita alla fine di questo capitolo.

Nella seguente figura è rappresentato schematicamente dove è possibile trovare i vari gruppi di dichiarazione in un progetto.



4.1 Variabili di configurazione

I simboli dichiarati nella unit di configurazione hanno una visibilità estesa a tutte le unit del progetto. Facciamo l'esempio di dichiarare una variabile del gruppo SYSTEM:

```
SYSTEM
  ConfigurationVariable  L  ;This variable is declared in the configuration unit
```

Questa variabile potrà essere utilizzata in qualsiasi altra unit presente nel progetto sia per assegnare ad essa un valore (scrittura) sia per essere usata come valore all'interno di una espressione (lettura).

```
ConfigurationVariable = 5                    ;Assign the value 5 to the variable
Unit1Variable = ConfigurationVariable * 2    ;Assign twice variable value to another variable
```

In realtà la notazione completa per riferirsi alla variabile *ConfigurationVariable* è la seguente:

```
APPLICATION.ConfigurationVariable
```

La radice *APPLICATION.* può essere omessa e spiegheremo in seguito quando usarla.

4.2 Variabili locali

I simboli che vengono dichiarati all'interno delle unit che contengono il codice operativo, prima della parola chiave *BEGIN*, hanno una visibilità diversa. Per esempio la seguente variabile:

```
GLOBAL
  local_variable  W  ;This variable is declared in a unit
BEGIN
  ...  ;Here are the code lines
```

è visibile solamente all'interno della unit in cui viene dichiarata. Nessuna altra unit potrà accedere a questa variabile né in lettura né in scrittura. Essa è considerata una variabile *locale* nel senso che viene dichiarata per scopi che si esauriscono nella stessa unit. È possibile che un'altra unit contenga una dichiarazione di una variabile con lo stesso nome. Non sarà possibile infatti incorrere in equivoci perché quando si usa il simbolo *local_variable* in una unit, è chiaro che ci si riferisce sempre alla variabile *locale* e non a quella di un'altra unit.

4.3 Variabili IN/OUT/INOUT

Aggiungendo dei particolari attributi alla dichiarazione di una variabile, è possibile creare un'interfaccia verso l'esterno della unit in cui è dichiarata. Questi attributi sono

- IN
- OUT
- INOUT

Se per esempio ampliamo il precedente esempio:

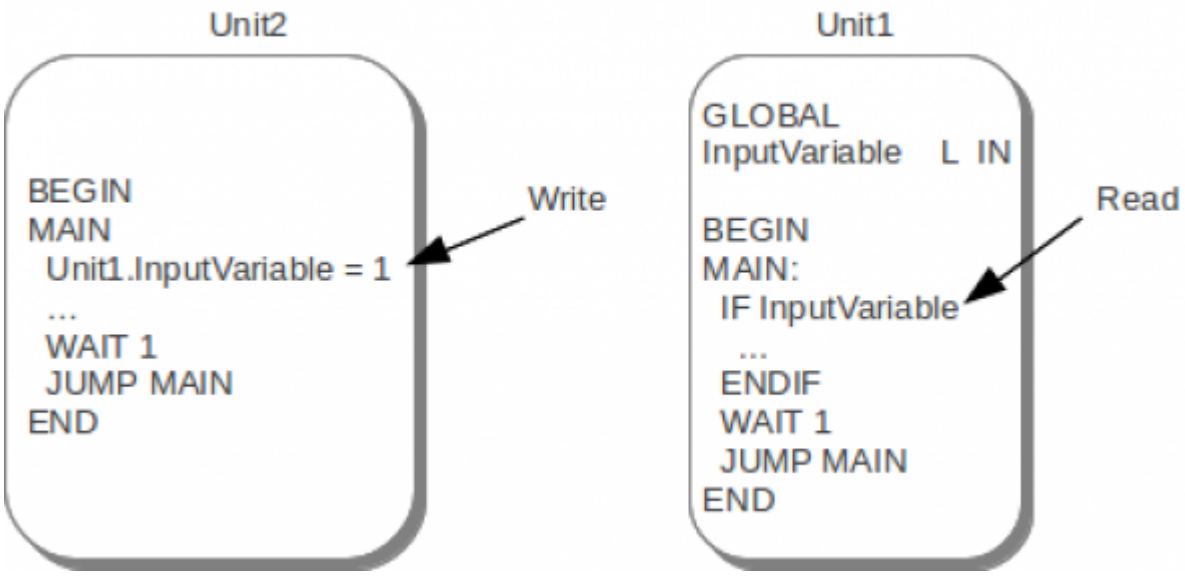
```
GLOBAL
  InputVariable    L  IN     ;This variable is like an input
  OutputVariable   L  OUT    ;This variable is like an output
  InOutVariable    L  INOUT  ;This variable is both an input and an output

  local_variable   W         ;This variable is declared in a unit
BEGIN
  ...  ;Here are the lines code
```

otteniamo una interfaccia della unit verso le altre unit.

4.3.1 IN

La variabile *InputVariable* viene vista dalla unit come un input e quindi essa non potrà essere modificata internamente ma solamente utilizzata per il suo valore. In pratica è una variabile in sola lettura per la unit che la dichiara.



Come si vede dalla figura, le altre unit (Unit2) possono accedere ad essa se specificano quale è la unit a cui la variabile appartiene:

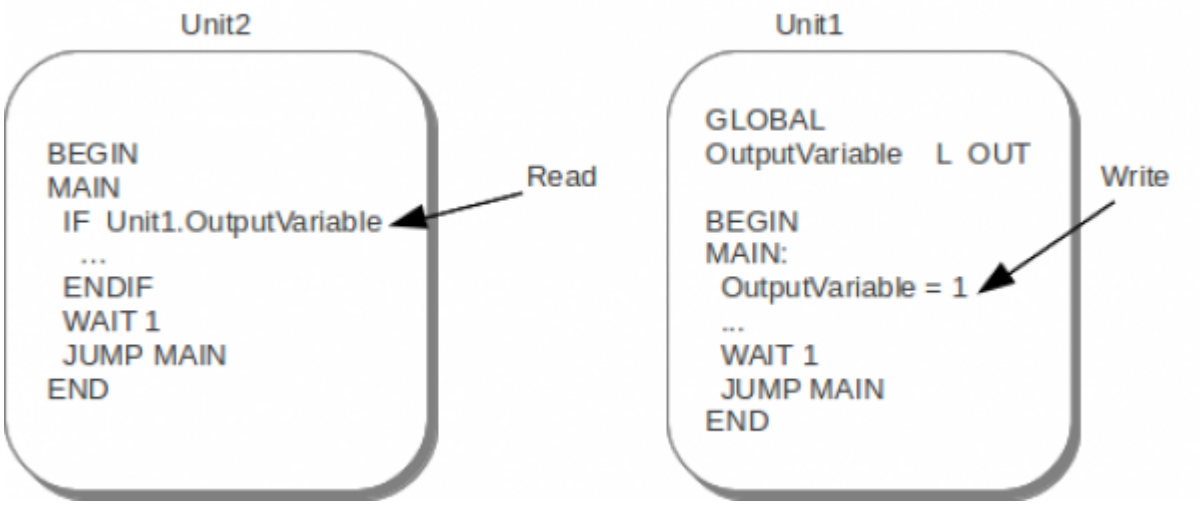
```
  Unit1.InputVariable = 1
```

In questo modo ci potranno essere unit che hanno la stessa o le stesse variabili con proprietà IN. Non potranno esserci equivoci perché ogni variabile appartiene a unit con nomi differenti.

4.3.2 OUT

La variabile *OutputVariable* è un output per la unit che la dichiara. Essa quindi è il risultato di una elaborazione interna alla unit ed è usata per fornire alle altre unit questo risultato. La variabile dichiarata con questa proprietà è in sola lettura per tutte le

altre unit.

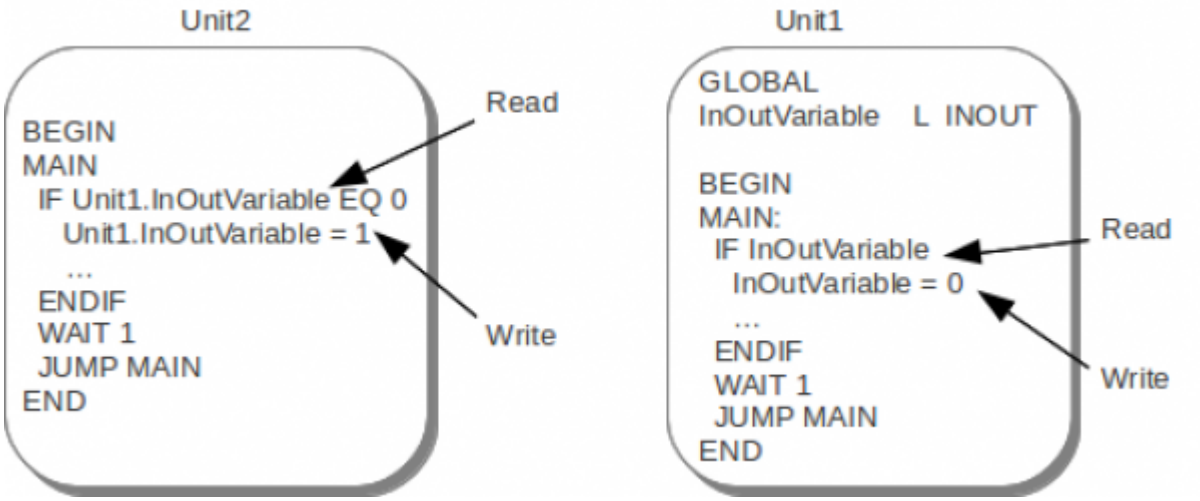


Come si vede dalla figura, le altre unit (Unit2) possono utilizzare il valore della variabile *OutputVariable* solo se specificano prima il nome della unit a cui la variabile appartiene:

```
IF Unit1.OutputVariable
  ...
ENDIF
```

4.3.3 INOUT

La variabile *InOutVariable* è sia un IN che un OUT. Tale variabile può essere modificata sia dalla unit che la dichiara sia dalle altre unit.



Anche in questo caso le altre unit (Unit2) possono utilizzare la variabile della Unit1 specificando prima del suo nome anche il nome della unit a cui appartiene:

```
IF Unit1.OutputVariable EQ 0
  Unit1.OutputVariable = 1
  ...
ENDIF
```

4.4 APPLICATION

Abbiamo già detto che la notazione completa per identificare le variabili dichiarate nella unit di configurazione è

```
APPLICATION.<variable name>
```

E' necessario usare la radice *APPLICATION.* prima del nome della variabile quando ci possono essere fonti di equivoco. Infatti se venisse dichiarata la variabile *HomePosition* nella unit di configurazione e un'altra variabile con lo stesso nome in Unit1, ciò sarebbe fonte di equivoco.
Per questo motivo quindi si deve usare la notazione

```
APPLICATION.HomePosition
```

e

```
Unit1.HomePosition
```

a seconda di chi ci si riferisce.

5. REFERENCE

Un diverso attributo che può essere assegnato ad un simbolo al momento della dichiarazione è il REFERENCE. In questo caso il simbolo che viene dichiarato come REFERENCE è semplicemente un riferimento ad una variabile dichiarata in un altro punto (tipicamente nella unit di configurazione). La dichiarazione di questi speciali simboli sarà composta di due fasi

1. dichiarazione nella unit dove viene utilizzata;
2. dichiarazione nella unit di configurazione (nella sezione REFERENCES) di quale variabile è il riferimento.

Per esempio:

Nella Unit1

```
GLOBAL
  MachineState  B  REFERENCE
```

Nella unit di configurazione

```
GLOBAL
GeneralState  B
...
REFERENCES
Unit1.MachineState  = GeneralState
```

La variabile *GeneralState* è un simbolo generico che viene associato alla variabile dichiarata localmente in Unit1 e chiamata *MachineState*. I riferimenti devono essere sempre stabiliti tra simboli dello stesso tipo. E' possibile creare dei riferimenti per tutti i tipi di dichiarazioni (SYSTEM, GLOBAL, ARRSYS, ARRGBL, INPUT, OUTPUT, INTDEVICE, TIMER).

La dichiarazione di **INTDEVICE**, **INPUT**, **OUTPUT** può avvenire solamente come **REFERENCE**.

6. COSTANTI

Anche le costanti possono essere dichiarate nelle unit che contengono il codice operativo. Se la dichiarazione avviene in questo modo nella Unit1 per esempio:

```
CONST
  MAX_PROGRAM_NUMBER  100  ;local constant
  ...
BEGIN
  ...
```

la costante *MAX_PROGRAM_NUMBER* potrà solamente essere usata internamente alla unit che la dichiara. Se invece la dichiarazione è come la seguente:

```
CONST
  MAX_PROGRAM_NUMBER  100  OUT  ;Output constant
  ...
BEGIN
  ...
```

allora la costante *MAX_PROGRAM_NUMBER* sarà visibile anche dalle altre unit. Vale sempre e comunque la regola che le altre unit potranno accedere a questa costante in questo modo:

```
  Unit1.MAX_PROGRAM_NUMBER
```

7. Istruzioni QCL

Il QCL (QEM Control Language) è un linguaggio nato appositamente per la programmazione del sistema QMOVE. Le caratteristiche principali del QCL sono la semplicità (poche ma potenti istruzioni), la facilità d'uso (somiglianza con il linguaggio BASIC) e l'orientamento verso l'automazione industriale grazie alle istruzioni appositamente studiate per il controllo assi.

7.1 Gli operatori del QCL

Nel linguaggio QCL vengono messi a disposizione tutti gli operatori elementari per la manipolazione dei dati.

7.1.1 Operatore di assegnamento

L'operatore di assegnamento può essere utilizzato all'interno di una qualunque espressione e può agire su tutte le variabili, sia elementari che appartenenti a DataGroup o a parametri Device. La forma generica di un'istruzione di assegnamento è la seguente:

```
<variable name> = <expression>
```

Se la dimensione della variabile *variable name* è inferiore alla dimensione del risultato dell'espressione il valore viene troncato, secondo le regole della conversione di tipo (vedi appendice dedicata).

7.1.2 Operatori aritmetici

Gli operatori aritmetici sono

- somma (+),
- sottrazione (-),
- moltiplicazione (*),
- divisione (/),
- resto della divisione (%).

Questi operatori agiscono su tutti i tipi di dato e sulle loro combinazioni (compresi i bit).
Esempio:

```
  sbVar01 = slVar02 + swVar03
  slVarxy = glVarA * glVarB
  sl01 = ss02 % ss03
```

7.1.3 Operatori logici

Gli operatori logici (AND, OR e NOT) possono agire su tutte le variabili tranne quelle a singola precisione. La loro funzione è riassunta dalla seguente tabella.

Negli esempi il valore 1 indica semplicemente che la variabile è diversa da 0.			
a b	a AND b	a OR b	NOT a
0 0	0	0	1
0 1	0	1	1
1 0	0	1	0
1 1	1	1	0

Esempi

```
  glBit01 = glBit02 AND gfBit03
  glBitA = NOT glBitB
```

7.1.4 Operatori binari

Gli operatori binari

- ANDB,
- ORB,
- NOTB,
- XORB

agiscono su tutti i tipi di variabili compresi i SINGLE (previa conversione automatica in intero). Questi operatori realizzano la stessa tabella di verità che regola le equivalenti operazioni logiche, con la differenza che agiscono sui singoli bit.

$(a)_{10}$	$(b)_{10}$	$(a)_2$	$(b)_2$	a ANDB b	a ORB b	NOTB a	a XORB b
3	2	011	010	010	011	100	001
7	5	111	101	101	111	000	010

Esempio

```
swVar01 = gwVar02 ANDB gwVar03
```

7.1.5 Operatori relazionali

Permettono di valutare le relazioni che intercorrono tra i valori. Il linguaggio QCL definisce cinque operatori relazionali.

EQ	uguale a (equal)
NEQ	non uguale a (not equal)
LT	minore di (lower than)
LE	minore o uguale a (lower equal)
GT	maggiore di (greater than)
GE	maggiore o uguale a (greater equal)

Esempio

```
gfBit01 = glCont GE glAsse01
```

7.1.6 Gli operatori di livello di precedenza

Sono le parentesi rotonde “()” che permettono di modificare il livello di precedenza degli operatori sopradescritti.
Esempio AA = [(A + B)/C] - (D x E)

```
gwVarAA = ((gwVarA + gwVarB) / glVarC) - (swVarD * glVarE)
```

7.1.7 Livelli di priorità degli operatori del QCL

Il livello di priorità degli operatori QCL ricalca quello di tutti i linguaggi di programmazione. Le priorità sono le seguenti:

Priorità massima
( )
Funzioni matematiche e Funzioni trigonometriche
NEG NOT NOTB
* / %
+ -
LT GT LE GE
EQ
AND ANDB
OR ORB XORB
=
Priorità minima

7.2 Istruzioni e strutture per il controllo del flusso

Sono tutte quelle istruzioni che permettono di modificare il flusso di esecuzione.

7.2.1 IF / ELSE / ENDIF

È la tipica istruzione condizionale; se la <expression> è vera, valore booleano 1, viene eseguito il blocco istruzioni

1

```
 altrimenti viene eseguito il blocco istruzioni <code lines 2>. Un blocco istruzioni può essere costituito da un
insieme di istruzioni ma anche da un'istruzione sola; inoltre la sezione ELSE è
opzionale. La sintassi è:
<code QCL>
  IF <expression>
    <code lines 1>
  [ELSE
```

```
    <code lines 2>]
  ENDIF
```

Esempio:

```
MAIN:
  IF tTime                    ;If timer has expired
    tTime = TM_SECOND         ;reload the timer and
    count = count + 1         ;increase by 1 the variable count
  ENDIF
  WAIT 1
  JUMP MAIN
END
```

Esempio di ciclo IF nidificato.
Modifichiamo l'esempio appena introdotto nel modo seguente:

```
MAIN:
  IF tSeconds                     ;If the timer has expired
    tSeconds = TM_SECONDO         ;reload the timer.
    IF sbSeconds LT 59            ;If the sbSeconds variable is less than 59
      sbSeconds = sbSeconds + 1   ;increase the seconds counter
    ELSE                          ;else (if the seconds are greater than 59)
      sbSecondi = 0               ;reset the seconds
      IF sbMinutes LT 59          ;If the sbMinutes variable is less than 59
        sbMinutes = sbMinutes + 1 ;increase the minutes counter
      ELSE                        ;else (if the minutes are greater than 59)
        sbMinuti = 0              ;reset the minutes
        IF swHours LT 167         ;If the swHours variable is less than 167
          swHours = swHours + 1   ;increase the hours
        ELSE                      ;else (if the hours is greater than 167)
          swHours = 0             ;reset the hours, because a week has passed.
        ENDIF
      ENDIF
    ENDIF
  ENDIF
  WAIT 1
  JUMP MAIN
END
```

Abbiamo realizzato così un orologio di sistema con tre variabili system: una per i secondi, una per i minuti e una per le ore. Si è utilizzata una variabile di tipo Byte per i secondi e per i minuti dato che il valore che esse assumono non esce mai dal range [-128, 127], mentre per la variabile swHours si è utilizzato una variabile di tipo Word dato che può arrivare fino al valore 167, all'interno del range [-32768, 32767], che è il numero di ore in una settimana.

7.2.2 FOR-NEXT

È un ciclo iterativo che serve normalmente per eseguire un certo numero di volte una o un blocco di istruzioni. È possibile uscire dal ciclo FOR-NEXT prima che diventi falsa la condizione di ciclo tramite l'istruzione BREAK; quando quest'ultima viene incontrata viene eseguito un salto alla prima istruzione successiva al NEXT. La sintassi è:

```
  FOR (<initialization>, <test>, <increment>)
    < code lines >
  NEXT
```

dove:

<initialization>	è una espressione eseguita all'avvio del ciclo e quindi una sola volta
<test>	è una espressione condizionale che, se falsa, determina la fine del ciclo FOR
<increment>	è un valore numerico che viene sommato all'indice al termine di ogni esecuzione del < code lines >

Esempio:

```
MAIN:
  FOR(gbYourByte = 1, gbYourByte LE 10, 1)      ;For gbYourByte from 1 to 10
    arwMyArray[gbYourByte] = POW(3, gbYourByte) ;the element arwMyArray[gbYourByte] gets the value 3^gbYourByte,
    dswStat2[gbYourByte] = 1                    ;assign the value 1 to the static variable,
    ddbDin1[gbYourByte,1] = 2                   ;assign the value 2 to the indexed variable if the 1st step.
  NEXT
  WAIT 1
  JUMP MAIN
END
```

Con il ciclo FOR-NEXT visto si riempie l'array global *arwMyArray* delle prime 10 potenze di 3.

7.2.3 WHILE-ENDWHILE

È un ciclo iterativo che serve per eseguire una o un blocco di istruzioni finché una condizione si verifica.
È possibile uscire dal ciclo WHILE-ENDWHILE prima che diventi falsa la condizione di ciclo tramite l'istruzione BREAK; quando quest'ultima viene incontrata viene eseguito un salto alla prima istruzione successiva al ENDWHILE. La sintassi è:

```
  WHILE (<condition>)
    < code lines >
  ENDWHILE
```

dove:

<condition>	è una espressione condizionale che deve dare un risultato booleano. Quando la condizione è vera vengono eseguite continuamente le < code lines >. Quando la condizione è falsa si passa ad eseguire il codice successivo a ENDWHILE.

Esempio:

Questi due cicli hanno lo scopo di poter impostare il valore delle variabili sbMinutes e swHours tenendo attivati due diversi ingressi ifSetUpMinutes e ifSetUpHours, rispettivamente. Si osservi che all'interno di ogni WHILE è stato inserito un timer di ritardo, tDelay, per fare in modo che l'incremento della variabile sia più lento per dare modo all'utente di disattivare l'ingresso quando la variabile assume il valore voluto. Naturalmente il valore da assegnare ai minuti o alle ore è limitato e quindi viene aggiunto un IF all'interno del while per controllare questo limite. La spiegazione dell'istruzione *WAIT 1* viene rimandata in seguito.

```
MAIN:

  WHILE (ifSetUpHours)           ;While the input is actived:
    IF tDelay                    ;if the timer is expired
      tDelay = 500               ;reload the timer,
      swHours = swHours + 1      ;increase the hours counter.
      IF swHours GE 168          ;If the hours are more than 167 (great equal than 168)
        swHours = 0              ;reset the hours counter.
      ENDIF
    ENDIF
    WAIT 1
  ENDWHILE

  WHILE (ifSetUpMinutes)         ;While the input is actived:
    IF tDelay                    ;if the timer is expired
      tDelay = 500               ;reload the timer,
      sbMinutes = sbMinutes + 1  ;increase the minutes counter.
      IF sbMinutes GE 60         ;If the minutes are more than 59 (great equal than 60)
        sbMinutes = 0            ;reset the minutes counter.
      ENDIF
    ENDIF
    WAIT 1
  ENDWHILE
WAIT 1
  JUMP MAIN
END
```

7.2.4 SWITCH-ENDSWITCH

Nel QView 6 è stata introdotta la istruzione SWITCH. Questa istruzione permette di analizzare una certa espressione e eseguire delle righe di codice diverse a seconda del risultato di questa espressione. La sintassi è:

```
  SWITCH(<expression>)
    CASE <value 1>
      < code lines for value 1>                                          [;comments]
    CASE <value 2>..<value 3>
      < code lines for values between value 2 and value 3>              [;comments]
    CASE <value 4>, <value 5>..<value 6>
      < code lines for value 4 and for values between value 5 and value 6>  [;comments]
    [DEFAULT
      < code lines for default>]                                         [;comments]

  ENDSWITCH
```

Un esempio banale di un controllo della temperatura rilevata da un sensore che scrive il suo valore nella variabile *Temperature*.

```
  SWITCH(Temperature)
    CASE 0
      CloseDoor = 1
      WarmUp = 100
      IceAlarm = 1
      OutOfRange = 0
    CASE 1..10
      CloseDoor = 1
      WarmUp = 50
      IceAlarm = 0
      OutOfRange = 0
    CASE 11..15
      CloseDoor = 1
      WarmUp = 15
      IceAlarm = 0
      OutOfRange = 0
    CASE 16..25
      CloseDoor = 0
      WarmUp = 0
      IceAlarm = 0
      OutOfRange = 0
    DEFAULT
      OutOfRange = 1
  ENDSWITCH
```

7.2.5 WAIT

Attende il verificarsi di una condizione o di un evento. L'istruzione WAIT è una istruzione molto importante poichè quando viene eseguita e la <condizione> è falsa, l'esecuzione delle istruzioni passa al task successivo. È comunque necessario fare una precisazione in questo contesto; infatti l'esecuzione dell'istruzione WAIT comporta comunque la cessione del controllo al task successivo quando viene incontrata la prima volta, senza che la condizione sia controllata. Quando il controllo del flusso istruzioni ritorna al task in questione viene rieseguita l'istruzione WAIT e quindi controllata la condizione. Per maggiori informazioni fare riferimento al capitolo dedicato al multitasking. La sintassi è:

```
WAIT <condizione>
```

7.2.6 JUMP

Esegue un salto incondizionato all'istruzione locata all'etichetta specificata. La sintassi è:

```
JUMP <etichetta>
```

Esempio
L'esempio classico per l'istruzione JUMP è quello che viene ripetuto in ogni task per saltare dalla fine del codice all'inizio, cioè all'etichetta MAIN. Questa istruzione per il controllo di flusso può essere utilizzata anche in altri punti del task. JUMP è un salto incondizionato, si può renderlo condizionato in combinazione con un IF con lo scopo di deviare il flusso di codice in base al verificarsi di particolari condizioni.

7.2.7 ETICHETTE

Tramite le etichette è possibile localizzare determinati punti nella sequenza delle istruzioni dell'applicativo. Vengono utilizzate per modificare il flusso di esecuzione delle istruzioni (comando JUMP del QCL). Quando viene inserita un'etichetta nelle istruzioni è necessario che sia seguita dai due punti ":", mentre se si fa riferimento all'etichetta stessa si deve scrivere il nome senza i due punti.
Esempio jump incondizionato

```
Label1:                    ;labeled line
  < code line >
  < code line >
  ;.....................
JUMP Label1                ;Jump to the label instruction
```

Esempio jump condizionato

```
Label1:                    ;labeled line
  < code line >
  IF <condition>           ;If the condition is true
    JUMP Label1            ;Jump to the label instruction
  ENDIF
```

7.2.8 SUBROUTINE

Quando, all'interno di una stessa unit, vi è una parte di codice ripetuta più volte è utile definire una subroutine. L'esecuzione del codice in essa contenuto avverrà ogni qualvolta viene incontrata una istruzione di chiamata a subroutine (CALL). La subroutine viene identificata da un nome preceduto dalla parola chiave SUB e deve terminare con la parola chiave ENDSUB.
Vedi anche istruzione CALL.
Per interrompere una subroutine prima del termine della stessa inserire la parola chiave RETURN.

7.2.9 CALL

Call to Subroutine: esegue un salto incondizionato alla prima istruzione della subroutine identificata dal nome. Al termine della subroutine (identificata da ENDSUB) il flusso del programma continua dalla istruzione successiva alla CALL.

```
CALL <nome_subroutine>
```

Queste righe di codice vanno poste oltre la parola chiave END, cioè oltre la fine della unit operativa.

Introduciamo una subroutine per calcolare la lunghezza di una diagonale di un quadrato data la lunghezza di un lato.

L'esempio a seguire illustra il richiamo della subroutine:

```
MAIN:

  glOurLong = 4              ;This is the square edge
  CALL CALC_DIAG             ;Subroutine call
  WAIT 1
  JUMP MAIN
END

;-------------This subroutione calcs the square diagonal-----------------
SUB CALC_DIAG
  gsYourSing = SQRT(2 * POW(glOurlong,2))
ENDSUB
```

7.2.10 NOP

No operation; questa istruzione non ha nessun effetto sul programma e può essere utilizzata per introdurre dei ritardi.

7.3 Istruzioni dedicate alle uscite digitali

Sono le istruzioni che permettono di attivare o disattivare un'uscita digitale. L'attivazione e la disattivazione possono essere eseguite anche tramite l'operatore di assegnamento (<nome uscita> = 1) ma le istruzioni dedicate sono più veloci.

7.3.1 SETOUT

Attiva una uscita digitale. La sintassi è la seguente:

```
SETOUT <nome uscita>
```

7.3.2 RESOUT

Disattiva una uscita digitale. La sintassi è la seguente:

```
RESOUT <nome uscita>
```

L'uscita ofOutput1 viene attivata con il comando SETOUT quando la variabile sbSeconds assume il valore 20 solo se l'ingresso ifEnableO1 e attivo. Tale uscita viene poi resettata quando sbSecondi assume il valore 40.

Introduciamo un esempio in cui vengono cambiate di stato un'uscita a seconda del valore di una variabile:

```
MAIN:
  IF (sbSeconds EQ 20) AND ifEnableO1   ;If sbSeconds is equal to 20 and the input si actived
    SETOUT ofOutput1                    ;set the output
  ENDIF
  IF sbSecondi EQ 40                    ;When the sbSeconds is equal to 40
    RESOUT ofOutput1                    ;reset the output
  ENDIF

  WAIT 1
  JUMP MAIN
END
```

7.4 Istruzioni di sistema

Sono le istruzioni che permettono il controllo dell'esecuzione delle unit (unità contenenti il codice QCL o LADDER) del programma utente. Tali istruzioni saranno più comprensibili quando sarà spiegato il sistema di multitasking adottato.

7.4.1 SUSPEND

Esegue la sospensione dell'esecuzione della unit di programma specificato.
Tale istruzione può essere utilizzata da una unit per bloccare l'esecuzione di un altra unit oppure per bloccare l'esecuzione di se stessa. In questo caso bisogna ricordare che la unit in questione, essendo sospesa, non può più riattivarsi ma tale operazione deve essere eseguita da una delle altre unit in esecuzione. La sintassi dell'istruzione SUSPEND è:

```
  <unit name>.SUSPEND
```

7.4.2 RESUME

Esegue il ripristino dell'esecuzione delle istruzioni di una unit sospesa. Tale istruzione fa riprendere l'esecuzione di una unit sospesa dall'istruzione successiva a quella eseguita al momento della sospensione. La sintassi è:

```
  <unit name>.RESUME
```

RESUME produce degli effetti solamente se la unit <unit name> era sospesa.

7.4.3 RESTART

Esegue una "ripartenza" dell'esecuzione di una unit di programma dalla prima istruzione. Tale istruzione non altera lo stato di sospensione della unit in questione; ciò significa che se viene eseguito un RESTART di una unit sospesa, quest'ultima si predispone per l'esecuzione della prima istruzione ma rimane tuttavia sospesa fino alla sua attivazione con l'istruzione RESUME. La sintassi è:

```
  <nome_task>.RESTART
```

Per comprendere meglio il significato di queste istruzioni di sistema si veda il capitolo "Multitasking".

7.5 Variabili di sistema

Il sistema QMOVE mette a disposizione delle variabili legate al sistema operativo interno di sola lettura; i nomi di queste

variabili sono predefiniti, non possono cioè essere cambiati dall'utente.

7.5.1 is_suspended

Fornisce lo stato di una unit:

- 0 = unit in esecuzione;
- 1 = unit sospesa.

La sintassi è:

```
<unit name>.is_suspended
```

7.5.2 watchdog

Il *Watchdog task* indica che il tempo di esecuzione di una unit del progetto ha superato i 200 ms. È la stessa informazione che compare nel pannello *Monitor - CPU* alla voce *Watchdog Task*. La sintassi è:

```
QMOVE.watchdog
```

7.5.3 time_task_lost

Indica che una unit a tempo del progetto ha perso un evento. È la stessa informazione che compare nel pannello *Monitor - CPU* alla voce *Time task Lost*. Una unit a tempo deve essere eseguita con tempistiche deterministiche. Questi tempi potrebbero non venire rispettati a causa del grosso impegno nell'eseguire il codice inserito nella stessa unit. Se questo succede si parla di *Time task lost* avvenuto.
La sintassi è:

```
QMOVE.time_task_lost
```

7.5.4 battery_low

Indica che la batteria ha un livello di carica basso. Il led BATT lampeggia. Questa informazione ha senso solamente nei modelli di QMOVE forniti di batteria tampone per la memoria RAM.
La sintassi è:

```
QMOVE.battery_low
```

7.5.5 battery_down

Indica che la batteria è scarica. Il led BATT è acceso. Questa informazione ha senso solamente per i modelli di QMOVE che sono dotati di batteria tampone per la memoria RAM.
La sintassi è:

```
QMOVE.battery_down
```

Esistono altre variabili di sistema che dipendono dal firmware della CPU utilizzata. Per accedere a queste variabili si utilizza la sintassi:

```
QMOVE.sys001
QMOVE.sys002
....
QMOVE.sys016
```

Il significato di queste variabili deve essere ricercato nella documentazione firmware della CPU.

7.6 Funzioni matematiche

7.6.1 Elevamento a potenza

Esegue l'elevamento alla potenza della base. La sintassi è:

```
POW(<base>, <exponent>)
```

Esempio: gsMaxVal = 2^{gwNbit}

```
gsMaxVal = POW(2,gwNbit)
```

7.6.2 Radice quadrata

Calcola la radice quadrata dell'argomento dato. La sintassi è:

```
  SQRT(<radicand>)
```

Esempio: gsIpot = v(gsL1^2 + gsL2^2)

```
  gsIpot = SQRT(POW(gsL1,2)+POW(gsL2,2))
```

7.6.3 Logaritmo naturale

Calcola il logaritmo naturale dell'argomento dato. La sintassi è:

```
  LN(<val>)
```

Esempio: gsValue = ln gsMaximum

```
  gsValue = LN(gsMaximun)
```

7.6.4 Esponenziale

Esegue l'elevamento a potenza del numero di Nepero. La sintassi è:

```
  EXP(<exponent>)
```

Esempio: gsA = e^2

```
  gsA =EXP(2)
```

7.6.5 Valore assoluto

Ritorna il valore assoluto dell'argomento. La sintassi è:

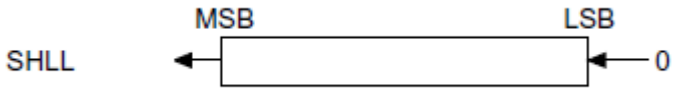
```
  ABS(<variable>)
```

Esempio: glModule = I glValue I

```
  glModule = ABS(glValue)
```

7.6.6 Shift a sinistra

Esegue una operazione di shift a sinistra di *n* bit sull'argomento.



Lo shift è da intendersi indifferentemente sia *logico* che *aritmetico*, ossia:

- I bit che escono dalla sequenza a sinistra vengono perduti.
- I bit che entrano nella sequenza a destra sono degli 0.

Dal punto di vista matematico si tratta della moltiplicazione dell'argomento per la potenza 2^n.
La sintassi è:

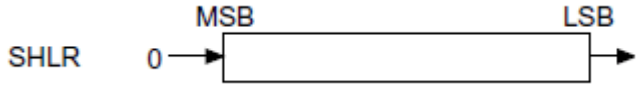
```
  SHLL(<variable>,<bits>)
```

Esempio:

```
  glNewValue = SHLL(glValue, 1)
  ; Se glValue = &B00001100 (12) allora glNewValue = &B00011000 (24)
```

7.6.7 Shift a destra

Esegue una operazione di shift a destra di *n* bit sull'argomento.



Lo shift è da intendersi *aritmetico* (non *logico*), ossia:

- I bit che escono dalla sequenza a destra vengono perduti.
- I bit che entrano nella sequenza a sinistra sono delle copie del bit più significativo (ossia di quello più a sinistra).

Dal punto di vista matematico si tratta della divisione dell'argomento per la potenza 2^n. Questa interpretazione continua a valere anche per i numeri negativi scritti in complemento a 2.
La sintassi è:

```
SHLR(<variable>,<bits>)
```

Esempio:

```
glValue = SHLR(glValue, 1)
; Se glValue = &B00001100 (12) allora glNewValue = &B00000110 (6)
; Se glValue = &B11110011 (-12) allora glNewValue = &B11111001 (-6)
```

7.6.8 Moltiplicazione e divisione

Esegue una moltiplicazione di valori interi a 32 bit con risultato a 64 bit e una successiva divisione per valore intero a 32 bit. Il risultato è un valore di 32 bit.
N.B. Se viene eseguita una divisione per zero, la CPU andrà in stato “Division by zero error”.
La sintassi è:

```
MULDIV(<factor1>,<factor2>,<divisor>)
```

Esempio: glValue = a * b / c

```
glValue = MULDIV(a, b, c)
```

7.6.9 Resto di moltiplicazione e divisione

Esegue una moltiplicazione di valori interi a 32 bit con risultato a 64 bit e una successiva divisione per valore intero a 32 bit. Il risultato è il resto della divisione.
N.B. Se viene eseguita una divisione per zero, la CPU andrà in stato “Division by zero error”.
La sintassi è:

```
RMULDIV(<factor1>,<factor2>,<divisor>)
```

Esempio: glValue = a * b % c

```
glValue = RMULDIV(a, b, c)
```

7.6.10 Arrotondamenti all'intero più vicino

ROUND	Esegue un arrotondamento di un valore in floating point all'intero più vicino.
TRUNC	Esegue un arrotondamento di un valore in floating point all'intero più vicino non maggiore di grandezza rispetto al valore dato.
FLOOR	Esegue un arrotondamento di un valore in floating point all'intero più vicino non superiore al valore dato.
CEIL	Esegue un arrotondamento di un valore in floating point all'intero più vicino non inferiore al valore dato.

La sintassi è:

```
ROUND(<variable>)
TRUNC(<variable>)
FLOOR(<variable>)
CEIL(<variable>)
```

Esempio:

```
gsValue = ROUND(2.7)      ;risulta 3.0
gsValue = ROUND(-2.7)     ;risulta -3.0
gsValue = TRUNC(2.7)      ;risulta 2.0
gsValue = TRUNC(-2.7)     ;risulta -2.0
gsValue = FLOOR(2.7)      ;risulta 2.0
gsValue = FLOOR(-2.7)     ;risulta -3.0
gsValue = CEIL(2.7)       ;risulta 3.0
gsValue = CEIL(-2.7)      ;risulta -2.0
```

7.6.11 Operatori di classificazione

ISFINITE	Controlla se il numero dato ha valore finito.
ISINF	Controlla se il numero dato ha valore infinito.
ISNAN	Controlla se il numero dato ha valore NaN.
ISNORMAL	Controlla se il numero dato ha valore normale.

La sintassi è:

```
ISFINITE(<variable>)
ISINF(<variable>)
ISNAN(<variable>)
ISNORMAL(<variable>)
```

Esempio:

```
gfValue = ISFINITE(gsValue)     ;risulta 1 se gsValue è un valore finito, altrimenti 0
gfValue = ISINF(gsValue)        ;risulta 1 se gsValue è un valore infinito, altrimenti 0
gfValue = ISNAN(gsValue)        ;risulta 1 se gsValue è un NaN (Not a Number), altrimenti 0
gfValue = ISNORMAL(gsValue)     ;risulta 1 se gsValue è un valore normale (non zero, non infinito, non NaN), altrimenti 0
```

7.7 Funzioni trigonometriche

7.7.1 Seno

Calcola il seno di un angolo espresso in radianti. La sintassi è:

```
SIN(<angle>)
```

Esempio: gsYPos = ssRadius x SIN gsalpha

```
gsYPos = ssRadius * SIN(gsalpha)
```

7.7.2 Coseno

Calcola il coseno di un angolo espresso in radianti. La sintassi è:

```
COS(<angle>)
```

Esempio: gsXPos = ssRadius x COS gsalpha

```
gsXPos = ssRadius * COS(gsalpha)
```

7.7.3 Tangente

Calcola la tangente di un angolo espresso in radianti. La sintassi è:

```
TAN(<angle>)
```

Esempio: gsMyVal = TAN gsalpha

```
gsMyVal = TAN(gsalpha)
```

7.7.4 Cotangente

Calcola la cotangente di un angolo espresso in radianti. La sintassi è:

```
COT(<angle>)
```

Esempio: gsMyVal = COT gsalpha

```
gsMyVal = COTG(gsalpha)
```

7.7.5 Arcoseno

Calcola l'angolo, espresso in radianti, il cui seno è uguale all'argomento. La sintassi è:

```
ASIN(<arc>)
```

Esempio: gsAngolo = ASIN Arc

```
gsAngolo = ASIN(Arc)
```

7.7.6 Arcocoseno

Calcola l'angolo, espresso in radianti, il cui coseno è uguale all'argomento. La sintassi è:

```
ACOS(<arc>)
```

Esempio: gsAngolo = ACOS Arc

```
gsAngolo = ACOS(Arc)
```

7.7.7 Arcotangente

Calcola l'angolo, esresso in radianti, la cui tangente è uguale all'argomento. La sintassi è:

```
ATAN(arc)
```

Esempio: gsMyVal = ATAN Arc

```
gsMyVal = ATAN(Arc)
```

7.7.8 Avvertenza

In generale, nelle funzioni trigonometriche, gli angoli vengono espressi in radianti. Essendo p un numero irrazionale (non finito) e potendolo rappresentare con una precisione di 7 cifre dopo il punto decimale, si introduce una approssimazione nei calcoli trigonometrici.
Altro limite è dato dai calcoli trigonometrici che danno come risultato un numero infinito, che non può sicuramente essere rappresentato con un numero floating point a singola precisione (7 cifre). Per esempio il risultato della tangente di p/2 è un numero molto grande in modulo con segno negativo, risultato che evidentemente si allontana da +infinito (risultato corretto).

ESEMPIO: tang p/2 = tang 1.570796371 = -22877332 ;
arctg -22877332 = -1.570796371 = - p/2 (!!)

Non è possibile calcolare la tangente di p/2 con la formula:
tang p/2 = (sen p/2) / (cos p/2)
in quanto viene eseguita una divisione per zero.

8. Applicazione di esempio

L'applicazione che vogliamo sviluppare ha lo scopo di applicare le informazioni finora introdotte.
Il progetto che andremo a realizzare avrà:

- una unit per gestire una macchina a stati;
- una unit per realizzare un sequenziatore;
- una unit per rilevare gli ingressi digitali ed attivare uscite digitali.

8.1 Nuovo Progetto

Crea un nuovo progetto con

- File - New Project

inserisci il *Project Name* come *MyApplication*.

8.2 Aggiungi la unit di configurazione

Seleziona

- File - Add Unit - Configuration Unit

e chiamala CONFIG. La unit verrà già precompilata con tutte le parole chiavi che distinguono i vari gruppi di variabili. Nella unit di configurazione è fondamentale, per ora, dichiarare quale hardware si vuole utilizzare.
In questo esempio abbiamo usato

```
...
BUS
  1  1P51F  30
  2  .
  3  IMG8F  .
```

Per ora non dichiariamo niente altro nella unit di configurazione.

8.3 Aggiungi le unit QCL

8.3.1 Unit MANAGER

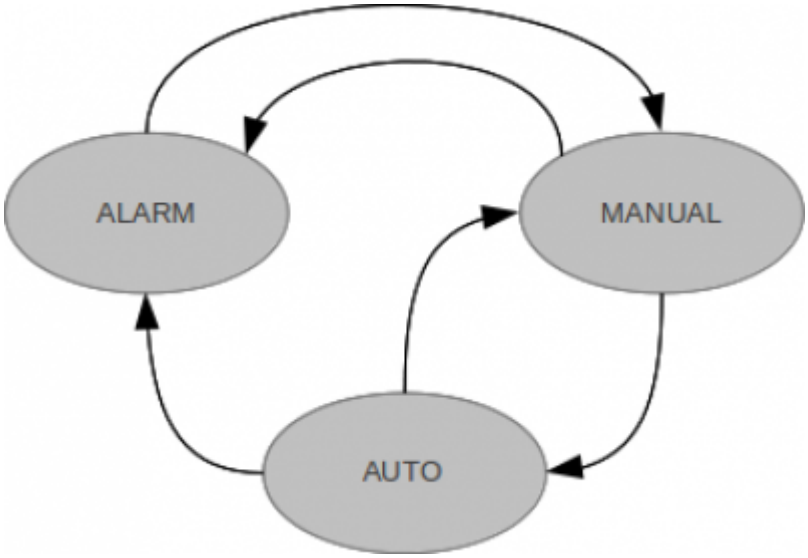
Seleziona

- File - Add Unit - Qcl Unit

e chiamala *MANAGER*. Lascia la proprietà ***Task Mode*** come *Normal*. La unit aggiunta presenta il seguente codice pre-inserito:

```
MAIN:
  WAIT 1
  JUMP MAIN
END
```

Questa unit *MANAGER* dovrà gestire una macchina a stati rappresentata nel seguente schema:



Il software della unit *MANAGER* dovrà gestire le transizioni di stato e dovrà fornire alcune informazioni (per esempio quale è lo stato attuale).
Dichiara le seguenti variabili e costanti nella unit *MANAGER*:

```
CONST
  ALARM_STATE      0  OUT
  MANUAL_STATE     1  OUT
  AUTOMATIC_STATE  2  OUT

GLOBAL
  NewState  B  IN    ;New state requested
  State     B  OUT   ;Actual state

BEGIN
MAIN:

  WAIT 1
  JUMP MAIN
END
```

Ora aggiungi il seguente codice:

```
CONST
  ALARM_STATE      0  OUT
  MANUAL_STATE     1  OUT
  AUTOMATIC_STATE  2  OUT

GLOBAL
  NewState  B  IN    ;New state requested
  State     B  OUT   ;Actual state

BEGIN
MAIN:

  SWITCH State
  CASE ALARM_STATE
    ;Put here the code to do when the state is ALARM
  CASE MANUAL_STATE
    ;Put here the code to do when the state is MANUAL
  CASE AUTOMATIC_STATE
    ;Put here the code to do when the state is AUTOMATIC
  ENDSWITCH

  WAIT 1
  JUMP MAIN
END
```

La variabile *State* è considerata un OUT per questa unit. Se un'altra unit vuole far cambiare lo stato alla macchina non può agire direttamente su *State* perché solo la unit *MANAGER* la può modificare. Viene quindi predisposta una variabile *NewState* che viene usata da *MANAGER* come un IN nel seguente modo

```
CONST
  ALARM_STATE      0  OUT
  MANUAL_STATE     1  OUT
  AUTOMATIC_STATE  2  OUT

GLOBAL
  NewState  B  IN    ;New state requested
  State     B  OUT   ;Actual state

BEGIN
MAIN:

  SWITCH State
  CASE ALARM_STATE
    ;Put here the code to do when the state is ALARM
  CASE MANUAL_STATE
    ;Put here the code to do when the state is MANUAL
  CASE AUTOMATIC_STATE
    ;Put here the code to do when the state is AUTOMATIC
  ENDSWITCH

  ;Check if there is a change state request
  IF State NEQ NewState
    SWITCH NewState
    CASE ALARM_STATE

      SWITCH State
      CASE MANUAL_STATE
```

```
        ;Put here the code for MANUAL to ALARM trasition
      CASE AUTOMATIC_STATE
        ;Put here the code for AUTO to ALARM trasition
      ENDSWITCH

    CASE MANUAL_STATE

      SWITCH State
      CASE ALARM_STATE
        ;Put here the code for ALARM to MANUAL trasition
      CASE AUTOMATIC_STATE
        ;Put here the code for AUTO to MANUAL trasition
      ENDSWITCH

    CASE AUTOMATIC_STATE

      SWITCH State
      CASE MANUAL_STATE
        ;Put here the code for MANUAL to AUTO trasition
      ENDSWITCH

    ENDSWITCH

    State = NewState  ;The trasition is done

  ENDIF

  WAIT 1
  JUMP MAIN
END
```

Come puoi osservare, quando la *NewState* è diversa da *State* viene eseguito uno dei possibili codici associati ad una delle 5 possibili transizioni. Alla fine della transizione lo stato della macchina viene definitivamente impostato al nuovo stato richiesto. Una unit che volesse utilizzare la unit MANAGER dovrà quindi usare un codice di questo tipo:

```
  MANAGER.NewState = MANAGER.AUTOMATIC_STATE
  WAIT MANAGER.State EQ MANAGER.NewState
```

Per esempio per comandare il cambio di stato in automatico.

8.3.2 Unit SEQUENC

Aggiungi un'altra unit con la stessa proprietà di *MANAGER* e chiamala *SEQUENC*. Per ottenere una sequenza devi scrivere un codice come questo:

```
GLOBAL
  Start           F  IN   ;Start sequence input
  Sequence        W  OUT  ;This is the step of the sequence

  start_rise_up   F  ;Local variable to check the rise edge of Start input

BEGIN
MAIN:

  IF Start
    IF start_rise_up
      start_rise_up = 0
      Sequence = 10         ;Start the sequence
    ENDIF
  ELSE
    start_rise_up = 1
    Sequence = 0            ;Reset the sequence
  ENDIF

  WAIT 1
  JUMP MAIN
END
```

Come puoi osservare quando la variabile *Start* vale 0 (FALSE) la variabile *Sequence* viene azzerata (questo serve per impedire alla sequenza di funzionare). Quando la variabile *Start* vale 1 (TRUE) la variabile *Sequence* viene impostata al valore 10 solo la prima volta perché poi la variabile *start_rise_up* viene impostata a 0.
Ora aggiungi il codice per usare la variabile *Sequence*.

```
CONST
  OFF  0  OUT
  ON   1  OUT

GLOBAL
  Start           F  IN   ;Start sequence input
  Force           F  IN   ;Force the signal output to ON
  Sequence        W  OUT  ;This is the step of the sequence
  Signal          F  OUT  ;This is an output signal

  start_rise_up   F  ;Local variable to check the rise edge of Start input

TIMER
  time_on
  time_off

BEGIN
MAIN:

  IF Start
    IF start_rise_up
      start_rise_up = 0
      Sequence = 10         ;Start the sequence
    ENDIF
  ELSE
    start_rise_up = 1
    Sequence = 0            ;Reset the sequence
    Signal = Force          ;Force the signal
  ENDIF

  ;This is the sequence management
  IF Sequence EQ 10         ;First step
    Signal = OFF            ;Reset signal
```

```
    time_off = 1000         ;1000 ms = 1 s
    Sequence = 20
  ENDIF
  IF Sequence EQ 20         ;Second step
    IF time_off             ;Wait time expired
      Sequence = 30
    ENDIF
  ENDIF
  IF Sequence EQ 30         ;3rd step
    Signal = ON             ;Set signal
    time_on = 500           ;500 ms = 0.5 s
    Sequence = 40
  ENDIF
  IF Sequence EQ 40         ;4th step
    IF time_on              ;wait time expired
      Sequence = 50
    ENDIF
  ENDIF
  IF Sequence EQ 50         ;5th step
    Sequence = 10
  ENDIF

  WAIT 1
  JUMP MAIN
END
```

Quando la variabile *Sequence* è impostata a 10, viene eseguito il codice contenuto in

```
  IF Sequence EQ 10
    ...
```

che la fa diventare 20 e via così eseguendo passo dopo passo la sequenza. La sequenza consiste nell'impostare a OFF la variabile *Signal* per 1s e quindi impostare ad ON per 0.5s e quindi ripetere la sequenza all'infinito (finché *Start* vale TRUE). Inoltre è stato aggiunto anche l'input *Force* che, quando la sequenza non è attiva, serve per forzare *Signal* a ON .

8.3.3 Unit COMMAND

Aggiungi la unit COMMAND e inserisci il seguente codice:

```
GLOBAL
  ForceOutput    F  OUT

INPUT
  Selector       F  REFERENCE
  Enable         F  REFERENCE
  ManualButton   F  REFERENCE

BEGIN
MAIN:

  WAIT 1
  JUMP MAIN
END
```

In questa unit dichiariamo tre *REFERENCE* ad tre ingressi digitali. Poi vedremo come assegnare questi *REFERENCE* a tre ingressi digitali veri e propri che andremo ad aggiungere nella unit di configurazione. Ora aggiungi il seguente codice:

```
GLOBAL
  ForceOutput    F  OUT

INPUT
  Selector       F  REFERENCE
  Enable         F  REFERENCE
  ManualButton   F  REFERENCE

BEGIN
MAIN:

  IF Enable  ;If enable input is on
    IF Selector  ;If selector in on automatic position
      ;Go to automatic state
      MANAGER.NewState = MANAGER.AUTOMATIC_STATE
      WAIT MANAGER.State EQ MANAGER.AUTOMATIC_STATE
    ELSE
      ;Go to manual state
      MANAGER.NewState = MANAGER.MANUAL_STATE
      WAIT MANAGER.State EQ MANAGER.MANUAL_STATE
    ENDIF
  ELSE
    ;Go to alarm state
    MANAGER.NewState = MANAGER.ALARM_STATE
    WAIT MANAGER.State EQ MANAGER.ALARM_STATE
  ENDIF

  ForceOutput = ManualButton  ;Replay  the input value to an output variable

  WAIT 1
  JUMP MAIN
END
```

Questo codice serve per decidere quando eseguire le transizioni di stato. Ora è importante completare il codice scritto in *MANAGER* in questo modo:

```
CONST
  ALARM_STATE      0  OUT
  MANUAL_STATE     1  OUT
  AUTOMATIC_STATE  2  OUT

GLOBAL
  NewState  B  IN    ;New state requested
  State     B  OUT   ;Actual state

BEGIN
MAIN:

  SWITCH State
```

```
  CASE ALARM_STATE
    ;Put here the code to do when the state is ALARM
    SEQUENC.Start = SEQUENC.OFF
    SEQUENC.Force = SEQUENC.OFF
  CASE MANUAL_STATE
    ;Put here the code to do when the state is MANUAL
    SEQUENC.Force = COMMAND.ForceOutput
  CASE AUTOMATIC_STATE
    ;Put here the code to do when the state is AUTOMATIC
    SEQUENC.Start = SEQUENC.ON
  ENDSWITCH

  ;Check if there is a change state request
  IF State NEQ NewState
    SWITCH NewState
    CASE ALARM_STATE

      SWITCH State
      CASE MANUAL_STATE
        ;Put here the code for MANUAL to ALARM trasition
        SEQUENC.Start = SEQUENC.OFF
        SEQUENC.Force = SEQUENC.OFF
      CASE AUTOMATIC_STATE
        ;Put here the code for AUTO to ALARM trasition
        SEQUENC.Start = SEQUENC.OFF
        SEQUENC.Force = SEQUENC.OFF
      ENDSWITCH

    CASE MANUAL_STATE

      SWITCH State
      CASE ALARM_STATE
        ;Put here the code for ALARM to MANUAL trasition
        SEQUENC.Start = SEQUENC.OFF
        SEQUENC.Force = SEQUENC.OFF
      CASE AUTOMATIC_STATE
        ;Put here the code for AUTO to MANUAL trasition
        SEQUENC.Start = SEQUENC.OFF
        SEQUENC.Force = SEQUENC.OFF
      ENDSWITCH

    CASE AUTOMATIC_STATE

      SWITCH State
      CASE MANUAL_STATE
        ;Put here the code for MANUAL to AUTO trasition
        SEQUENC.Start = SEQUENC.OFF
        SEQUENC.Force = SEQUENC.OFF
      ENDSWITCH

    ENDSWITCH

    State = NewState  ;The trasition is done

  ENDIF

  WAIT 1
  JUMP MAIN
END
```

Come si vede, in questo semplice esempio, le operazioni che vengono eseguite durante le transizioni di stato sono tutte uguali e servono per bloccare qualsiasi operazione sulla variabile *Signal*. Poi si vede che quando lo stato è in manuale l'attivazione di *Signal* è affidata a *COMMAND.ForceOutput* che a sua volta ha lo stesso valore di un ingresso digitale. Nello stato automatico invece viene attivata la seguenza gestita nella unit *SEQUENC*.
Ora facciamo l'ultima aggiunta alla unit di configurazione *CONFIG* in questo modo:

```
INPUT
  ManAuto   F    3.INP01
  PowerOn   F    3.INP02
  Push          F   3.INP03

REFERENCES
  COMMAND.Selector     = ManAuto
  COMMAND.Enable       = PowerOn
  COMMAND.ManualButton = Push
```

In questo modo i simboli della unit *COMMAND*: *Selector*, *Enable*, *ManualButton* sono dei riferimenti associati agli ingressi dichiarati come: *ManAuto*, *PowerOn*, *Push* rispettivamente. E' evidente che la funzionalità dei REFERENCE in questo esempio è poco utile. Infatti al posto del simbolo *Selector* avrei potuto usare direttamente *ManAuto*. Si apprezza la potenzialità dei REFERENCE se si vuole riutilizzare il codice scritto per la unit *COMMAND*. Infatti potrei aggiungere una nuova unit chiamata *COMMAN2* contenente lo stesso indentico codice QCL di *COMMAND* ma con i REFERENCE riferiti ad altri input. Ottengo in questo modo un codice perfettamente riutilizzabile.

9. Funzioni di libreria QCL

Una funzione QCL è una parte di codice che permette di risolvere particolari problemi, di eseguire elaborazioni dati, di fornire specifiche funzionalità ad un progetto per QMOVE. Per poter utilizzare una funzione QCL nel codice di una unit è sufficiente richiamarla passandole gli argomenti necessari, come si trova descritto in seguito.
Questo modo di procedere comporta alcuni vantaggi:

- Si trovano parti di codice già scritte, collaudate, che risolvono in maniera ottimizzata i piú frequenti problemi che si affrontano nella stesura di un progetto.
- Offrono la possibilità di riutilizzare quante volte si vuole la stessa funzione, senza doversi preoccupare di riscriverla ogni volta.
- Esiste la possibilità di aggiornare semplicemente la libreria man mano che questa viene dotata di nuove funzioni, mantenendo una assoluta compatibilità con gli applicativi già sviluppati.

La lista delle funzioni QCL disponibili e la relativa modalità di utilizzo è disponibile nel menù ***Help > Functions info***, richiamabile in qualsiasi momento dall'ambiente di sviluppo.

Se indichiamo con *FuncQCL01* una qualsiasi funzione della libreria, la sintassi corretta per richiamarla è la seguente:

```
  FuncQcl01(<arguments list>)
```

Non restituendo valori, le funzioni non possono essere utilizzate a destra di un assegnazione o come parte di un'espressione in un istruzione IF, FOR o WHILE.

```
slType01 = FuncQCL01(...)                       !COMPILING ERROR!
IF (FuncQCl01(...) AND ...)                     !COMPILING ERROR!
FOR (sInt01 = 1, sInt01 LT FuncQCL01(...), ...) !COMPILING ERROR!
WHILE (FuncQCL01(...) LT 24)                    !COMPILING ERROR!
```

Anche se una funzione QCL non restituisce alcun valore, è tuttavia possibile che una funzione QCL imposti dei valori a dei parametri passati come argomento e quindi in pratica è come se si avessero piú valori di ritorno.

Vediamo alcuni semplici esempi:

```
  AC10AvergArr (MyArray, average_value, ok_calc)
```

La funzione calcola il valore medio di un array passato come argomento. In questo caso *MyArray* è un parametro di ingresso e si vuol calcolarne il valore medio.
average_value e *ok_calc* sono parametri di uscita e sono rispettivamente il valore medio calcolato dalla funzione e un flag indicante il calcolo completato. Quando *ok_calc* assume valore 1 significa che la funzione ha terminato i suoi calcoli e quindi è possibile recuperare tale valore leggendo la variabile *average_value*.

Per ciascuna funzione è poi indicato in quale parte dell'unità chiamarla; infatti vi sono funzioni, che proprio per le operazioni che devono svolgere, vanno chiamate in una parte ciclica del codice, mentre alcune altre possono essere chiamate anche in una parte non ciclica del codice. Con parte ciclica si intende la parte del codice QCL che viene eseguita ad ogni scansione del programma.

Con parte non ciclica si intende la parte di codice che non viene eseguita ciclicamente perchè mancano determinate condizioni (per esempio se il codice è contenuto in una istruzione IF la cui condizione non è vera).
L'indicazione sul "dove" richiamare una funzione è riportato nella descrizione di ciascuna funzione (***Help > Functions info***).
Un'altra caratteristica da tenere presente per alcune funzioni è il fatto che, quando sono necessarie operazioni che impiegano molto tempo, la funzione stessa si preoccupa di eseguire un'istruzione di WAIT al suo interno, ogni 180 millisecondi. Questo per impedire il "blocco" dell'esecuzione della unit dovuto a compiti particolarmente gravosi. Solitamente in funzioni di questo tipo è presente un argomento di tipo flag che puó essere monitorato per verificare quando la funzione ha terminato il suo compito.
Nell'help della funzione utilizzata viene indicato chiaramente il tipo degli argomenti che la funzione si aspetta. Se questo tipo non viene rispettato dal programmatore viene generato un errore in fase di compilazione del progetto.

9.0.1 Esempio di utilizzo delle funzioni QCL

Supponiamo di utilizzare la funzione di calcolo della media aritmetica dei valori contenuti in un array (AC10AvergArr):

```
GLOBAL
  glAverage L           ; This is used as the result variable
  gfDone F              ; This flag is used to result variable
ARRSYS
  aslMyArray 10 L       ; Array System

BEGIN

MAIN:
  IF NOT gfDone
    AC10AvergArr(aslMyArray, glAverage, gfDone)
  ENDIF
  WAIT 1
```

```
  JUMP MAIN
END
```

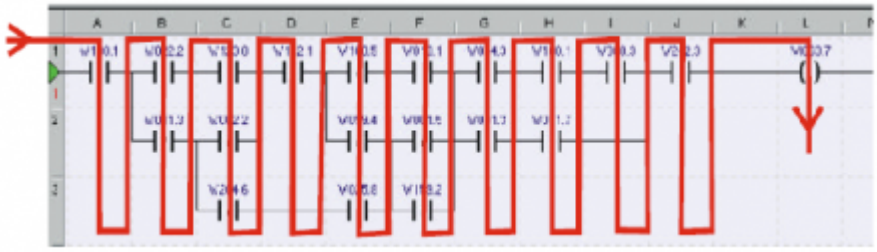
10. Editor Ladder

Il Ladder (logica a contatti) è un linguaggio nato per la programmazione dei PLC, per cui è molto performante per la gestione di ingressi uscite e piccole sequenze.
Si dà per scontato che l'utilizzatore conosca il linguaggio Ladder IEC1131; in caso contrario si consiglia di integrare il presente manuale con un corso PLC generico.

10.1 L'esecuzione del rung

Nel linguaggio LADDER bisogna tener presente che i vari elementi sono analizzati, all'interno del rung, partendo dall'angolo in alto a sinistra e spostandosi dall'alto in basso e da sinistra a destra (Figura 1).

Figura 1: esecuzione del rung.

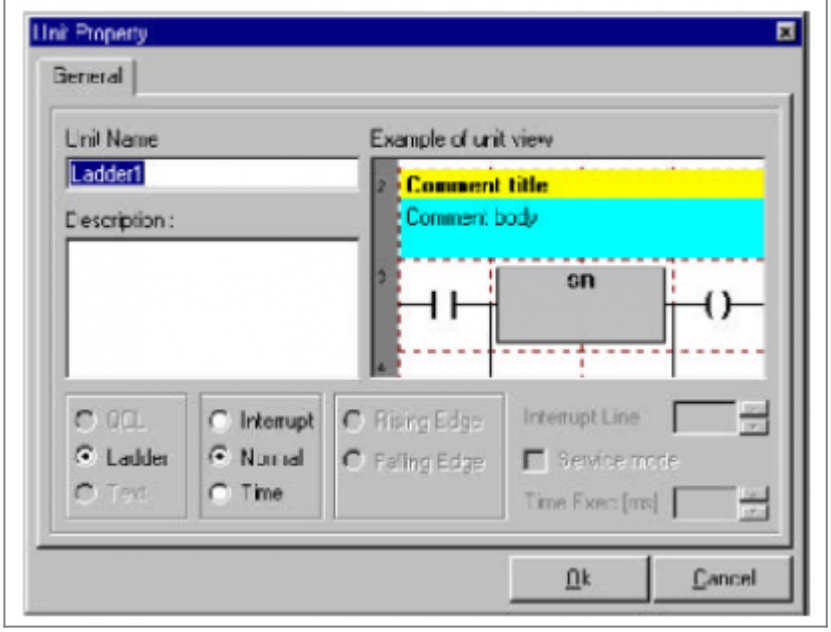


Nel caso si inserisca un elemento che occupa più celle (ad esempio un contatore), la cella che fa da riferimento per lo svolgimento del rung è quella in alto a sinistra.
Il nome delle variabili per il programma Ladder sono dichiarate nella unità di configurazione e vanno dichiarate come descritto nella sezione QCL. Le stesse variabili possono essere utilizzate sia in Ladder che in QCL anche contemporaneamente.
In seguito si guida l'utente all'inserimento di un elemento nella rete LADDER utilizzando le voci del menù predisposte.

10.2 Come inserire un elemento

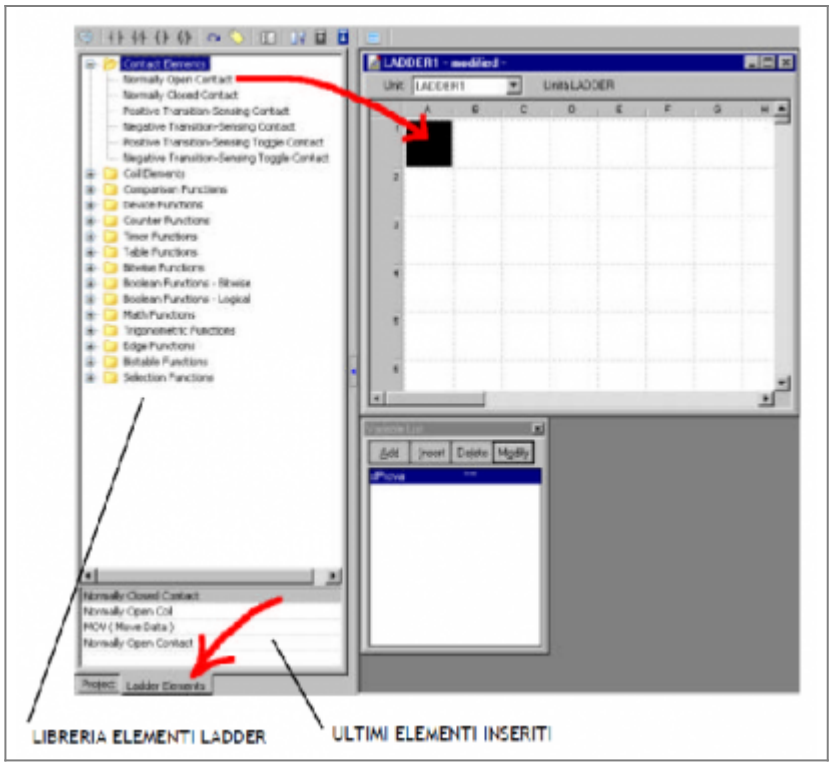
Spieghiamo con un esempio come inserire un elemento in una rete LADDER. Innanzitutto si deve inserire una nuova unità LADDER selezionando ***File > Add unit > Ladder unit***. Comparirà la finestra di Figura 2.

Figura 2: inserimento di nuova unità LADDER.



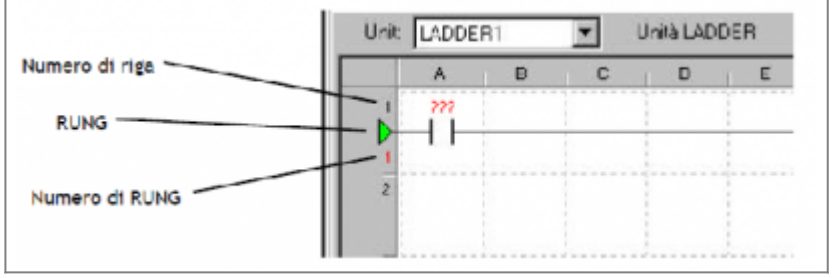
Una volta confermato con *Ok* una nuova unità verrà ad aggiungersi nella finestra **UNIT MANAGER**. Con un doppio click su questa nuova unità LADDER si aprirà l'editor LADDER. A questo punto è possibile inserire il primo elemento della nostra rete LADDER utilizzando la "libreria degli elementi LADDER" come mostrato in figura 3.

Figura 3: Libreria dei blocchi LADDER



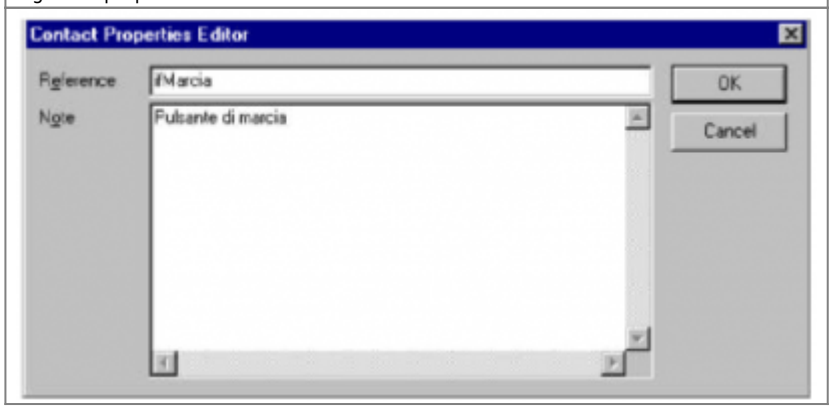
A questo punto è possibile scegliere tra i vari elementi messi a disposizione e posizionarli sulla griglia dell'editor. L'operazione si può fare sia trascinando l'elemento sulla posizione voluta, sia facendo un doppio click sull'elemento stesso. In questo caso l'elemento viene posizionato nella casella dell'editor evidenziata in quel momento.
E' possibile selezionare l'elemento ladder da inserire scegliendolo anche dalla lista degli "ultimi elementi inseriti".
L'elemento ladder deve essere inserito in una rete esistente. Se la rete non esiste bisogna crearla inserendo un nuovo "rung".
Per inserire un nuovo "rung" selezionare il menù ***Edit > New Rung***.
Se supponiamo di voler inserire il contatto normalmente aperto si deve selezionare Normally Open Contact e sull'editor appare il simbolo del contatto (Figura 4).

Figura 4: contatto normalmente aperto inserito in una rete LADDER.



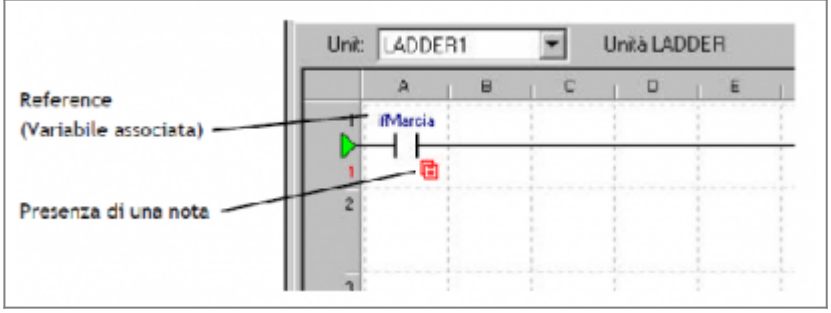
La libreria degli elementi LADDER è formata da una serie di categorie in cui sono suddivisi i vari elementi disponibili per comporre la rete LADDER. Nel seguito faremo una veloce carrellata su queste categorie. Il programmatore tenga presente che la maggior parte degli elementi sono elementi di tipo standard (IEC1131) e tutti sono dotati di un help in linea richiamabile direttamente da Qview (selezionando l'elemento e premendo F1).
L'elemento inserito nella figura 4 non ha associata nessuna variabile. Per associare una variabile all'elemento inserito si deve evidenziare l'elemento e selezionare il menù ***Edit > Element Properties***. Apparirà la finestra di figura 5.

Figura 5: proprietà del blocco.



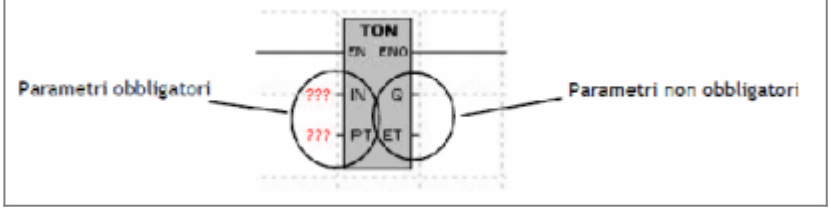
Il campo “Note” serve per associare all’elemento delle notazioni utili per il programmatore. Una volta compilata la finestra come in figura 5, la rete LADDER appare come in figura 6.

Figura 6: proprietà del blocco.



La finestra delle proprietà dell’elemento permette di inserire tutte le variabili di cui l’elemento necessita. Alcune variabili però non sono obbligatorie. Per esempio in figura 7 si osserva che alcuni parametri dell’elemento **TON (On Delay)** non sono obbligatori e questo è indicato dal fatto che non presentato i “???” (tre punti interrogativi).

Figura 7: TON: parametri obbligatori e non.



10.3 Categorie degli elementi

Le categorie degli elementi LADDER disponibili per comporre la rete LADDER sono:

Contact Elements:	elementi di contatto;
Coil Elements:	elementi con bobine di contatto;
Comparison Function:	confronto tra variabili;
Device Functions:	funzioni relative ai device;
Counter Functions:	elementi contatori;
Timer Functions:	temporizzatori;
Table functions:	funzioni su array;
Bitwise Functions:	operatori sul singolo bit;
Boolean Functions - Bitwise:	operatori binari a bit;
Boolean Functions - Logical:	operatori binari su variabili;
Math Functions:	operatori matematici;
Trigonometric Functions:	operatori trigonometrici;
Edge Functions:	Funzioni di trigger;
Bistable Functions:	Funzioni di set/reset;
Selection Functions:	selettori, multiplexer, limitatori.

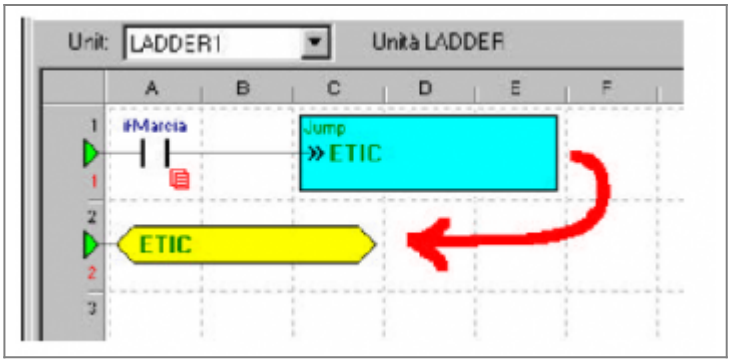
10.4 Commenti

Nella rete LADDER è possibile utilizzare una riga di caselle per poter inserire dei commenti. Per inserire un commento si deve selezionare del menù la voce ***Edit > New Element > Comment***.
Ogni commento può essere composto da un titolo e da un testo.

10.5 Jump e Label

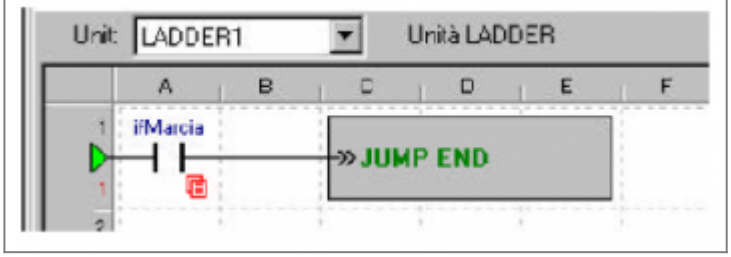
Nella struttura Ladder è prevista la possibilità di saltare ad una label in modo che il programma possa saltare delle parti gravose di programma quando non è neccessario eseguirle. Per inserire l’elemento di jump si deve selezionare dal menù la voce ***Edit > New Element > Jump***. Le proprietà di questo elemento consistono nel specificare l’etichetta associata al rung a cui saltare. Per inserire l’etichetta a cui saltare si deve agire al solito modo selezionando dal menù la voce ***Edit > New Element > Label***. Naturalmente non ci possono essere etichette (label) con lo stesso nome nella stessa unità (Figura 8).

Figura 8: Elemento di salto (JUMP) alla etichetta (LABEL) ETIC.



E’ possibile inserire anche l’elemento “Jump end” per saltare alla fine dell’unià direttamente senza bisogno di etichette (Figura 9).

Figura 9: Elemento di salto alla fine (JUMP END).



10.6 Movimento delle righe LADDER

Nella struttura Ladder è prevista la possibilità di muovere parti di programma verso l’alto e verso il basso, in modo da consentire al programmatore di inserire linee di programma non previste. Per muovere una linea si deve selezionare un rung presente e selezionare tramite le voci del menù ***Edit > Move Rows Up*** o ***Edit > Move Rows Down***.
Una volta terminato il task ladder, esiste la possibilità di compattare le linee di programma in modo di non lasciare spazi vuoti con il comando di ***Edit > Compact Rows***.

10.7 Drag & Drop degli elementi LADDER

Dalla parola stessa si intuisce che il comando serve a trascinare (drag) e posizionare (drop) uno o piú elementi ladder tramite mouse dopo averli opportunamente selezionati. Selezionando uno o piú elementi ladder e tenendo premuto il pulsante di sinistra del mouse sulla selezione, la selezione cambia colore ed il cursore del mouse cambia forma. È possibile in queste condizioni spostare la selezione. Durante lo spostamento della selezione, compare, nella status bar in basso a sinistra il messaggio: “Placing mode for drag and drop action”. Come per il comando *Paste* si possono presentare le seguenti situazioni:

- Drag & Drop eseguito in un'area libera da altri elementi ladder:
Una volta piazzato il riquadro, se si rilascia il pulsante del mouse, il riquadro colorato sparisce e l'elemento o gli elementi selezionati vengono inseriti in editor.

- Drag & Drop eseguito sopra o nell'area di altri elementi ladder:
come precedentemente descritto, una volta piazzato il riquadro, se si rilascia il pulsante del mouse, il riquadro colorato sparisce e viene visualizzato un box di messaggio che chiede di confermare o annullare l'operazione. Confermando l'operazione l'elemento o gli elementi vengono inseriti nella rete sottostante operando automaticamente tutti i collegamenti necessari per permettere una corretta compilazione del risultato finale.

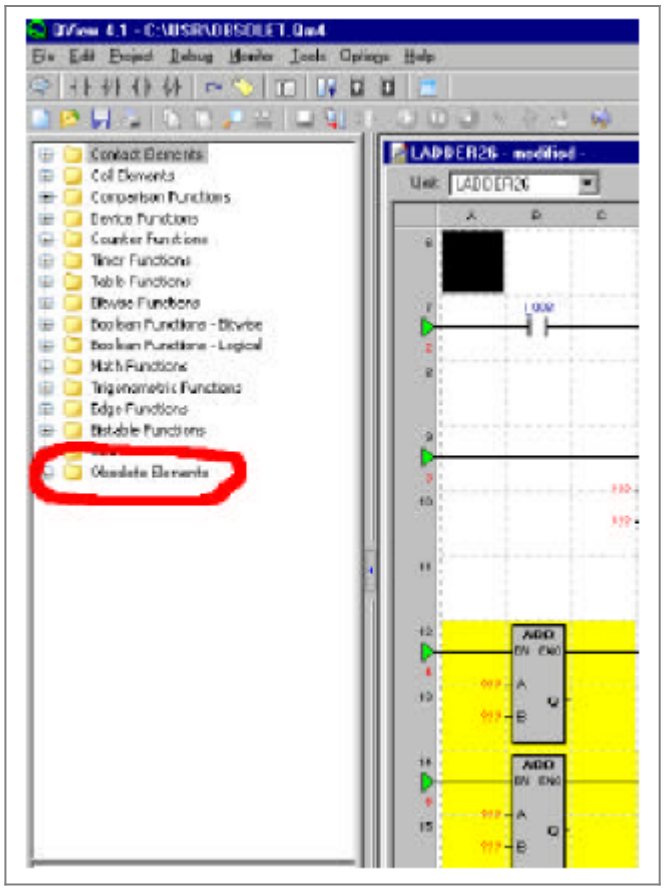
10.8 Elementi LADDER obsoleti

Gli **“Obsolete elements”**, sono elementi ladder utilizzati dai programmatori, ma che sono stati successivamente dichiarati obsoleti da QEM in qualità di fornitore e manutentore della libreria degli elementi ladder.

Generalmente, un elemento ladder puó venire marcato come obsoleto se è stato sostituito da un'equivalente elemento piú aggiornato, al quale per esempio, è stata aumentata l'efficienza del codice interno.

L'elemento obsoleto è sempre rimpiazzabile da un'equivalente elemento della libreria ladder in uso. In un progetto che utilizza degli elementi LADDER obsoleti essi sono riconoscibili dal fatto che vengono visualizzato con un colore di backgroud giallo (per default). La presenza di elementi obsoleti nel programma è facilemente constatabile dalla presenza della voce ***Edit > New Elements > Obsolete Elements*** e dalla cartella “Obsolete Elements” nella libreria degli elelenti (Figura 10).

Figura 10: Cartella “Obslete Elements”



Nelle impostazioni ***Options > Program Setup… > Ladder Editor*** è possibile modificare le impostazioni di default relative agli elementi ladder obsoleti.

10.9 Sostituzione degli elementi obsoleti

La funzionalità di sostituzione degli elementi obsoleti (**Sobstitute Obsolete element**) è stata inserita per operare la sostituzione di un elemento obsoleto inserito nella rete ladder, con un corrispondente elemento sostitutivo.
La funzionalità è attivata dalla voce di menù ***Edit > Sobstitute Obsolete Element***. Questa funzione è attivata se il cursore dell'editor ladder è posizionato sopra un'elemento obsoleto sostituibile.
Una volta sostituiti, gli ex elementi obsoleti cambiano il loro colore di background uniformandosi con gli elementi non obsoleti.
Utilizzando la voce di menù ***Edit > Sobstitute All Obsolete Elements*** si ha la possibilità, in una sola passata, di sostituire su tutto il progetto (tutte le unità ladder) di tutti gli elementi ladder obsoleti. Una volta avviata questa procedura si ricercano inizialmente gli elementi ladder obsoleti, appare il messaggio:

Searches for obsolete elements in progress…

Questo messaggio viene seguito da:

No obsolete elements to substitute

se non ci sono elementi obsoleti, oppure

There are obsolete ladder elements…

A questo punto è possibile scegliere di avviare la sostituzione completa o di abbandonare l'operazione. Una volta completata la sostituzione appare il messaggio:

All obsolete elements are substituted!

11. Multitasking

Il multitasking è la capacità di un sistema di gestire l'esecuzione contemporanea di diversi compiti.
Un progetto realizzato in QCL e ladder è formato da un insieme di unit o *tasks* (in italiano *compiti*, appunto). Il multitasking implementato nel sistema QMOVE è chiamato di tipo *cooperativo*, nel senso che l'esecuzione delle istruzioni passa da un task ad un altro solamente quando, nel task in esecuzione, viene incontrata una particolare istruzione

```
WAIT < condizione >.
```

Questo è vero per i task scritti in codice QCL, mentre per i task sviluppati in ladder il passaggio al task successivo avviene dopo l'ultima istruzione. La CPU smette di eseguire il codice di un task quando il task stesso viene messo in attesa tramite la istruzione WAIT.
Si schematizza il cosiddetto *ciclo di tasks* nella figura 1. Ogni task del progetto deve quindi poter *cooperare* con gli altri task mettendosi in uno stato di attesa del verificarsi di una condizione e quindi nel frattempo passare ad eseguire gli altri task. Per esempio, un task che ha il compito di eseguire un posizionamento di un asse deve aspettare che l'asse arrivi alla quota target prima di continuare con altre operazioni. Nel frattempo, quindi, la CPU potrà eseguire altre istruzioni prgrammate in altre unit.
Al momento dell'accensione l'ordine con cui vengono eseguiti le unit presenti è quello dichiarato nella finestra UNIT MANAGER in QVIEW solo che il primo task eseguito è il secondo in lista poi si procede fino all'ultimo e quindi si esegue per ultimo il primo in lista.

Figura 1: esecuzione task (condizione iniziale).

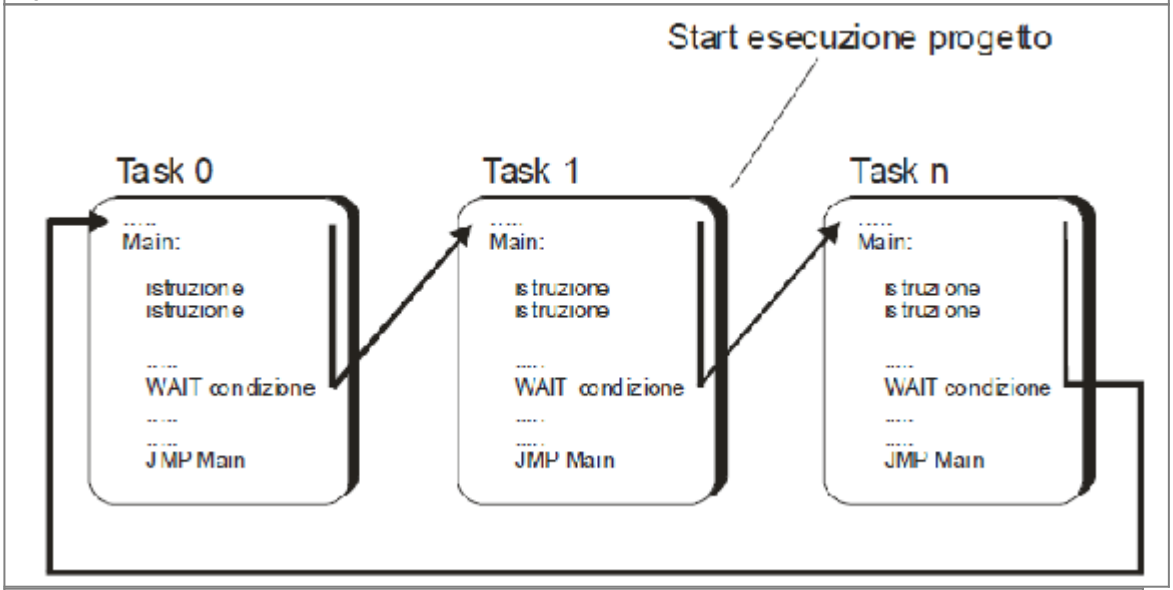
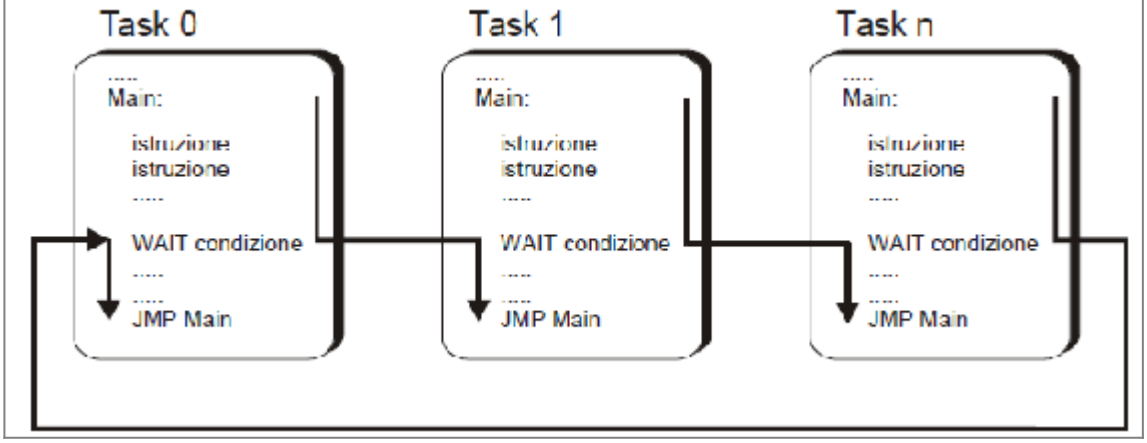


Figura 2: esecuzione task (flusso operativo).

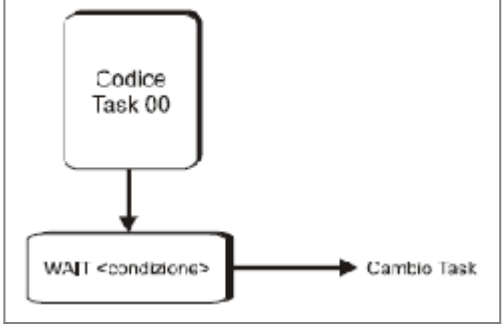


Oltre all'esecuzione del codice, la CPU si occupa di eseguire contemporaneamente i device. La trattazione dei device è rimandata al relativo capitolo; in breve, sono degli strumenti che forniscono delle funzionalità, richiamabili dal codice QCL o Ladder, per realizzare le operazioni più comuni che si incontrano nel campo dell'automazione industriale.

11.1 Struttura standard di un task QCL

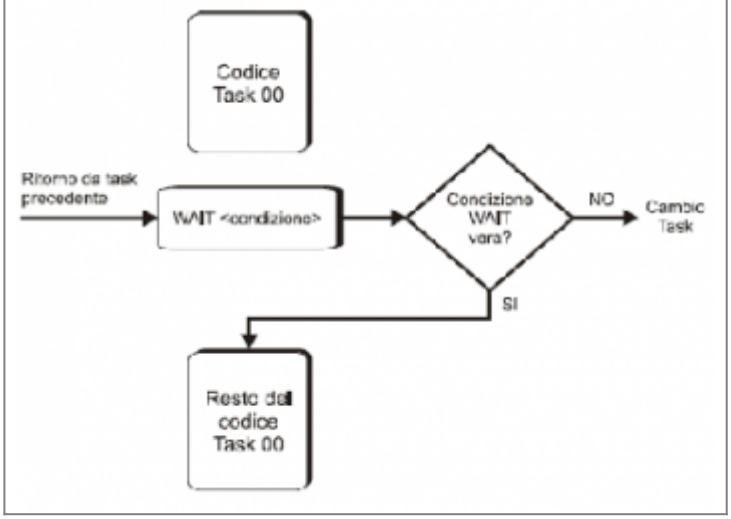
Ogni task inizia la sua esecuzione dalla prima riga di codice. Il controllo poi passa al task successivo quando si incontra nel codice un'istruzione WAIT (Figura 3).

Figura 3: flusso del programma (condizioni iniziali).



Quando il controllo ritorna al task in questione esso riprende l'esecuzione del codice dal punto in cui l'aveva interrotta solo se la condizione posta subito a destra della parola chiave WAIT è vera altrimenti esegue nuovamente un cambio task, come è schematizzato nella figura 4.

Figura 4: flusso del programma (condizione di cambio task).



Può succedere quindi che se in un task esiste un WAIT la cui condizione non è mai verificata il codice successivo a tale WAIT non venga mai eseguito.
In ogni unità il codice QCL viene eseguito fino alla fine. Per fare in modo che questo codice venga ripetuto continuamente si deve inserire un salto incondizionato dalla fine all'inizio del modulo. Ogni task, normalmente, ha una dotazione minima di codice QCL del tipo:

```
[< declaration code >
BEGIN]
[< initialization code >]
MAIN:
  < operative code >
  WAIT 1
  JUMP MAIN
END
```

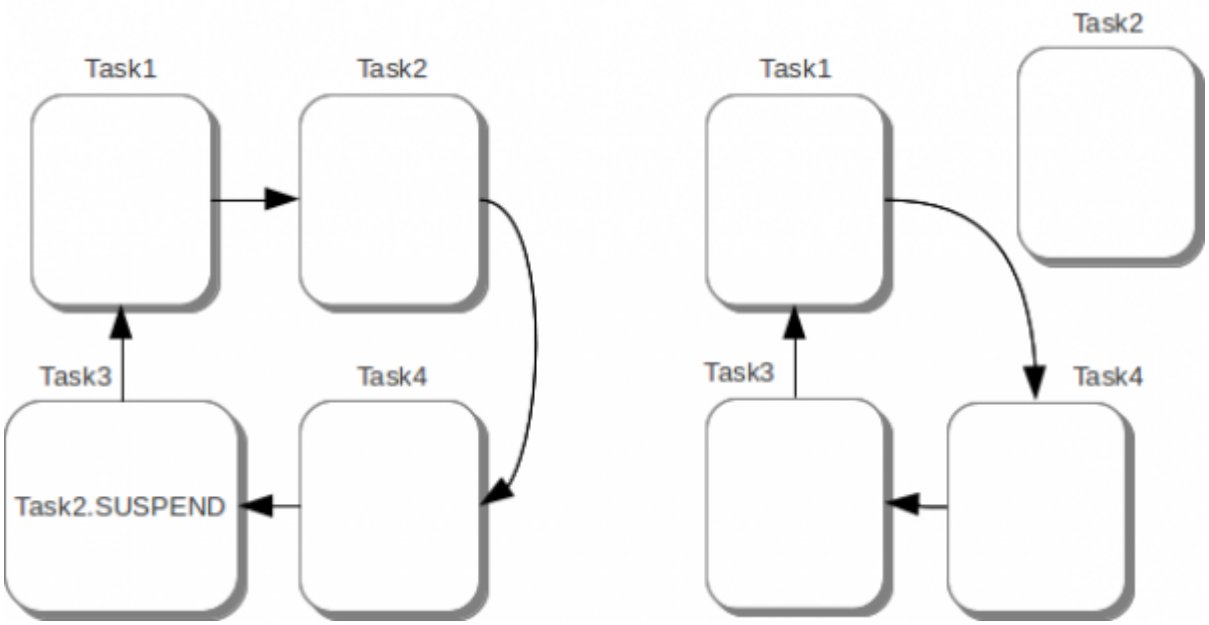
Il codice di inizializzazione presente prima del MAIN viene eseguito una sola volta in fase di accensione, mentre tutto quello che sta tra MAIN e JUMP MAIN viene eseguito ciclicamente. Se in un qualsiasi task non fosse presente l'istruzione JUMP MAIN, il task arriverebbe alla fine - END - paralizzando l'intero ciclo di multitasking. La CPU evidenzia che il ciclo è bloccato perennemente in un task attraverso il segnale di *Watchdog Active* (in QVIEW **Monitor > CPU**).
L'istruzione WAIT 1 è necessaria per assicurare il cambio di task.

11.1.1 Esempio di SUSPEND

Se si inserisce nel task3 questa istruzione

```
Task2.SUSPEND
```

il task2 viene sospeso e non verrà più eseguito.

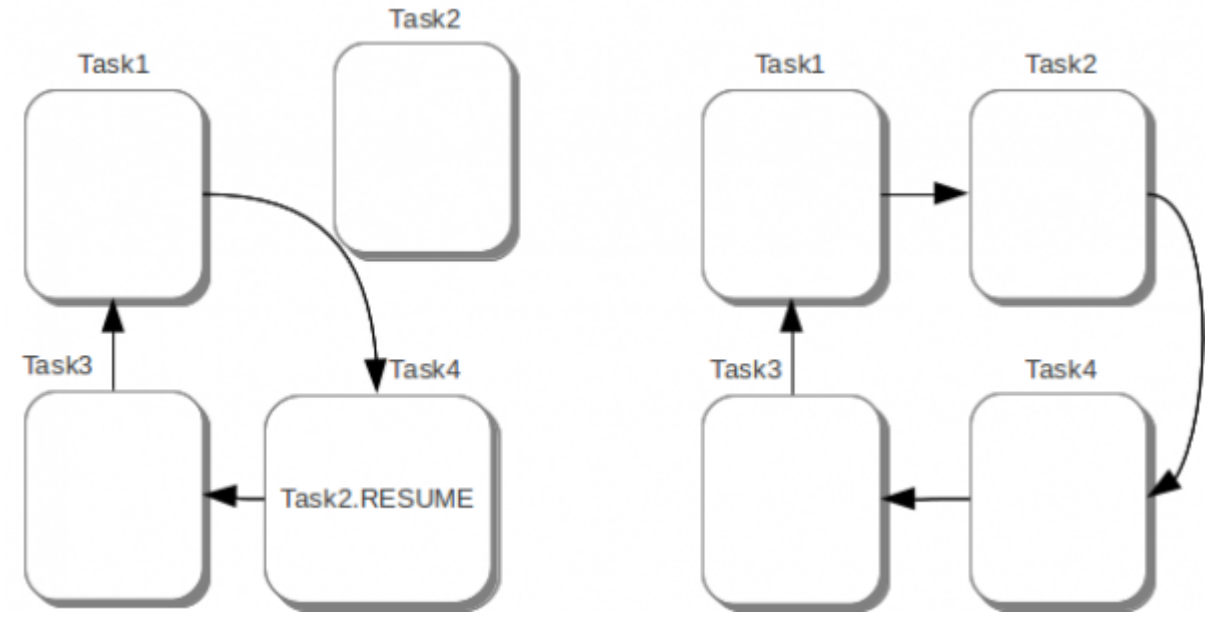


11.1.2 Esempio di RESUME

Se, per esempio, nel task4 la CPU esegue l'istruzione

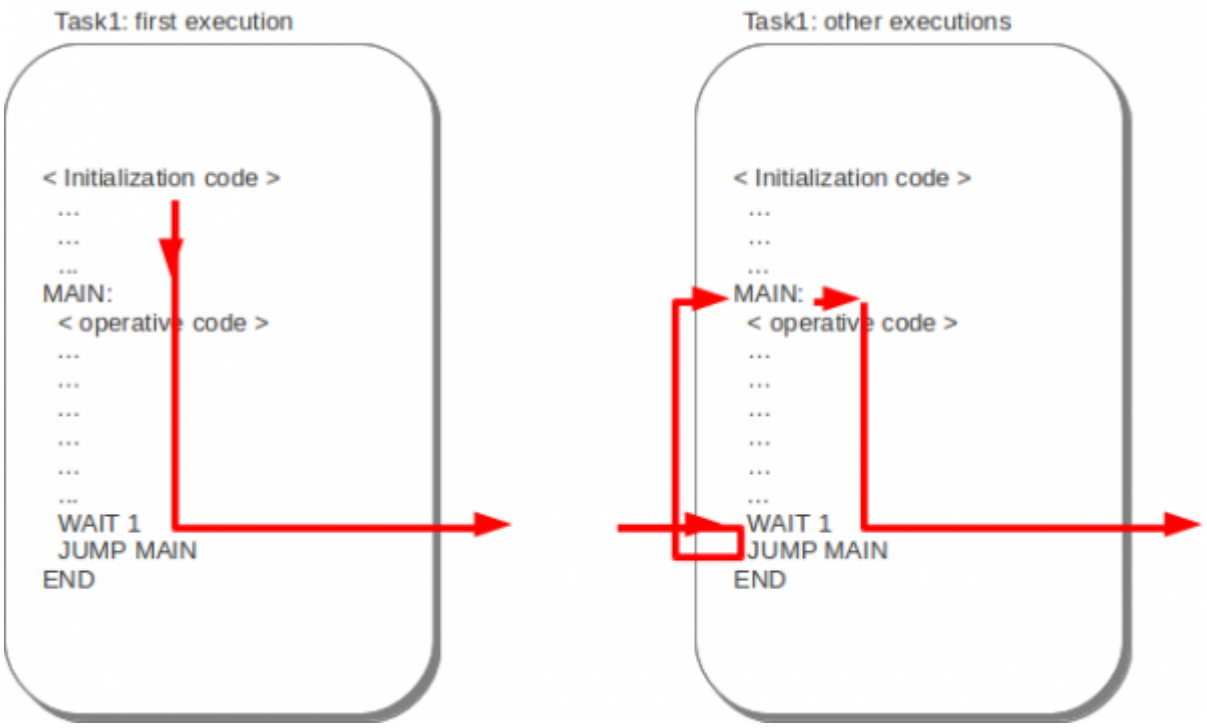
```
task2.RESUME
```

la unit task2 viene riammessa nel ciclo di multitasking e il codice contenuto in essa riprende a funzionare.



11.1.3 Esempio di RESTART

L'istruzione RESTART applicata ad una unit fa ripartire l'esecuzione del codice dalla prima istruzione della unit stessa. Per esempio l'ordine di esecuzione delle istruzioni contenute in un task sono schematizzate nella figura seguente:



Se nello stesso task o in un altro presente nel progetto la CPU esegue l'istruzione

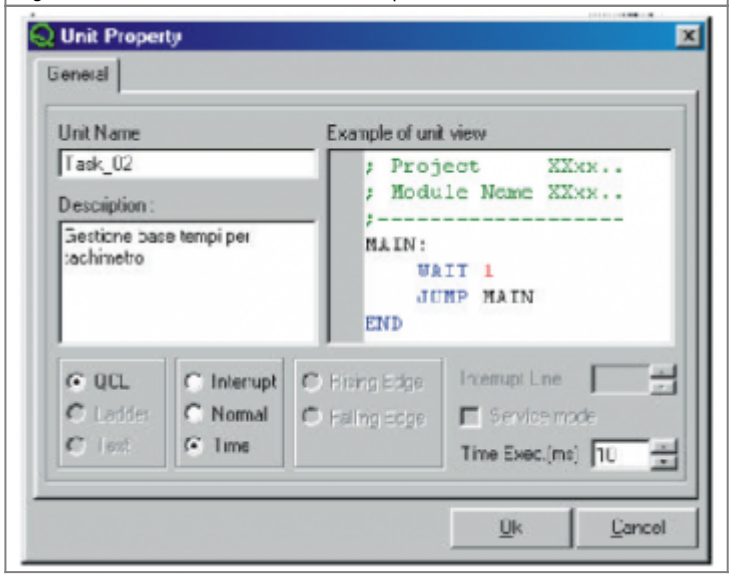
```
task1.RESTART
```

la prossima esecuzione del codice contenuto nella unti task1 riprende come se fosse la prima esecuzione (first execution).

11.2 Unità a tempo

Può succedere che nella stesura di un programma si debbano gestire eventi a scadenza di tempo fissa e ripetibile. Essendo che il sistema multitasking cooperativo non ha un tempo di ciclo ripetibile, sono state inserite le unit a tempo che sono attivabili selezionando l'apposita casella nella unit property e dichiarando la base tempi da utilizzare.

Figura 1: dichiarazione di una unit a tempo

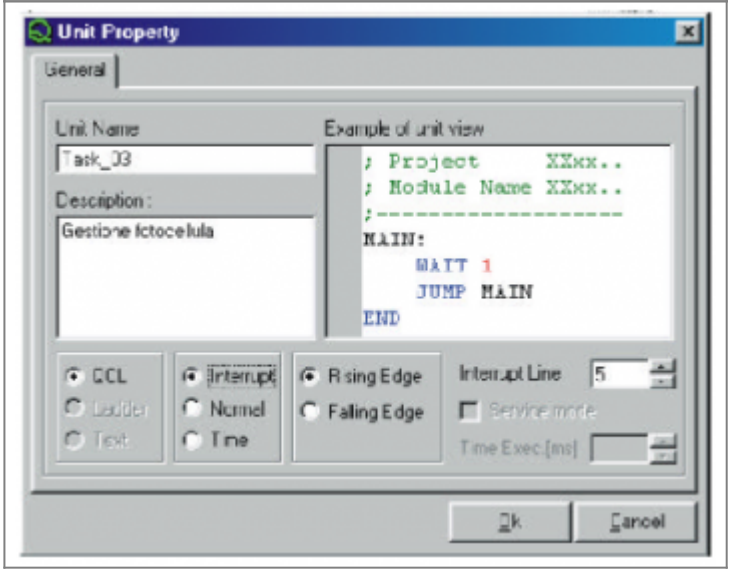


Le unit a tempo sono prioritarie rispetto alle unit normali. Si deve porre attenzione al fatto che il codice inserito in una unità a tempo (QCL o LADDER che sia) non impieghi più tempo per essere eseguito del tempo di ripetizione impostato.
Nel caso questo succeda alcune ripetizioni del codice vengono "perse" (cioè non eseguite) e la CPU segnala tale evento con una stato di "Time task lost". Questo si può vedere nella finestra CPU Monitor. Altra particolarità di questo tipo di unità è che non possono essere inserite le istruzioni di WAIT, per cui il codice contenuto deve essere eseguito in un solo ciclo.
Ci possono essere al massimo 7 unità a tempo per ogni progetto.

11.3 Unità in interruzione

Può succedere che nella stesura di un programma si debbano gestire eventi provenienti da un ingresso in interruzione (tipicamente fotocellule o sensori di prossimità). Per assolvere a questa funzione sono state inserite le unità in interrupt che sono attivabili selezionando l'apposita casella nella unit property e dichiarando la linea di interruput ed il fronte di attivazione.

Figura 2: dichiarazione di un task in interruzione



Le unità in interruzione sono prioritarie rispetto alle unità normali e rispetto alle unità a tempo, e non possono essere inserite le istruzioni di WAIT, per cui il codice contenuto deve essere eseguito in un solo ciclo. Per non sovraccaricare la CPU con le interruzioni è stato inserito il controllo che se prima non si è completata l'esecuzione del codice di interruzione, il sistema non abilita la linea di interrupt, per cui in nessun caso si possono accodare più interruzioni provenienti dalla stessa linea.
Le unità in interrupt possono essere scritti sia in QCL che in Ladder.
Ci possono essere al massimo 3 unità in interruzione per progetto.

12. Device

Il termine “device” identifica una categoria di dispositivi software atti a svolgere attività di supporto o controllo (più o meno complesse), semplificando operazioni e procedure proprie dell’automazione industriale.
Per esempio, un device può gestire un posizionatore di tipo CNC con uscita analogica +/-10V, rendendo disponibili tutte le funzioni, i comandi ed i parametri necessari alla corretta gestione dei posizionamenti (ricerca di home, comandi per la movimentazione manuale, parametri di velocità, tempo di inversione, ...).
I device dispongono di proprie funzioni, variabili, comandi e parametri; l’utente ha la possibilità di configurarli ed inserirli all’interno del proprio progetto, in modo che uno o più device diventino parte integrante dell’applicativo sviluppato.

I parametri sono delle variabili che permettono di configurare il funzionamento del device (ad esempio la velocità di posizionamento, il conteggio, ecc.):

```
<device name>.<parameter device>
```

I comandi fanno eseguire al device determinate funzioni (ad esempio START, STOP, ecc.):

```
<device name>.<command name>
```

La lista dei device interni disponibili in un certo modello di QMOVE è inserita nei manuali d’installazione e manutenzione relativi alla CPU utilizzata.

12.1 Dichiarazione dei Device

La sintassi per la dichiarazione dei device viene trattata approfonditamente nella documentazione dei device.

L’utilizzo di un device deve essere dichiarato all’interno del file di configurazione oppure nella parte riservata alle dichiarazioni di ogni unit. La sintassi della dichiarazione varia a seconda del device (interno o esterno).

12.1.1 Dichiarazione device

Per la dichiarazione del device interno, nel file di configurazione deve essere inserita la parola chiave INTDEVICE.
Per ulteriori informazioni fare riferimento alla documentazione relativa ai device interni.

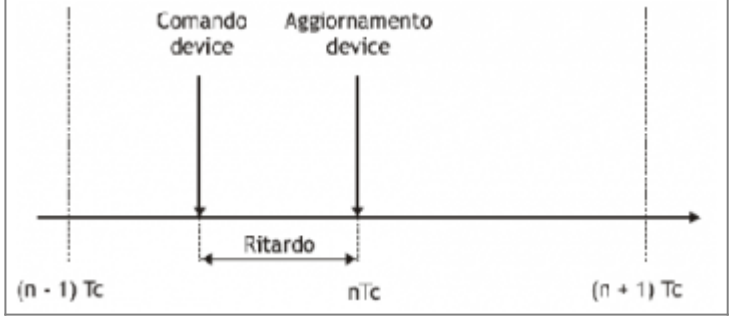
12.2 Utilizzo dei devices

In questo capitolo si daranno delle informazioni preliminari per un corretto utilizzo dei devices interni. È indispensabile comunque completare ciò che segue con la documentazione specifica fornita per ognuno dei devices.

12.2.1 Tempo di campionamento

Come abbiamo già detto, nel capitolo dedicato al multitasking, una delle caratteristiche peculiari dei devices è il tempo di campionamento (Tc) che stabilisce ogni quanto tempo viene eseguita la gestione del device da parte della CPU.
Vediamo quali sono i criteri generici nella scelta dei tempi di campionamento. Solitamente un tempo piccolo permette al device di venire eseguito molto frequentemente e quindi di poter reagire velocemente alle azioni esterne (es.: miglior regolazione degli assi). Durante l’esecuzione del codice QCL un accesso in scrittura al device o un comando viene processato dopo un tempo massimo pari al tempo di campionamento. Questo fa sì che più piccolo è Tc, minore è il ritardo di esecuzione del device. Nel grafico di figura 1 si dà una esemplificazione di quanto detto.

Figura 1: ritardo aggiornamento device.

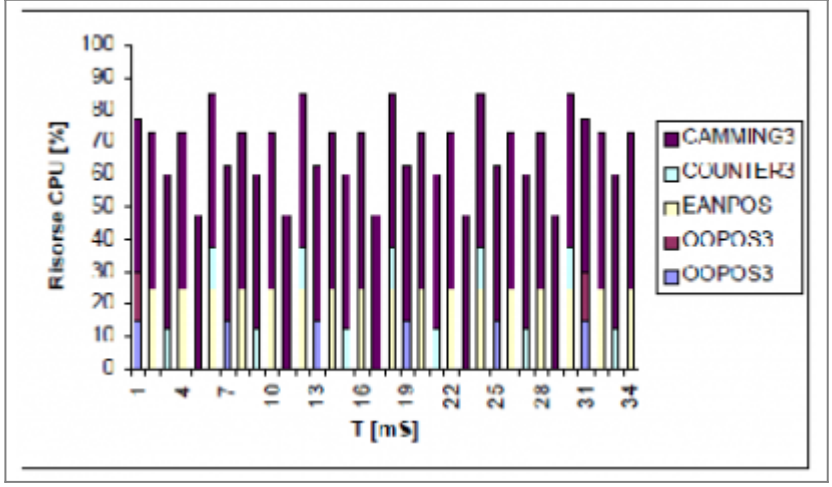


La scelta dei tempi di campionamento per i device utilizzati nel progetto che si sta realizzando deve essere fatta con le opportune attenzioni. Una domanda che il lettore si sarà posto è la seguente: “Perchè non posso scegliere come tempi di campionamento il valore minimo consentito per ognuno dei device?”. In effetti in tal modo le prestazioni sarebbero le massime. Questa scelta però non è sempre possibile.

La CPU riserva una parte della sua risorsa di calcolo per la gestione dei devices utilizzati nel progetto. Ogni device ad ogni campionamento impegna una parte di questa risorsa. Per determinare e stimare l'utilizzo della risorsa consideriamo che la CPU ad ogni millisecondo mette a disposizione una risorsa pari a 100. Nella documentazione relativa ai firmware si puo' trovare la percentuale di risorse che ogni device utilizza in un istante di campionamento. La CPU può gestire nello stesso campionamento più devices e la risorsa occupata complessivamente è la somma delle percentuali relative ad ogni devices. Nella scelta dei tempi bisogna evitare che in qualche campionamento la risorsa complessiva superi il valore 100%. Automaticamente la CPU sfasa l'esecuzione dei device per evitare di superare il limite massimo della risorsa di calcolo disponibile.
Il grafico di figura 2 dà una esemplificazione di quanto detto. Nell'esempio sono stati installati:

- un device OOPOS3 con tc=6msec.,
- un OOPOS3 con tc=30msec.,
- un device EANPOS con tc=2msec.
- un COUNTER3 con tc=3msec ed un
- CAMMING3 con tc=1msec.

Figura 2: impegno risorse CPU per gestione device interni.



12.2.2 Comandi consecutivi e loro priorità

Un comando ad un device non viene elaborato subito dalla CPU ma al successivo tempo di campionamento, senza per questo che la sequenza delle successive istruzioni QCL venga interrotta. Per questo motivo il device potrà trovarsi a processare nello stesso istante di campionamento più comandi, ed in questo caso la gestione del device non terrà conto della sequenza con cui tali comandi sono stati dati ma li processerà secondo un ordine interno di priorità. Tale priorita è specificata nella documentazione relativa ad ogni device.
Quando si vuole assicurare la sequenza con cui i comandi sono stati inseriti nella stesura del codice QCL, bisogna porre delle istruzioni di WAIT e come condizione uno stato del device. In tal modo l'istruzione WAIT attende l'esecuzione del comando prima di eseguire l'istruzione successiva.

12.2.3 Comandi complementari

Esistono dei comandi che sono tra loro complementari e cioè che producono ognuno l'effetto inverso dell'altro. Se ad un campionamento il device si trova ad eseguire alcuni comandi fra loro complementari, risulta avere effetto solamente l'ultimo della sequenza. Anche in questo caso per assicurare l'esecuzione dei due comandi fare riferimento al paragrafo "Comandi consecutivi e loro proprietà".

12.3 Gruppi di device

Nella sezione INTDEVICE è possibile definire un gruppo di device attraverso le parole chiavi *DEVGROUP* e *ENDDEVGROUP*. Esempio:

```
DEVGROUP Assi
  X ANPOS2 4 3.CNT01 1 3.INP19 X.X 3.AN01
  Y ANPOS2 4 3.CNT02 2 3.INP18 X.X 3.AN02
ENDDEVGROUP
```

I device all'interno di un gruppo sono sottoposti a due vincoli:

1. Il loro tempo di campionamento deve essere lo stesso;
2. La somma dei loro tempi di esecuzione non deve superare il massimo tempo di esecuzione in un campionamento.

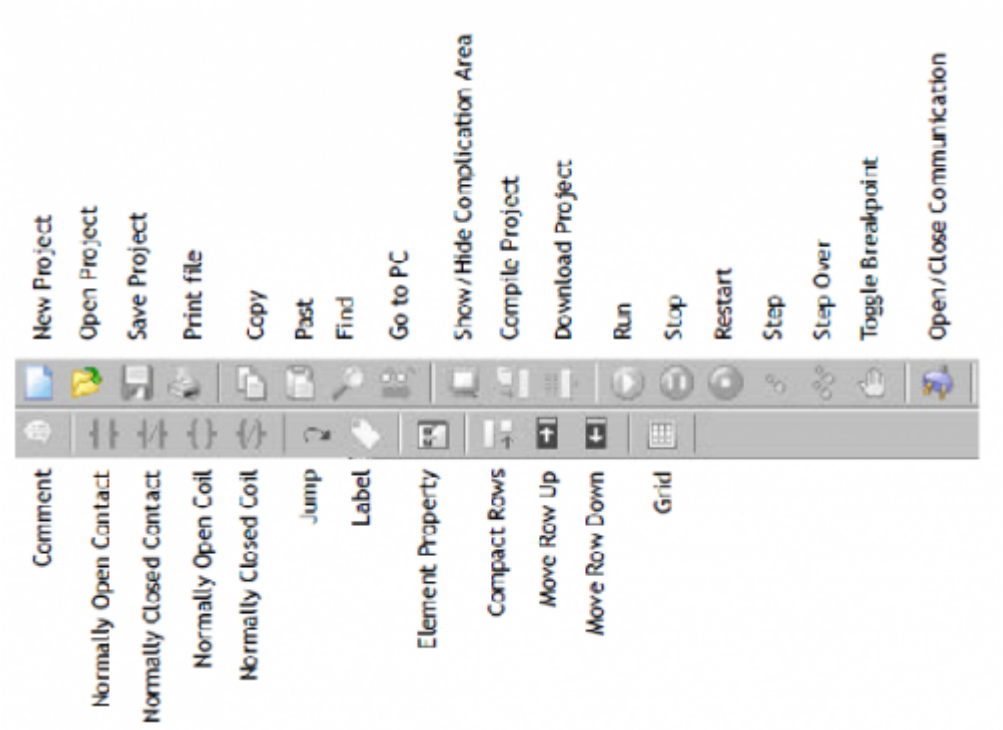
La definizione di un gruppo permette di garantire l'esecuzione di tutti i device che ne fanno parte nello stesso istante di campionamento. E' una funzionalità riservata per particolari applicazioni di motion control.

13. Ambiente Qview

In questo capitolo verranno descritti i vari menu e comandi di QVIEW. Per l'approfondimento di alcune funzionalità proprie dell'ambiente Windows (Apri, Salva, ...) si rimanda alla documentazione del sistema operativo.

13.1 Barra degli strumenti

La barra degli strumenti è composta da una serie di icone che ripropongono le funzionalità dei comandi principali (o di uso frequente).



13.2 Barra di stato

La barra di stato è composta da quattro sezioni distinte.
La prima sezione a sinistra è dedicata alla visualizzazione del valore della variabile selezionata. E' sufficiente cliccare sul tasto destro di una variabile o di un parametro di un device per leggere il suo valore in questa sezione (naturalmente bisogna che il collegamento seriale sia attivo). La seconda sezione da sinistra visualizza la posizione del cursore (riga e colonna) e la modalità di scrittura (INS = inserimento di testo e OVR = sovrascrittura testo) (Figura 1).
La terza sezione da sinistra visualizza lo stato della porta seriale di comunicazione tra PC e CPU: stato della porta (*No connection* o *Connected with:...*), protocollo di comunicazione in uso e velocità di trasmissione (Figura 2).
La quarta sezione più a destra visualizza i messaggi "Match OK" o "No Match" quando la comunicazione seriale e attiva e il progetto aperto è lo stesso di quello presente nella CPU oppure e diverso, rispettivamente.

Figura 1: seconda sezione da sinistra: posizione cursore.

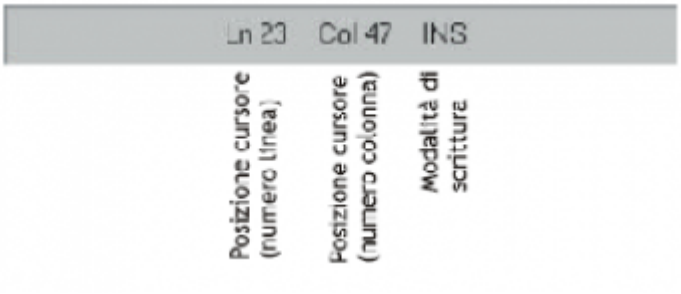
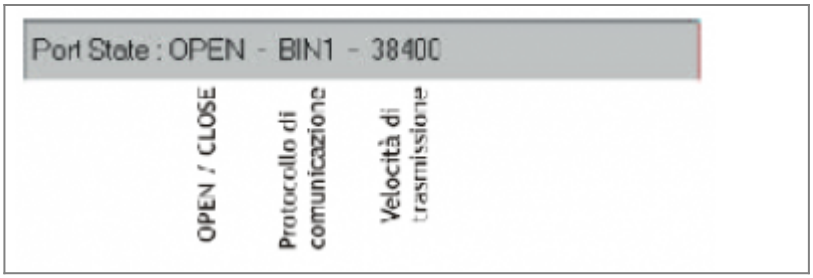


Figura 2: terza sezione da sinistra: stato e parametri porta di comunicazione seriale.



13.3 Menu File

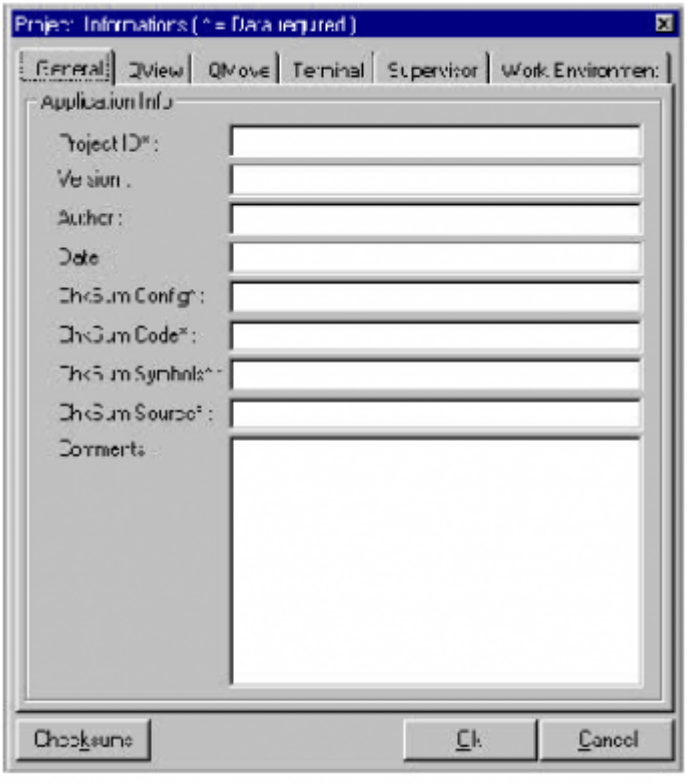
Contiene i comandi per la gestione dei progetti e dei files che li compongono.

13.3.1 New Project

Permette di creare un nuovo progetto. Inizialmente viene richiesto il nome da assegnare al progetto (Figura 1).
Viene quindi aperta automaticamente la finestra “Project Information” (Figura 2) per inserire i dati del progetto che si intende realizzare. Questa tabella ha lo scopo di raccogliere una serie di informazioni, essa può essere compilata anche in un secondo momento (in ogni caso non è obbligatorio compilarla).

Figura 1: inserimento del nome del progetto.

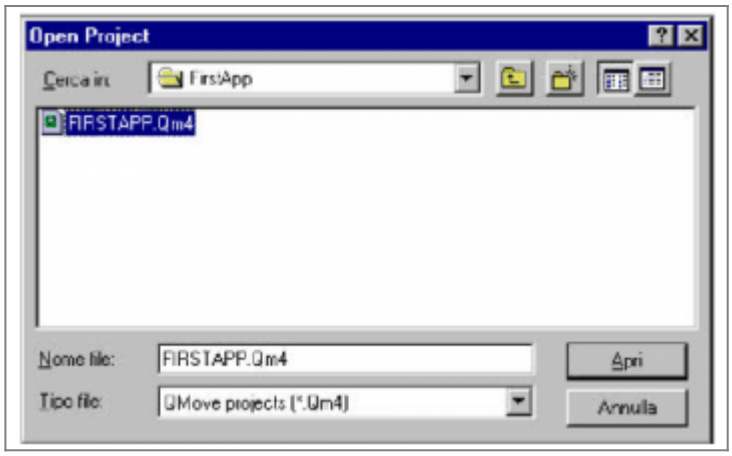
Figura 2: finestra con le informazioni di progetto.



13.3.2 Open Project

Permette di aprire un progetto esistente (Figura 3).

Figura 3: apertura di un progetto.



E’ possibile aprire i progetti realizzati con versioni precedenti di Qview selezionando le diverse estensioni da “Tipo file”.

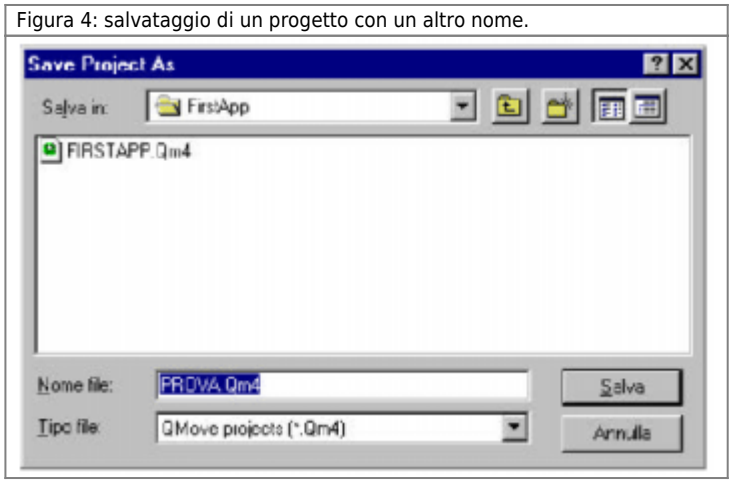
13.3.3 Save Project

Permette di salvare le modifiche apportate al progetto in uso.

13.3.4 Save Project As

Permette di salvare una copia del progetto in uso (comprese le eventuali modifiche apportate) (Figura 4).

Figura 4: salvataggio di un progetto con un altro nome.



13.3.5 Close Project

Permette di chiudere il progetto in uso.

13.3.6 Add Unit

Permette di aggiungere una nuova unità al progetto in uso.
Le nuove unità possono essere:

- Configuration Unit: inserisce la unit di configurazione. Non è abilitato se nel progetto è già presente una unit di configurazione.
- QCL Unit: inserisce una nuova unit (task) in linguaggio QCL
- Ladder Unit: inserisce una nuova unit (task) in linguaggio Ladder
- Document Unit: inserisce una nuova unit document (documento di testo per la raccolta di commenti, specifiche, note ...).

Questa nuova unit verrà aggiunta in coda alla lista delle unit già presenti.

13.3.7 Insert Unit

Permette di inserire una nuova unit al progetto in uso.

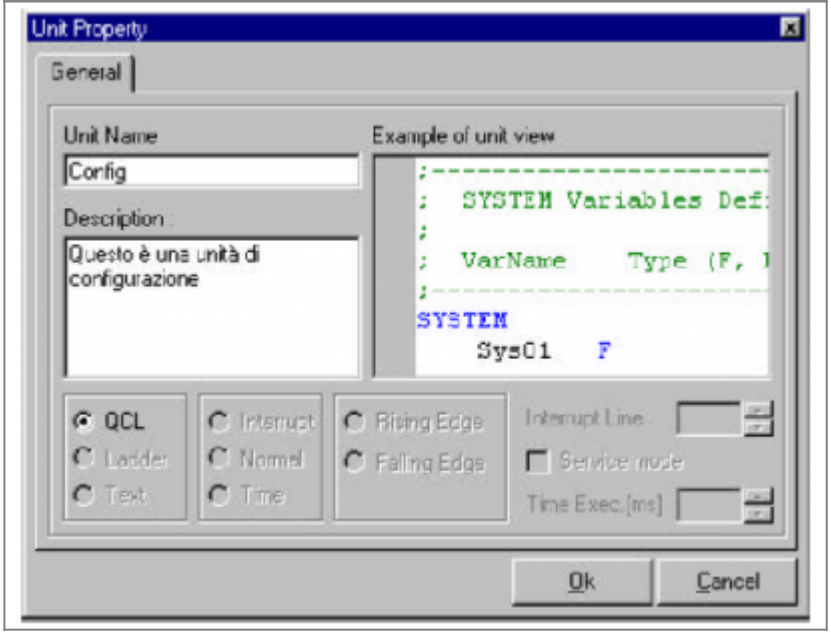
Le nuove unità possono essere le stesse elencate per il comando *Add Unit*.

Questa nuova unit verrà inserita nella lista appena sopra alla unit selezionata.

13.3.8 Creazione di una unit di configurazione

Ogni volta che viene creato una nuova unit di configurazione con i comandi “Add unit” o “Insert unit”, appare la finestra “Unit property” (Figura 5).

Figura 5: intestazione dell’unit di configurazione.



In questa finestra è possibile assegnare un nome all’unit e una breve descrizione della unit. Una volta confermato con ***Ok***, la unit di configurazione viene aggiunta alla lista delle unit e viene aperta la finestra di editor per iniziare a modificarla.
In un progetto deve esistere una sola unit di configurazione.

13.3.9 Creazione di una unit QCL o LADDER

Ogni volta che viene creato una nuova unit QCL o LADDER con i comandi “Add unit” o “Insert unit”, appare la finestra “Unit property” (Figura 6, 7).

Figura 6: intestazione dell unità QCL.

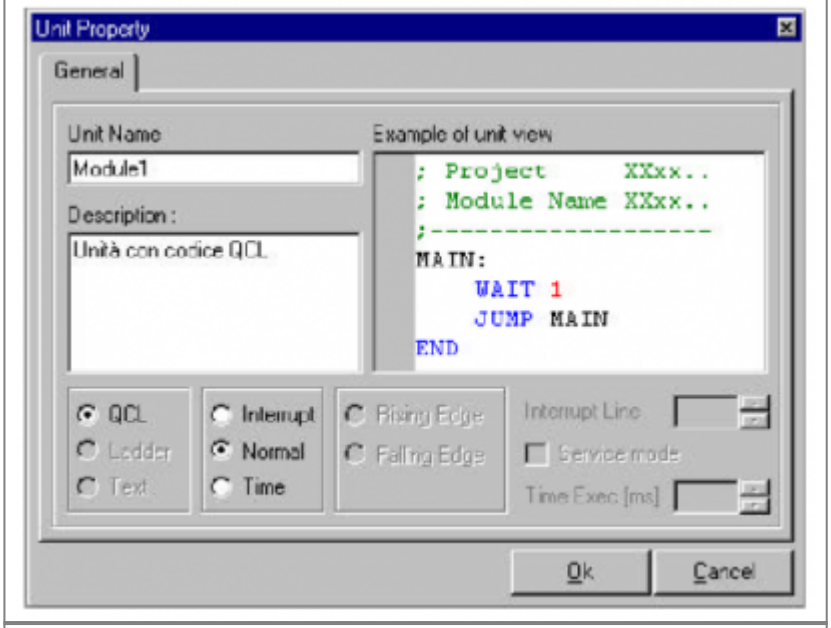
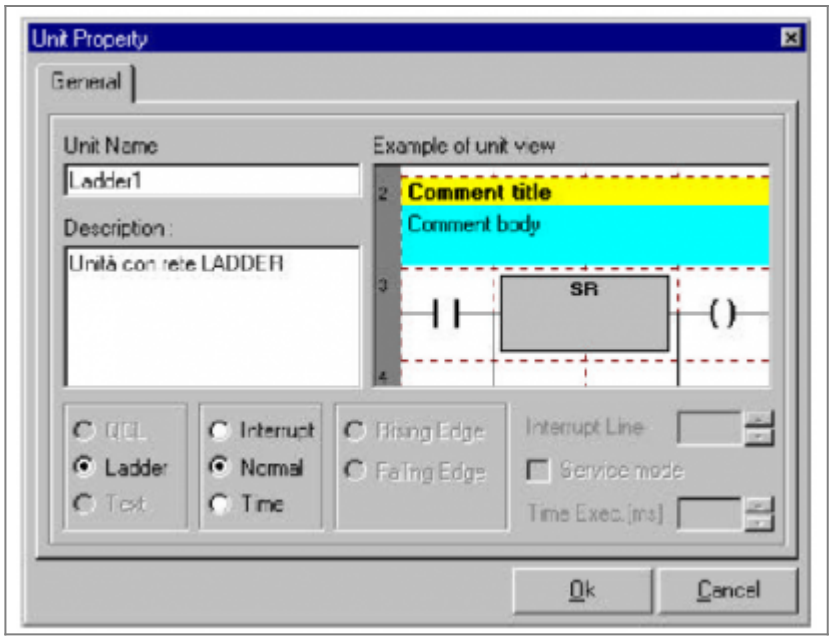


Figura 7: intestazione delle unit LADDER.



In questa finestra è possibile assegnare un nome alla unit e una breve descrizione della unit.
Una volta confermato con ***Ok***, la unit QCL o LADDER viene aggiunta alla lista delle unit e viene aperta la finestra di editor per iniziare a modificarla.
Nelle finestre “Unit property” è possibile specificare se la unit (QCL o LADDER) è una unit

- **normale**;
- **in interrupt**: si deve specificare il fronte (**Rising Edge**: fronte di salita, **Falling Edge**: fronte di discesa) dell’impulso che avvia l’interrupt e la linea di interrupt utilizzata (**Interrupt Line**);
- **a tempo**: si deve specificare ogni quanto tempo si deve ripetere il codice inserito nell’unità (**Time Exec.**).

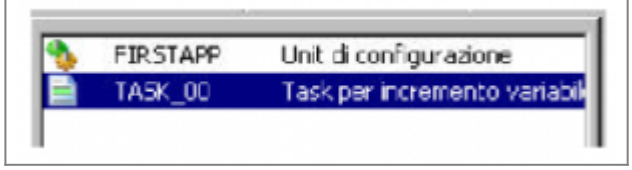
13.3.10 Creazione di un unit di documentazione

Le unit di documentazione sono utilizzate dal programmatore per scrivere appunti, note, particolari di funzionamento. E’ possibile utilizzare queste unit solamente in modalità di testo.

13.3.11 Remove Unit

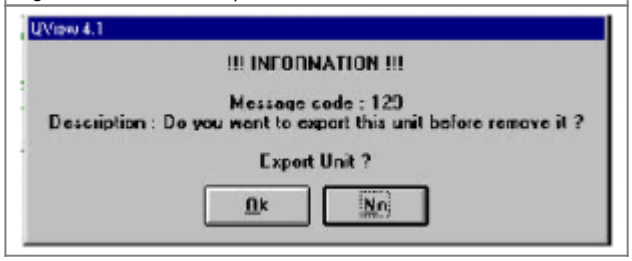
Permette di rimuovere la unit selezionata.

Figura 8: selezione della unit da rimuovere.



Prima dell’eliminazione della unit, il programma chiede se si vuole esportare la unit prima di rimuoverla; in caso contrario, la unit verrà persa.

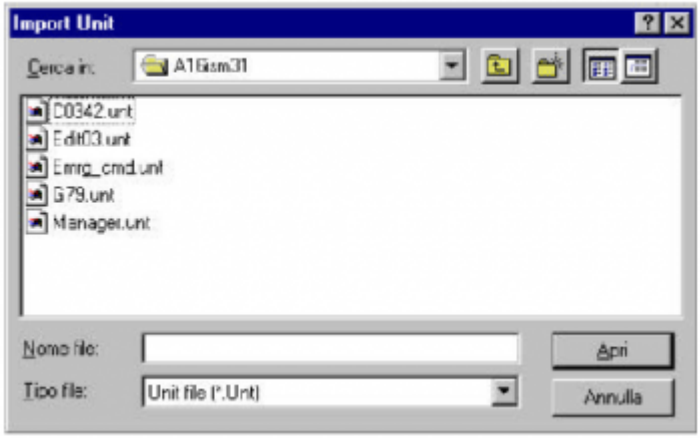
Figura 9: richiesta di esportazione unit.



13.3.12 Import Unit

Permette di importare una unit precedentemente esportata anche da un altro progetto (Figura 10).

Figura 10: Importazione di una unit.



E' possibile selezionare l'alternativa "Add unit..." o "Insert unit..." per importare la unit in coda alla lista o sopra la unit selezionata in quel momento.

13.3.13 Export Unit

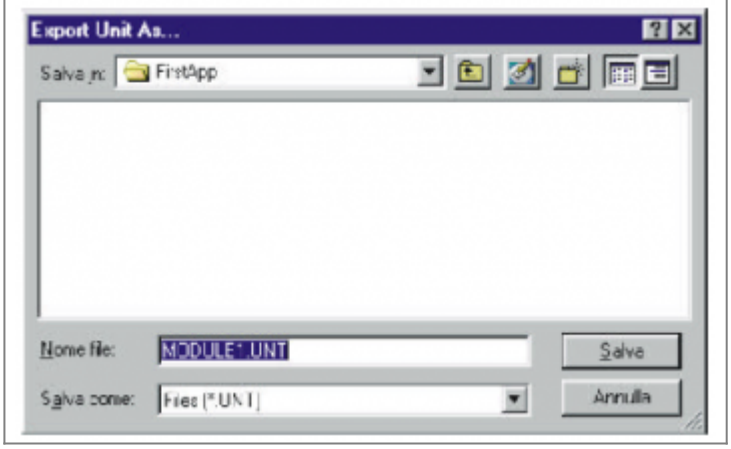
Permette di esportare una copia della unit di progetto selezionata, per poterla trasferire ad un altro progetto. La copia originale rimarrà nel progetto stesso. La unit verrà esportata nella stessa directory del progetto aperto. La unit verrà esportata come un file con estensione unt.

Figura 11: Conferma di esportazione.

13.3.14 Export Unit As

Permette di esportare una copia dell unit di progetto selezionata rinominando la unit. La copia originale rimarrà nel progetto stesso.

Figura 12: richiesta di esportazione unit con rinomina del nome.



13.3.15 Unit Property

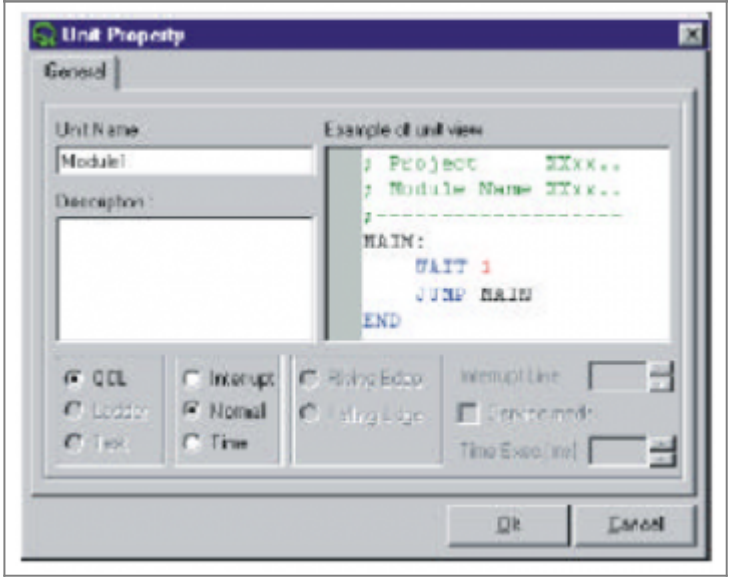
Permette di modificare le proprietà della unit selezionata. In particolare è possibile impostare il comportamento Runtime della unit in questione, cioè la modalità di esecuzione della unit una volta che verrà trasferita nella CPU del Qmove. Le unit possono avere i seguenti comportamenti:

Normal Fino al limite massimo consentito (totale massimo di unit: 65535).
Conferisce alla unit un comportamento Runtime ciclico (vedere il capitolo relativo al multitasking).

Interrupt Massimo tre unit per progetto.
Conferisce alla unit un comportamento Runtime determinato da eventi esterni all'apparecchiatura (linee di interrupt hardware). Si puó determinare se l'esecuzione della unit sarà attivata sul fronte di salita o discesa del segnale entrante ed il numero della linea hardware ad essa collegata, impostabile da 1 a 8.

Time Massimo sette unit per progetto.
Conferisce alla unit un comportamento Runtime determinato dallo scadere di un tempo prefissato, interno all'apparecchiatura. Il predetto tempo è regolabile in un range da 1 a 999 millisecondi. Nell' inserimento di una nuova unit a tempo, il tempo proposto è di 100ms.

Figura 13: modifica delle proprietà di una unit.



Aprendo la finestra “Unit property” (Figura 13) è possibile rinominare un unità precedentemente introdotta cambiando il suo Unit Name.

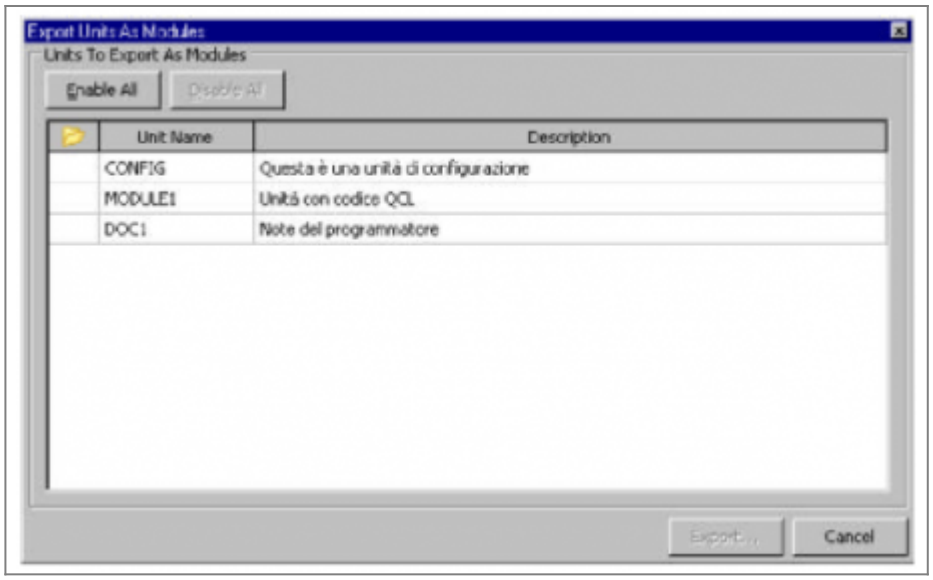
13.3.16 Import Module As Unit

Permette di importare una unit realizzata con le vecchie versioni di Qview (*.mod). E’ possibile aggiungere la unit in coda alla lista (Add Module as unit) oppure inserirla in una posizione intermedia sopra a quella selezionata (Insert Module as unit).

13.3.17 Export Unit as Module

Permette di esportare una o più unità in un formato compatibile con le precedenti versioni del Qview (moduli per Qview 2.x e Qview 3.x). Una volta selezionata questa funzione viene visualizzata la finestra di figura 14 dove è possibile selezionare quali unità esportare (vengono elencate solo le unità non LADDER).

Figura 14: esportazione delle unità come moduli.



Per selezionare l'unità da esportare bisogna eseguire un doppio-click sull'unità stella.

13.3.18 Export Symbols File

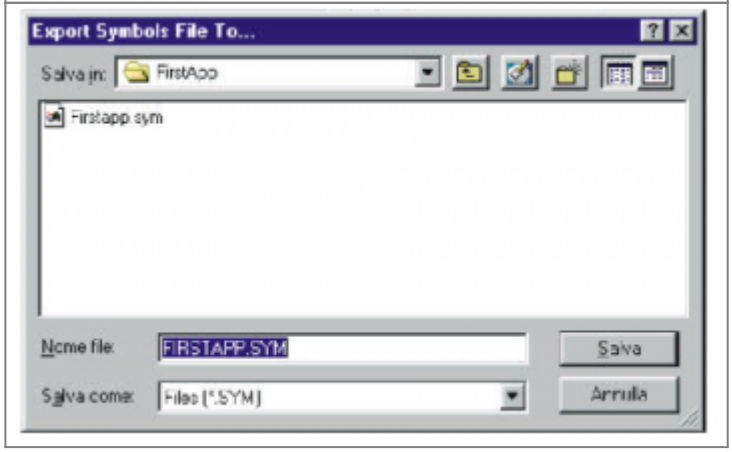
Permette di esportare il file simboli dal progetto per poter riallineare i simboli nel terminale.

Figura 15: esportazione simboli.

13.3.19 Export Symbols File As ...

Permette di esportare il file simboli dal progetto, rinominandoli, per poter riallineare i simboli nel terminale.

Figura 16: esportazione simboli con rinomina.



13.3.20 Export Binary File & Export Binary File As ...

Questa funzionalità è disponibile se la compilazione del progetto è andata a buon fine. Esso permette di esportare il file binario (risultato della compilazione) utile per poter eseguire il download dell'applicativo nella CPU senza l'utilizo della comunicazione seriale (per esempio trasferendolo alla CPU attraverso una Multi Media Card). E' possibile esportare il file binario con lo stesso nome del progetto (**Export Binary File**) oppure cambiando il suo nome (**Export Binary File As...**).

13.3.21 Project Information ...

Permette di visualizzare una finestra (Figura 20) per l'inserimento delle informazioni specifiche del progetto.

Figura 17: visualizzazione informazioni sul progetto in uso.

In questa finestra sono presenti delle cartelle che suddividono le informazioni di progetto per argomento. Alcune informazioni sono contrassegnate con un asterisco per indicare che sono obbligatorie per una completa informazione sul progetto. Se alcune di queste informazioni obbligatorie non vengono inserite, periodicamente appariranno dei messaggi che avviseranno il programmatore di questa mancanza (per disabilitare questi avvisi si veda ***Menu Options - Program Setup***). In questa finestra esiste il tasto "Checksums" che inserisce in tabella i codici checksum del progetto automaticamente.

13.3.22 Print

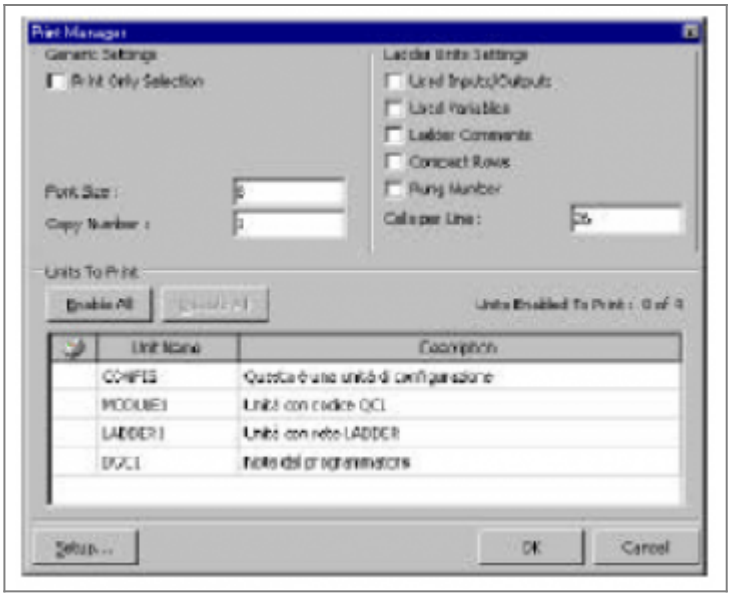
Il comando è disponibili solo selezionando una finestra di editor.

Permette di accedere ad un finestra "Print Manager" tramite la quale è possibile stampare una parte o tutto il progetto con alcune possibilità di configurazione della stampa stessa. In figura 18 viene mostrata questa finestra dove nel riquadro basso appare la lista delle unit che compongono il progetto. Selezionando queste unità con il mouse è possibile decidere quali stampare. Nel riquadro in alto a sinistra si possono impostare dei settaggi generici sulla dimensione del font e sul numero di copie da eseguire.
Nel riquadro in alto a destra è possibile configurare la stampa dei task LADDER dove si può specificare:

- stampa della lista degli ingressi e uscite utilizzati nel task LADDER;
- stampa delle variabili utilizzate nel task LADDER;
- stampa dei commenti inseriti nel task LADDER;
- stampa con compattamento delle righe LADDER eliminando gli spazi vuoti;
- stampa dei numeri dei rung;
- numero di celle per linea da stampare.

Inoltre in alto a sinistra è possibile selezionare la stampa solo della parte dell unit evidenziata.

Figura 18: selezione modalità di stampa.



13.3.23 Richiamo ultimi progetti aperti

È possibile richiamare direttamente gli ultimi 4 file aperti (Figura 19).

Figura 19: apertura progetti recenti.

13.3.24 Exit

Chiude QVIEW chiedendo il salvataggio delle eventuali modifiche apportate al progetto in uso.

13.4 Menu Edit

I comandi a seguire sono disponibili solamente quando è visualizzata un'unità con l'apposito editor.

13.4.1 New Element

Permette di inserire un nuovo elemento nella rete Ladder (deve essere visualizzata un'unità LADDER).

13.4.2 New Rung

Permette di inserire un nuovo rung nella rete Ladder (deve essere visualizzata un'unità LADDER).

13.4.3 Delete Rung

Permette di eliminare il rung selezionato nella rete Ladder (deve essere visualizzata un'unità LADDER).

13.4.4 Toggle Link

Permette di collegare in verticale due elementi ladder. Il collegamento viene fatto sempre verso il basso e nella parte sinistra della cella selezionata (deve essere visualizzata un'unità LADDER).

13.4.5 Substitute Obsolete Element & Substitute All Obsolete Elements...

Permette di sostituire l'elemento o tutti gli elementi obsoleti. Per una descrizione più completa si veda il capitolo "Editor LADDER - Elementi LADDER obsoleti" (deve essere visualizzata un'unità LADDER).

13.4.6 Element Properties...

Permette di modificare le variabili utilizzate all'interno dell'elemento LADDER selezionato (deve essere visualizzata un'unità LADDER).

13.4.7 Undo

Permette di cancellare l'ultima modifica fatta.

13.4.8 Redo

Permette di annullare il comando di Undo e reinserire la modifica eliminata.

13.4.9 Cut - Copy - Paste - Delete

Per tagliare, copiare, incollare ed eliminare del testo o degli elementi.

13.4.10 Select All

Permette di selezionare tutto il testo o gli elementi contenuti in una finestra di editor.

13.4.11 Compact Rows

Permette di compattare le righe di codice Ladder nel minor numero di celle possibili eliminando le righe vuote (deve essere visualizzata un'unità LADDER).

13.4.12 Move Rows Up

Permette di spostare in alto una riga di codice ladder (deve essere visualizzata un'unità LADDER).

13.4.13 Move Rows Down

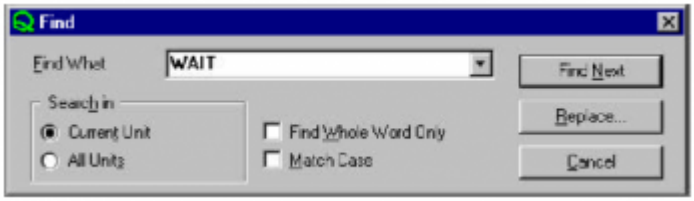
Permette di muovere in basso una linea di codice Ladder (deve essere visualizzata un'unità LADDER).

13.4.14 Find

Ricerca la parola inserita in funzione dei criteri di ricerca selezionati (Figura 20).
Criteri di ricerca

- Search in Current Unit: ricerca solo nell'unità selezionata.
- Search in All Units: ricerca in tutte le unità del progetto.
- Find Whole Word Only: cerca solo parole intere
- Match Case: cerca parole come digitato, rispettando maiuscole e minuscole.

Figura 20: finestra inserimento criteri di ricerca.



13.4.15 Find Next

Una volta trovato un termine (comando Find), permette di continuare la ricerca dello stesso termine sul resto del documento (o sulle altre unità), mantenendo inalterati i criteri di ricerca.

13.4.16 Replace

Ricerca la parola inserita (Find What) con i criteri di ricerca impostati e la sostituisce con la nuova parola (Replace With)(Figura 21).

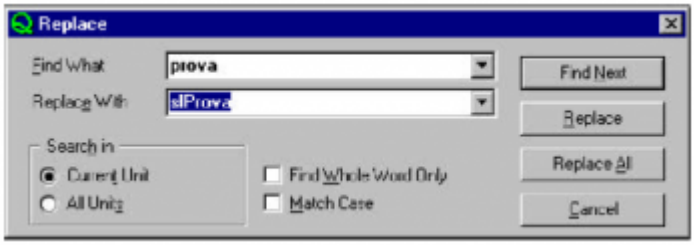
Criteri di ricerca

- Search in Current Unit: ricerca solo nella unit selezionata;
- Search in All Units: ricerca in tutti le unit del progetto;
- Find Whole Word Only: cerca solo parole intere;
- Match Case: cerca parole come digitato, rispettando maiuscole e minuscole.

Criteri di sostituzione

- Find Next: non esegue nessuna sostituzione ma inizia la ricerca della parola (Find What) nel resto del progetto;
- Replace: sostituisce la parola cercata (Find What) con la nuova parola (Replace With);
- Replace All: sostituisce automaticamente tutte le parole (Find What) dell'intero progetto.

Figura 21: finestra inserimento criteri per ricerca e sostituzione.



13.4.17 Go to ...

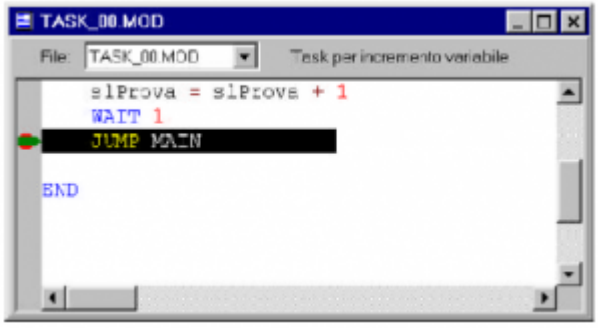
Sposta il cursore al numero di riga inserito (Figura 22).

Figura 22: inserimento numero di riga per spostamento cursore.

13.4.18 Go to PC

Dal momento in cui l'esecuzione del progetto viene interrotta (per esempio per l'inserimento di un breakpoint), questo comando visualizza la riga di programma alla quale si è verificata l'interruzione. In figura 23, il comando Go to PC visualizza l'interruzione per l'inserimento di un breakpoint.

Figura 23: esecuzione programma interrotta da breakpoint.



13.4.19 Next Unit / Previous Unit

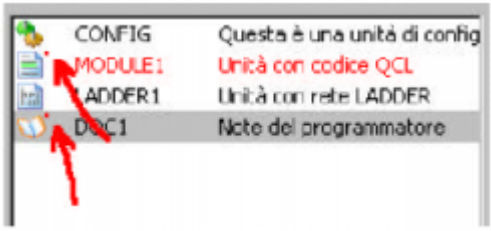
Permette di spostarsi tra le unit del progetto.

13.4.20 Next Selected Unit / Previous Selected Unit

Quando fosse necessario spostarsi tra alcune unit del progetto ignorandone alcuni altri, selezionare uno per volta le unit da

visualizzare e premere la barra spaziatrice. A fianco delle unit selezionate compare un punto rosso (Figura 24).
Con i comandi Next e Previous Selected Unit è possibile scorrere in avanti ed indietro solamente tra le unit selezionate.
Per rimuovere la marcatura, selezionare la unit e ripremere la barra spaziatrice; il punto rosso scompare.

Figura 24: selezione unit da analizzare.



13.5 Menu: Project

Contiene i comandi la gestione dei progetti e dei dati contenuti nella CPU.

13.5.1 Compile

Converte il progetto in uso in un formato interpretabile dalla CPU. Il progetto potrà essere scaricato nella CPU solo se la compilazione si è conclusa senza errori (Figura 25).

Figura 25: finestra compilazione progetto.

Nel caso di compilazione del progetto conclusa con errori, è possibile visualizzare sull'editor la riga di codice contenente l'errore facendo un doppio click con il mouse sul messaggio fornita dalla finestra di compilazione.

13.5.2 Force Compile

La funzionalità ha lo scopo di indurre la compilazione forzata di tutto il progetto indipendentemente che esso sia già stato compilato o meno.

13.5.3 Ladder Network Checking

Permette eseguire un check della rete Ladder. Sul desktop appare la finestra del risultato della compilazione eseguita (Figura 26).

Figura 26: finestra compilazione rete ladder.

13.5.4 View compilation result

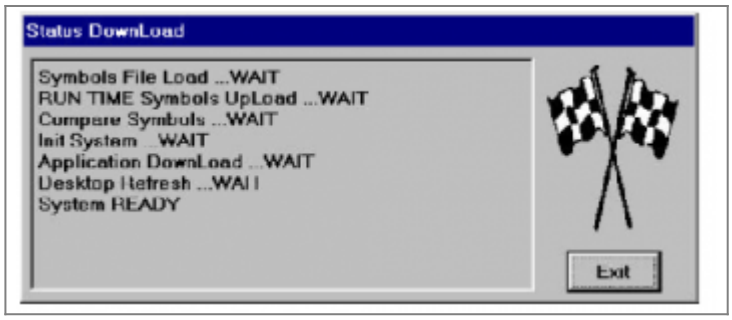
Visualizza o nasconde la finestra del risultato dell'ultima compilazione (Figura 24).

13.5.5 Download

Questo comando è disponibile solo dopo l'attivazione della comunicazione seriale PC - QMOVE.

Permette di scaricare nella CPU il progetto compilato. Le varie fasi del download vengono visualizzate in una finestra dedicata (Figura 27).

Figura 27: esito download.



13.5.6 Backup data

Questo comando è disponibile solo dopo aver eseguito il download dell'applicativo.

L'applicativo che l'utente trasferisce nella CPU è costituito da informazioni non mutabili ed informazioni mutabili.

Informazioni non mutabili Sono le informazioni dell'applicativo che non subiscono variazioni, quali istruzioni QCL, simboli utilizzati nel progetto, titolo applicativo, ...

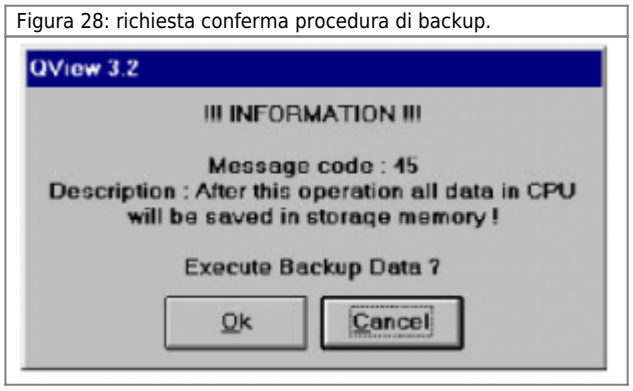
Informazioni mutabili Sono le informazioni che indicano situazioni di funzionamento dell'applicativo quali aree dati per devices interni, valore variabili, array system, contenuto datagroup e dati il cui valore viene modificato durante il funzionamento (tutti i dati applicativo).

Il backup è un comando che consente di creare una copia di sicurezza di tutti i dati mutabili, registrandola all'interno della memoria interna non-volatile. L'utilità di questa operazione deve essere vista nella possibilità di ripristinare tutta la parametrizzazione esistente in un determinato momento. Poichè i valori vengono copiati in una memoria non-volatile, esiste la massima sicurezza nel dispositivo di memorizzazione (Figura 28).
Condizioni che permettono l'esecuzione del comando di backup:

- CPU in stato di READY.
- Applicazione che non utilizzi ram per un valore superiore al limite di backup; la quantità di ram disponibile viene definita al momento dell'acquisto scegliendo tra i tagli di memoria disponibili.
- La somma dello spazio occupato dai dati mutabili e non mutabili non deve superare la dimensione della memoria non-volatile.

Se il backup, eseguito da terminale operatore, impiega un tempo maggiore del timeout del terminale, interviene l'errore di timeout error nella comunicazione seriale tra CPU e terminale.

Figura 28: richiesta conferma procedura di backup.



13.5.7 Restore data

Questo comando è disponibile solo dopo aver eseguito il download dell'applicativo.

L'applicativo che l'utente richiama dalla CPU è costituito da informazioni non mutabili ed informazioni mutabili.

Informazioni non mutabili Sono le informazioni dell'applicativo che non subiscono variazioni, quali istruzioni QCL, simboli utilizzati nel progetto, titolo applicativo, ...
Questi dati sono memorizzati in flash memory.

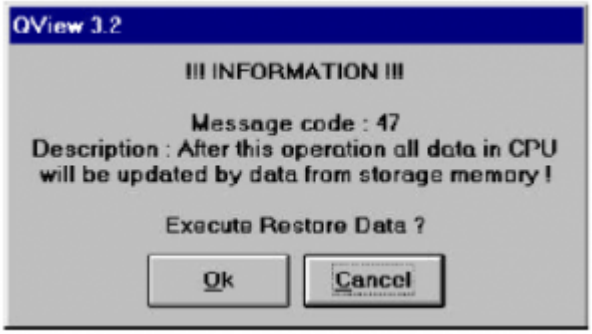
Informazioni mutabili Sono le informazioni che indicano situazioni di funzionamento dell'applicativo quali aree dati per devices interni, valore variabili, array system, contenuto datagroup e dati il cui valore viene modificato durante il funzionamento (tutti i dati applicativo).
Questi dati sono memorizzati in memoria tamponata.

Il comando restore consente di ripristinare tutte le informazioni mutabili con quelle presenti al momento del backup (Figura 29).
Il contenuto del backup viene cancellato durante la procedura di download; infatti non ha alcun senso copiare i valori dei dati

mutabili di un applicativo su un applicativo diverso. Condizioni che permettono l'esecuzione del comando di restore:

- Deve essere stato fatto precedentemente un backup.
- CPU in stato di READY o ERROR.

Figura 29: richiesta conferma procedura di restore.

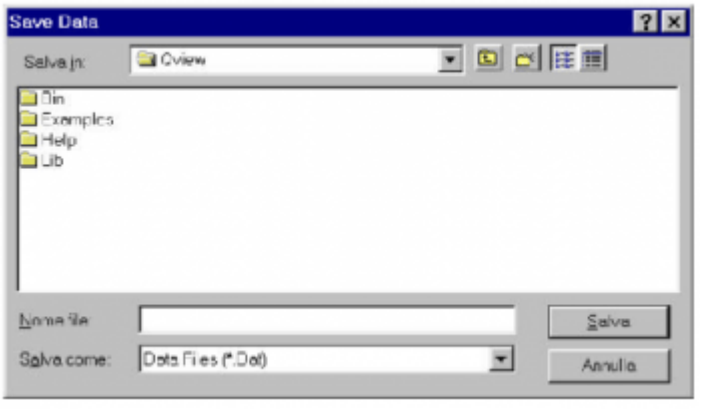


13.5.8 Save Data...

Questo comando è disponibile solo dopo aver eseguito il download dell'applicativo.

Salva i dati presenti nella CPU (valori delle variabili ritentive) in un file .DAT; è possibile definire la directory di destinazione del file (Figura 30).

Figura 30: salvataggio dati CPU su file.

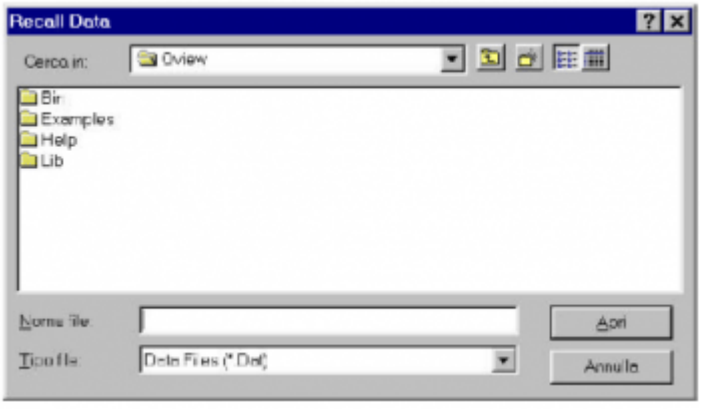


13.5.9 Recall Data...

Questo comando è disponibile solo dopo aver eseguito il download dell'applicativo.

Sostituisce i valori delle variabili ritentive presenti nella CPU con i valori delle stesse variabili archiviate in un file .DAT (Figura 31). La sostituzione interessa solamente le variabili in comune all'applicativo ed al file .DAT; se, per esempio, nella CPU sono state aggiunte delle variabili successivamente all'archiviazione dei dati, queste non saranno interessate dalla sostituzione.

Figura 31: sostituzione dati CPU con file .DAT

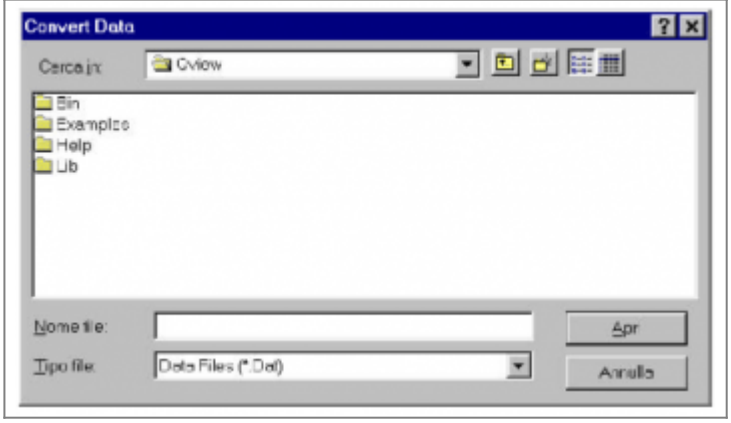


13.5.10 Convert Data...

Questo comando è sempre disponibile.

Converte il file .DAT selezionato in un file di testo (.TXT) contenente il nome delle variabili ed il relativo valore. Il file di testo viene generato all'interno della directory che contiene il file .DAT e gli viene assegnato lo stesso nome del file .DAT (Figura 32).

Figura 32: conversione file dati.



13.5.11 Checksum View

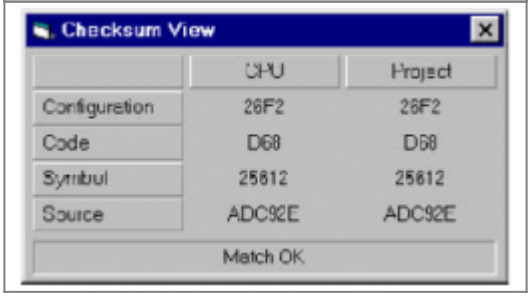
Questo comando è disponibile dopo l'apertura di un progetto.

Confronta i checksums del progetto in uso con quelli dell'applicativo scaricato nella CPU (Figura 33), se sono gli stessi vuol dire che la CPU contiene il progetto su cui sto lavorando. Le eventuali diversità vengono segnalate con i valori di checksum in rosso. Vengono confrontati i seguenti tipi di dati:

- Configuration: configurazione memoria utilizzata.
- Code: codice QCL generato dalla compilazione.
- Symbol: simboli delle variabili utilizzate. Dipende dall'elenco delle variabili e dal loro tipo.
- Source: contenuto delle unità

ATTENZIONE! La compilazione dello stesso progetto con due Qview di build diverse garantisce le stesse funzionalità, ma non viene garantito che i codici checksum si mantengano uguali.

Figura 33: riepilogo e confronto checksum.



La colonna CPU visualizza i valori presenti in CPU e rappresentanti il suo applicativo.
La colonna Project viene aggiornata all'apertura del progetto (se precedentemente compilato) e dopo ogni compilazione.

13.5.12 Project ID View

Questo comando è disponibile solo dopo l'attivazione della comunicazione seriale PC - QMOVE.

Visualizza il nome assegnato al progetto. Questo nome è stato definito nella finestra Project Information. Durante il download il Project ID viene scaricato nella CPU con l'applicativo ed è poi visualizzabile tramite il comando Project ID View (Figura 34).

Figura 34: visualizzazione stringa progetto.

13.6 Menu Debug

Contiene i comandi per l'esecuzione dell'applicativo scaricato nella CPU.

13.6.1 Run

Questo comando è disponibile dopo un download.

Mette in esecuzione l'applicativo scaricato nella CPU..

13.6.2 Stop

Questo comando è disponibile dopo un run.

Interrompe l'esecuzione dell'applicativo residente nella CPU (i device continuano comunque a funzionare).

13.6.3 Restart

Questo comando è disponibile dopo uno start o uno stop.

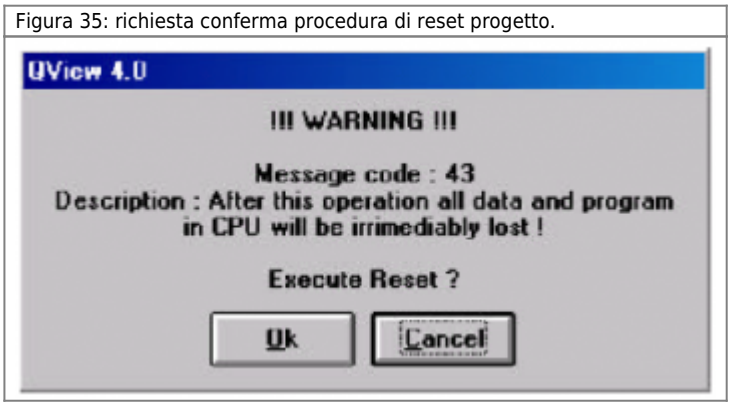
L'esecuzione dell'applicativo viene interrotta e al RUN successivo il programma riparte dalla prima unit.

13.6.4 Reset

Questo comando è disponibile solo dopo l'attivazione della comunicazione seriale PC - QMOVE.

Cancella l'applicativo residente nella CPU. La cancellazione dei dati è definitiva; viene quindi richiesta la conferma al reset dell'applicativo (vedi figura 35).

Figura 35: richiesta conferma procedura di reset progetto.



13.6.5 Step

Questo comando è disponibile solo dopo un download e selezionando la finestra di editor testo.

Questo comando permette di eseguire il progetto in uso un passo alla volta; ad ogni step l'esecuzione del flusso di programma avanza di un passo. Raggiunta l'istruzione di wait, la finestra di editor passa a visualizzare la unit successiva permettendo in questo modo l'esecuzione del programma un passo alla volta su tutte le unit che compongono il progetto.
Una freccia verde sulla sinistra delle finestra di editor identifica la riga di programma che non è ancora stata eseguita (Figura 36). Nel caso di unità LADDER sarà possibile eseguire un rung ad ogni passo.

Figura 36: esecuzione programma in modalità Step.

13.6.6 Step Over

Questo comando è disponibile solo dopo un download e selezionando la finestra di editor testo.

Questo comando permette di eseguire il progetto in uso un'istruzione alla volta; ad ogni step l'esecuzione del progetto avanza

di un'istruzione. Raggiunta l'istruzione di WAIT, l'esecuzione del codice continua normalmente sulle altre unit fino a ritornare a quella in uso riprendendo l'esecuzione passo a passo.
Una freccia verde sulla sinistra delle finestra di editor identifica la riga di programma che è stata eseguita (Figura 37). Nel caso di un'unità LADDER sarà possibile eseguire un rung ad ogni passo.

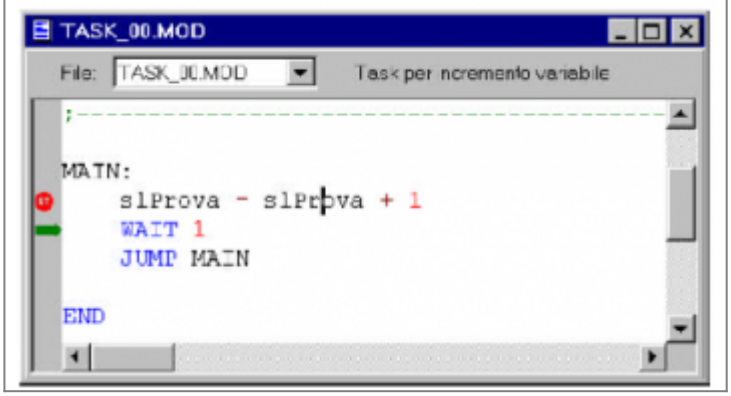
13.6.7 Toggle Breakpoint

Questo comando è disponibile solo dopo un download e selezionando la finestra di editor testo.

E' possibile inserire un massimo di 5 breakpoint. Vengono resettati allo spegnimento della CPU e con il download del progetto.

Questo comando permette di inserire un breakpoint ovvero di identificare una riga di codice, raggiunta la quale, l'esecuzione del programma si interrompe. E' possibile definire più punti di breakpoint. Selezionando una riga di codice sulla quale è stato inserito un breakpoint, il comando *Toggle breakpoint* elimina il breakpoint. L'inserimento del breakpoint viene visualizzato sulla sinistra della finestra di editor con un contrassegno rosso all'interno del quale viene visualizzata la sigla *ST* (Figura 37). E' possibile inserire un breakpoint anche in una unit LADDER in corrispondenza di uno dei rung.

Figura 37: inserimento breakpoint.



13.6.8 Clear All

Questo comando è disponibile solo dopo un download.

Questo comando elimina da tutti i task tutti i breakpoint inseriti.

13.6.9 Watchpoint

Questo comando è disponibile solo dopo un download.

E' possibile inserire un massimo di 5 watchpoint. Vengono resettati allo spegnimento della CPU e con il download del progetto.

Questo comando permette di interrompere l'esecuzione del progetto in uso alla soddisfazione di una particolare condizione; si può quindi parlare di breakpoint condizionato. Selezionando il comando watchpoint viene proposta la finestra di figura 38 che visualizza le condizioni di interruzione del programma che sono state impostate.

- Add: permette di aggiungere una condizione di arresto del programma (Figura 39).
- Delete: elimina la condizione selezionata.
- Reset: elimina tutte le condizioni di arresto impostate.

Figura 38: elenco watchpoint.

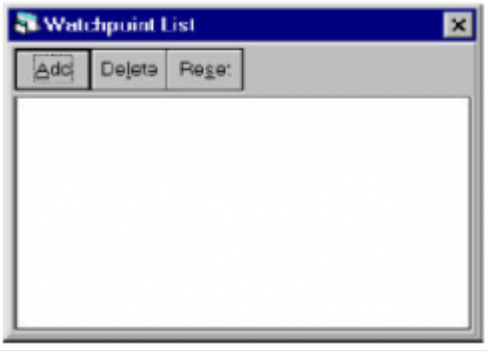
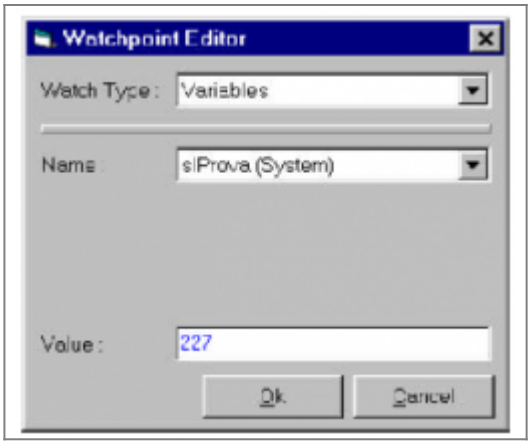


Figura 39: impostazione parametri di watchpoint.



La finestra di figura 39, visualizzata alla pressione del tasto Add (Figura 38), permette di impostare la condizione di arresto del programma.

- Watch Type: selezione la tipologia di variabili (Variables, I/O, Arrays, Data Groups, Devices).
- Name: permette di selezionare la variabile appartenente al gruppo definito nella casella a scorrimento Watch Type.
- Value: permette di inserire il valore della variabile selezionata al quale l'esecuzione del programma si deve arrestare.

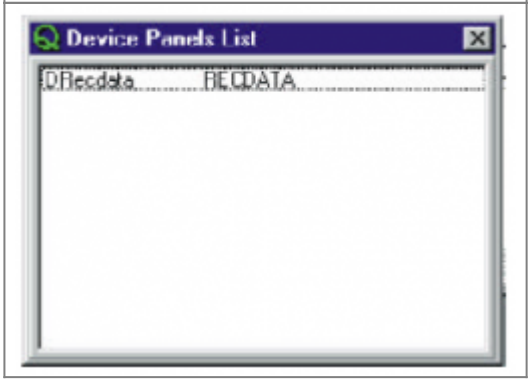
13.7 Menu Monitor

Contiene i comandi per la diagnostica del sistema QMOVE e dell'esecuzione di progetti.

13.7.1 Devices Panels

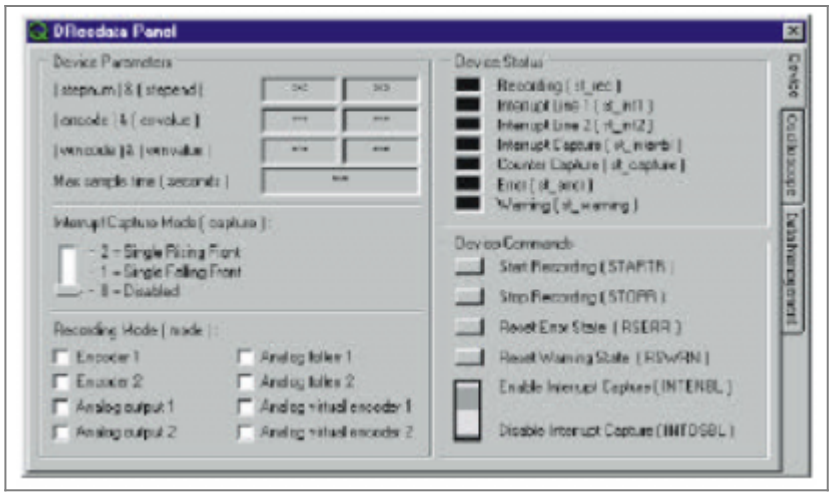
L'ambiente Qview prevede che per i devices ci sia, oltre che alla lista di stati parametri comandi, un pannello che permetta all'utilizzatore un semplice approcio con il devices stesso. vengono visualizzati il nome assegnato al device ed il tipo di device (Figura 51).

Figura 51: elenco pannelli dei device.



Selezionando un device (con il tasto ENTER o con doppio clic) viene proposta una finestra (Figura 52) all'interno della quale è possibile visualizzare i dati del device; per una spiegazione dei devices panel, fare riferimento all'apposito help.

Figura 52: esempio di un pannello.



13.7.2 CPU MONITOR

La finestra *CPU MONITOR* fornisce una serie di informazioni generali riguardanti lo stato della CPU e le tempistiche relative all'esecuzione del progetto. La finestra CPU Monitor Panel si compone di tre sezioni:

- GENERAL: informazioni generali sullo stato della CPU e i tempi di esecuzione dell'intero progetto.
- TASK INFO: informazioni relative ad ogni singola unit che compone il progetto.
- CPU’s oscilloscope: strumento che permette di determinare l'andamento nel tempo del valore di una o più variabili del progetto.

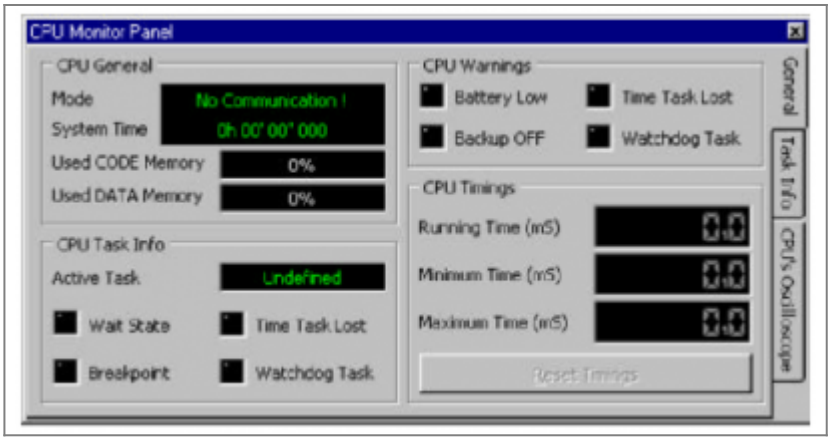
13.7.3 General

In questa finestra vengono fornite informazioni relative allo stato della CPU.

- Mode: visualizza lo stato della CPU (RUN, STOP, … Err). In caso di errore viene visualizzato anche il tipo di errore.
- System Time: visualizza il tempo reale di funzionamento dell’applicativo dal momento dell’accensione o restart della CPU.
- Used CODE Memory: visualizza la percentuale di memoria impegnata dal codice.
- Used DATA Memory: visualizza la percentuale di memoria impegnata dai dati.
- Battery low: segnala lo stato di batteria tampone interna scarica.
- Watchdog Task: segnala che vengono impiegati più di 200 ms per eseguire un task. Questa segnalazione viene mantenuta fino allo spegnimento della CPU. Il progetto continua a funzionare correttamente, tuttavia la segnalazione indica un’errata programmazione QCL.
- Backup OFF: segnala l’impossibilità di eseguire il backup (ad esempio perchè viene impegnata una percentuale elevata di DATA Memory).
- Time Task Lost: segnala che nell’esecuzione del programma non è stato eseguito una o più volte un task a tempo.
- Running time in ms: tempo di esecuzione dei task.
- Active Task: task in uso.
- Time Task Lost: segnala che nell’esecuzione del programma non è stato eseguito una o più volte un task a tempo.
- Wait State: segnala il raggiungimento dell’istruzione di wait (all’interno del task in uso).
- Breakpoint: segnala che il flusso del task attivo ha raggiunto un breackpoint e si è interrotto.
- Watchdog Active: segnala che il task attivo ha causato la segnalazione di Watchdog.

Si possono, inoltre, leggere i tempi MINIMUM TIME e MAXIMUM TIME che sono il minimo e massimo tempo impiegato per eseguire il progetto.
Vi è inoltre un tasto di *Reset Timing* il quale azzera i due valori ed il *Running time* riabilitando il conteggio del tempo di elaborazione.

Figura 56: CPU general panel.



13.7.3.1 Task info

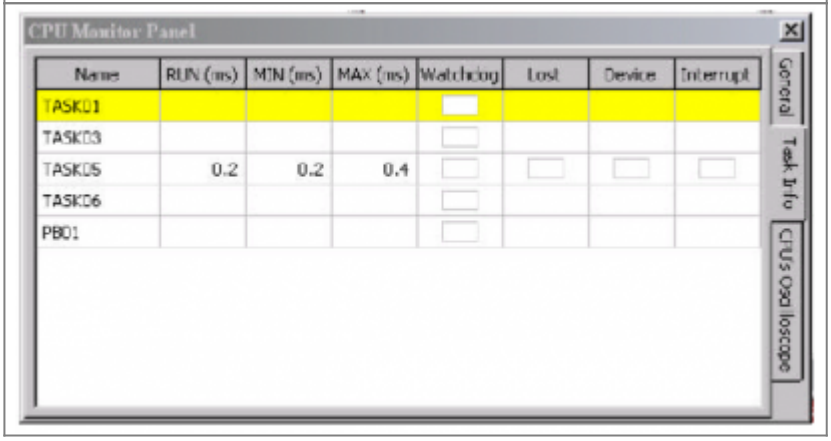
In questo menù possono essere controllati tutte le unit elaborati dal programma, e possono essere ricavate informazioni di vario genere. Tutti le unit possono dare infomazioni sul tempo di esecuzione minimo, massimo ed attuale.

Per i task normali si possono avere inoltre informazioni relative al *WATCHDOG*. Quando viene segnalato un *WATCHDOG* per un task significa che la CPU ha impiegato più di 200ms per eseguire il suo codice senza mai avere eseguito nessun altro task (questa è una anomalia che il programmatore dovrebbe risolvere).

Per tutti gli altri task si possono avere ulteriori informazioni relative a :

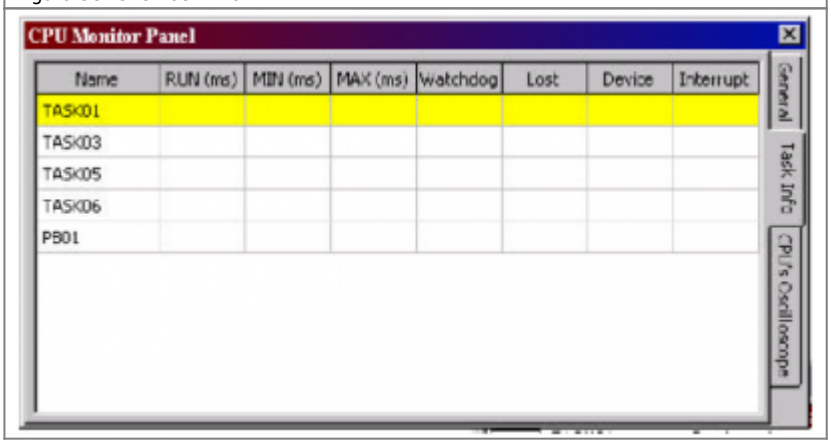
- **TASK LOST:** Indica che il task a tempo ha perso un evento.
- **DEVICE ACCESS:** Indica che il task ha eseguito un accesso a device e che questo non era aggiornato.
- **INTERRUPT:** Indica che un task a tempo è stato interrotto da un task in interrupt (Figura 57).

Figura 57: CPU general panel.



Tutte le informazioni della videata precedente (Figura 57) possono essere visualizzate solo se i check-sum del programma in CPU e su PC corrispondono, altrimenti, nella videata di Task Info verranno visualizzati solamente i task di programma senza alcun'altra informazione (Figura 58).

Figura 58: CPU Task info.

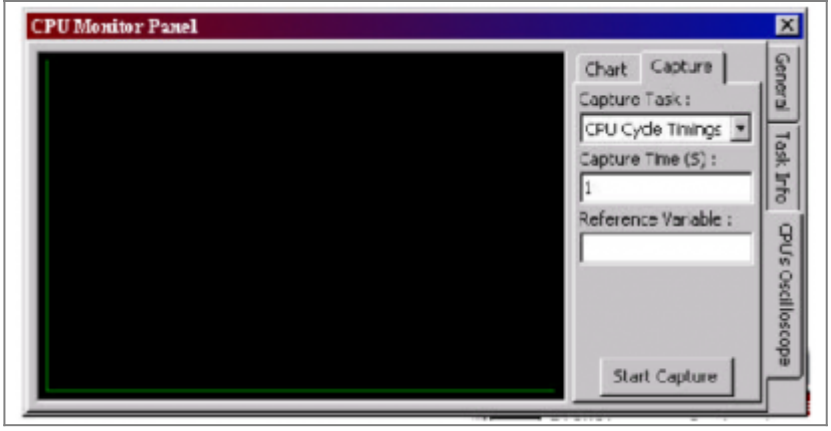


Nel caso di errori, i vari campi assumeranno il colore rosso, in caso contrario, i campi rimarranno di colore bianco.

13.7.3.2 CPU's Oscilloscope

L'oscilloscopio permette di visionare nel tempo cio' che accade alla CPU (Figura 59).

Figura 59: CPU Oscilloscopio.



La velocità di acquisizione è determinata da due fattori:
- Velocità della seriale. - Quanti dati vogliamo controllare.

Se si vuole fare un'acquisizione si inserisce il *Capture Time (S)* e la scansione inizia continuando per il tempo impostato, comunque non superiore ai 9999 sec.
Quando si lancia una acquisizione tutto il resto del Qview viene bloccato proprio per non togliere risorse a questa funzionalità.
L'acquisizione può essere associata ad una variabile qualunque, tranne un datagroup, e l'acquisizione nel tempo può essere abilitata anche con macchina in funzione senza pregiudicarne il funzionamento in quanto il controllo non grava sull'elaborazione.
L'acquisizione può essere fermata in qualunque momento tramite la pressione del tasto Backspace.

Impostazioni per la memorizzazione:

CAPTURE TASK: Questo parametro determina in quale campo agire per effettuare la misurazione:
- Su tutti i task , se impostato come CPU CICLE TIMINGS , in questo modo possiamo vedere il tempo di ciclo di tutto il programma (Min. Max. Run). - Su un task singolo, se impostato come task a tempo o ad interrupt, in questo modo possiamo vedere il tempo di ciclo del singolo task seleziona to (Min. Max. Run). Naturalmente se il task selezionato risulta essere un task normale il tempo di scansione che verrà visualizzato sara' quello di tutto il programma, questo particolare caso comunque viene evidenziato con l'ausilio di un messaggio (Figura 60).

Figura 60: CPU Oscilloscopio Capture Task.

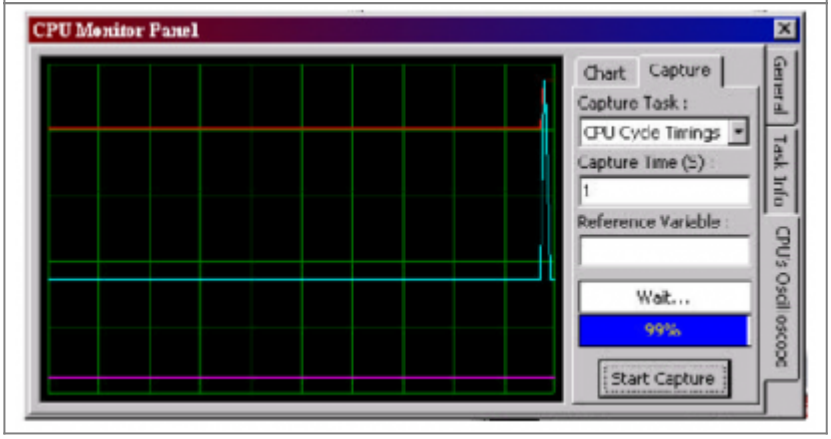
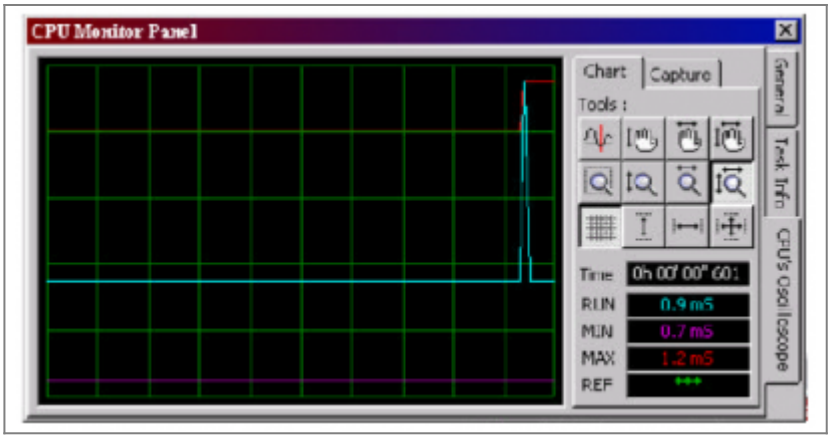


CHART: Questa finestra serve per impostare al meglio le caratteristiche grafiche di visualizzazione ed i riferimenti relativi alle misurazioni effettuate (Figura 61).

Figura 61: CPU Oscilloscopio Chart.



Per conoscere il significato dei vari tasti basta posizionarsi sopra al tasto interessato con il cursore del mouse e comparirà la descrizione specifica.
Oltre alle misure delle tempistiche dei task è possibile monitorare l'andamento nel tempo di una variabile di riferimento che si puo specificare nella casella "Reference variable" nella sezione Capture.
Se la variabile non esiste compare un messaggio di segnalazione, ma la registrazione viene compiuta ugualmente senza, naturalmente, la variabile.

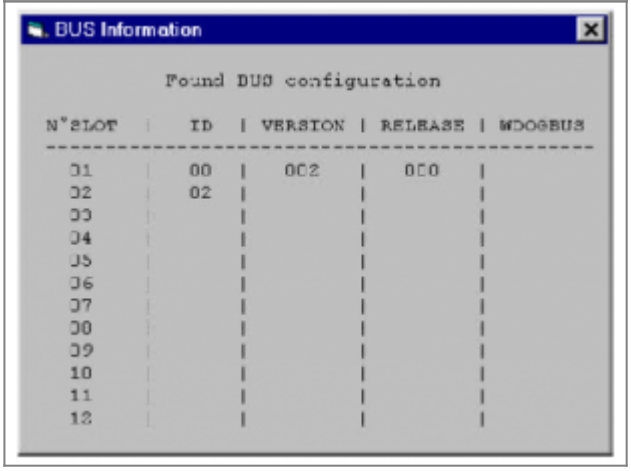
13.7.4 BUS

Questo comando è disponibile dopo l'apertura di un progetto esistente e l'apertura della porta di comunicazione seriale PC - QMOVE.

La finestra BUS Information (Figura 63) visualizza la composizione hardware e le versioni/release dei firmware. Queste informazioni vengono acquisite dalla CPU tra il BUS; per questo motivo sono disponibili solamente se la comunicazione è attiva.

- N° SLOT: posizione nello slot.
- ID: identifica il tipo di scheda installata.
- VERSION: indica la versione del firmware installato sulla scheda.
- RELEASE: indica la release del firmware installato sulla scheda.
- WDOGBUS: segnala le eventuali anomalie di funzionamento delle schede.

Figura 63: composizione bus.



13.8 Menu Tools

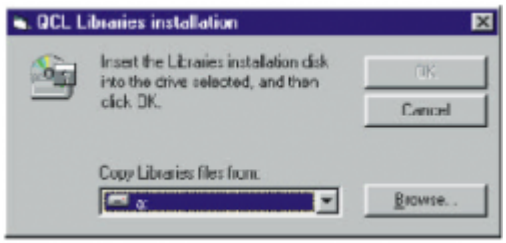
13.8.1 Upgrade QCL Card Library

Questo comando è sempre disponibile.

Il comando Upgrade QCL Card Library permette di aggiornare le librerie del linguaggio QCL (Figura 64). Tali librerie contengono informazioni relative a:

- Set dei modelli di QMOVE utilizzabili.
- Set di schede installabili sul QMOVE.
- Set di device presenti nel QMOVE.

Figura 64: aggiornamento librerie.



13.8.2 Upgrade QCL Functions Library

Questo comando è sempre disponibile.

Il comando Upgrade Functions Library permette di aggiornare le librerie di funzioni QCL (Figura 65).

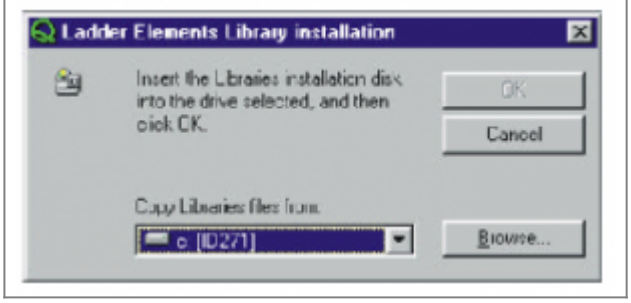
Figura 65: aggiornamento librerie delle funzioni QCL.

13.8.3 Upgrade Ladder Elements Library

Questo comando è sempre disponibile.

Il comando Upgrade ladder elements permette di aggiornare le librerie degli elementi ladder (Figura 67).

Figura 67: aggiornamento librerie Ladder.



13.9 Menu Options

I comandi del menu Options permettono di personalizzare alcune funzionalità di QVIEW.

13.9.1 Open COM / Close COM

Permette di aprire e chiudere la porta di comunicazione seriale PC - QMOVE.

13.9.2 Program Setup ...

Questo comando è sempre disponibile.

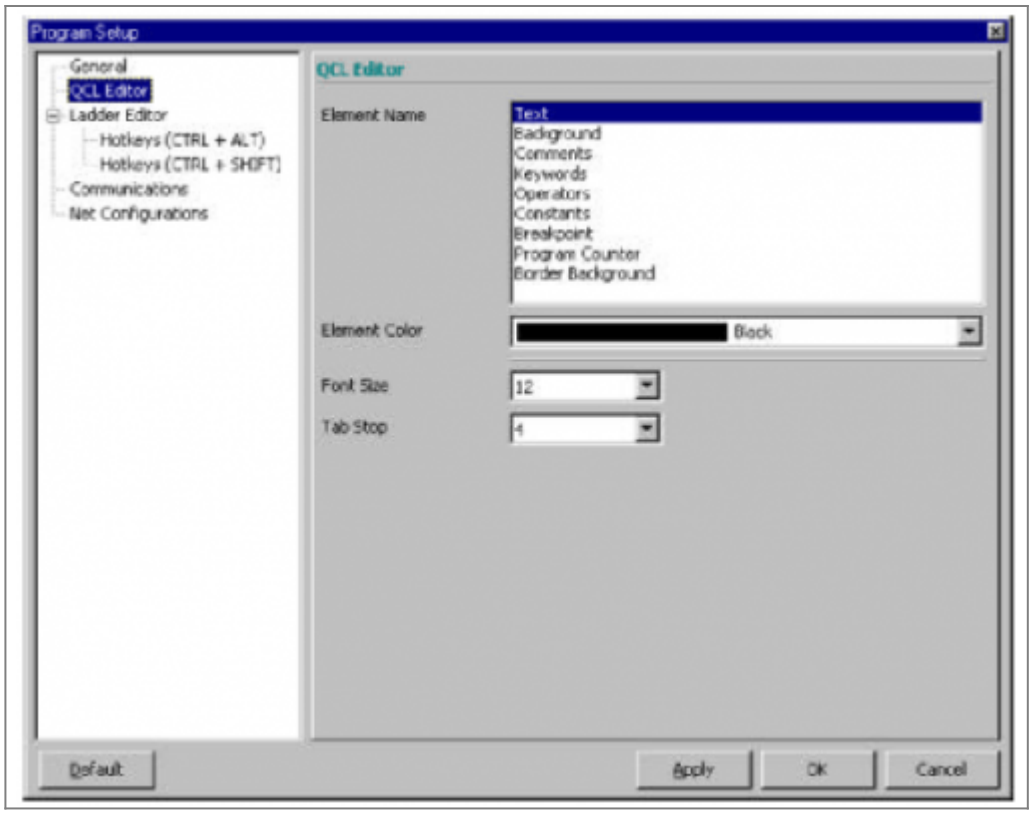
Viene presentata una finestra composta da sei diverse cartelle.

13.9.2.1 Cartella QCL Editor

Questo comando è sempre disponibile.

Permette di personalizzare l'interfaccia grafica di QVIEW, assegnando dimensioni e colori diversi in funzione del testo QCL da editare (Figura 68).

Figura 68: personalizzazione editor QCL.



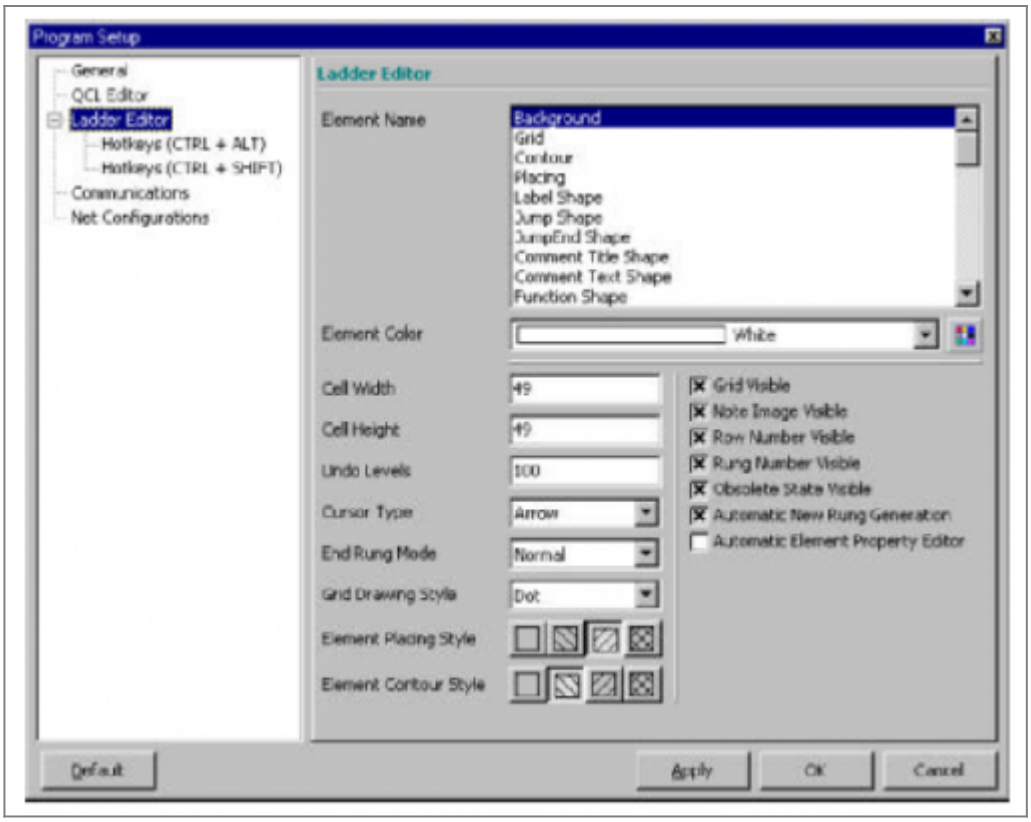
* Font Size: definisce le dimensione del font utilizzato nella finestra di editor.

- Tab Stop: definisce la lunghezza della tabulazione.
- Text Color: definisce il colore del testo (codice QCL).
- Element Color: permette di specificare il colore dell'elemento selezionato.
- Element Name: permette di selezionare gli elementi dell'editor per specificarne il colore. Gli elementi selezionabili sono:
    - Text: testo presente nell'editor.
    - Background: sfondo della finestra di editor.
    - Comments: commentipresenti nel programma.
    - Keywords: parole chiave.
    - Operators: operatori.
    - Constant: costanti.
    - Breakpoint: simbolo di breakpoint.
    - Program Counter: puntatore di programma (visibile solamente in condizioni di stop).
    - Border Background: bordo della finestra di editor.

13.9.2.2 Cartella Ladder Editor

Permette di personalizzare l'interfaccia grafica di QVIEW, assegnando dimensioni e colori diversi agli elementi che conpongono l'editor LADDER (Figura 69).

Figura 69: personalizzazione editor Ladder.

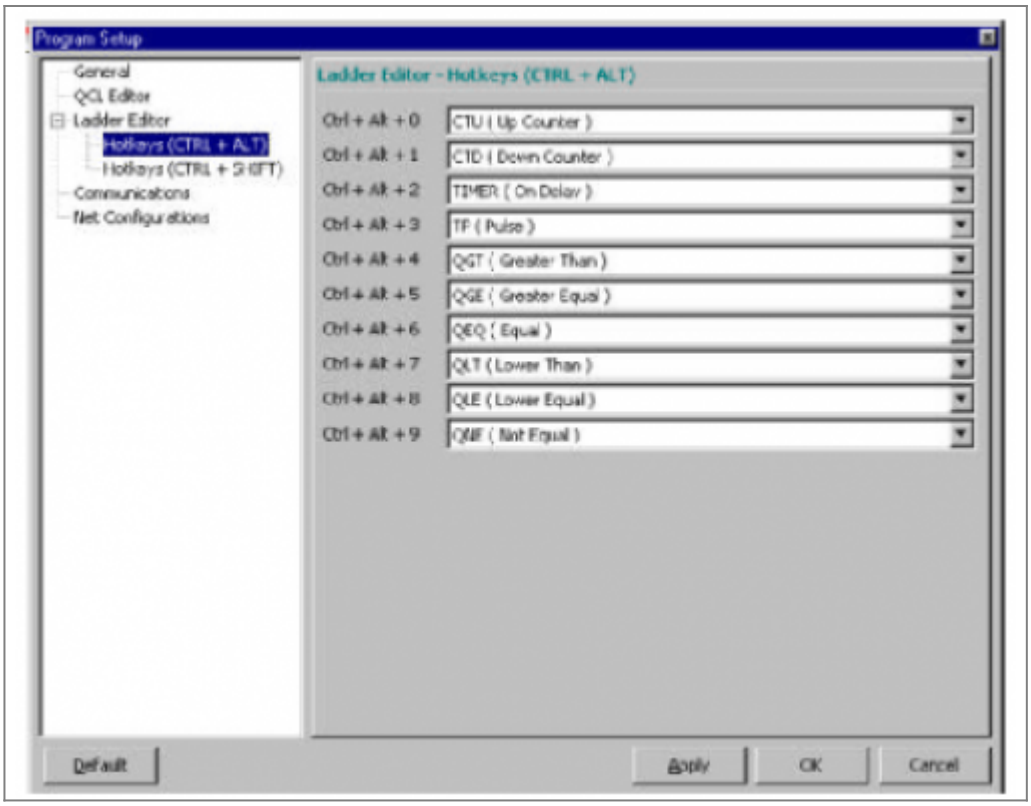


- Cell Width: definisce la larghezza delle caselle dell'editor.
- Cell Height: definisce l'altezza delle caselle dell'editor.
- Undo Levels: numero massimo di UNDO possibili.
- Cursor type: scelta del tipo di cursore.
- End Rung Mode: scelta del tipo di fine rung.
- Grid Drawing Style: scelta del tipo di tratto utilizzato per disegnare la griglia.
- Element Placing Style: grafica utilizzata come sfondo degli elementi LADDER selezionati.
- Element Contour Style: grafica utilizzata come sfondo dell'area dell'editor selezionata.
- Grid Visible: seleziona se rendere visibile la griglia.
- Note image Visible: seleziona se rendere visibili il simbolo che indica la presenza di una nota associata ad un elemento LADDER.
- Row Number visible: seleziona se rendere visibile i numeri di riga.
- Rung Number visible: seleziona se rendere visibile i numeri di rung.
- Obsolete State Visible: seleziona se indicare gli elementi LADDER obsoleti (cambiandone i lcolore di sfondo).
- Automatic New Rung Generation: seleziona se rendere possibile la generazione automatica dei rung al momento dell'inserimento di un nuovo elemento LADDER.
- Automatic Element Property Editor: seleziona se far apparire automaticamente la finestra con le proprietà dell'elemento LADDER una volta che viene inserito.
- Color: definisce il colore dello sfondo della finestra di editor.

- Element Color: permette di specificare il colore dell'elemento selezionato.
- Element Name: permette di selezionare gli elementi dell'editor per specificarne il colore.
Gli elementi selezionabili sono elencati in questa finestra.

In questa sezione del "Program Setup" è possibile associare ad una serie di HotKeys (Tasti Caldi) l'inserimento degli elementi LADDER più utilizzati. Nella Figura 70 è possibile vedere che è possibile associare l'inserimento di 10 elementi LADDER alla combinazione dei tasti CTRL + ALT + 0…9 e altri 10 a CTRL + SHIFT + 0…9. Esiste già un associamento per default, ma il programmatore potrà modificarlo.

Figura 70: associazione ai tasti caldi.

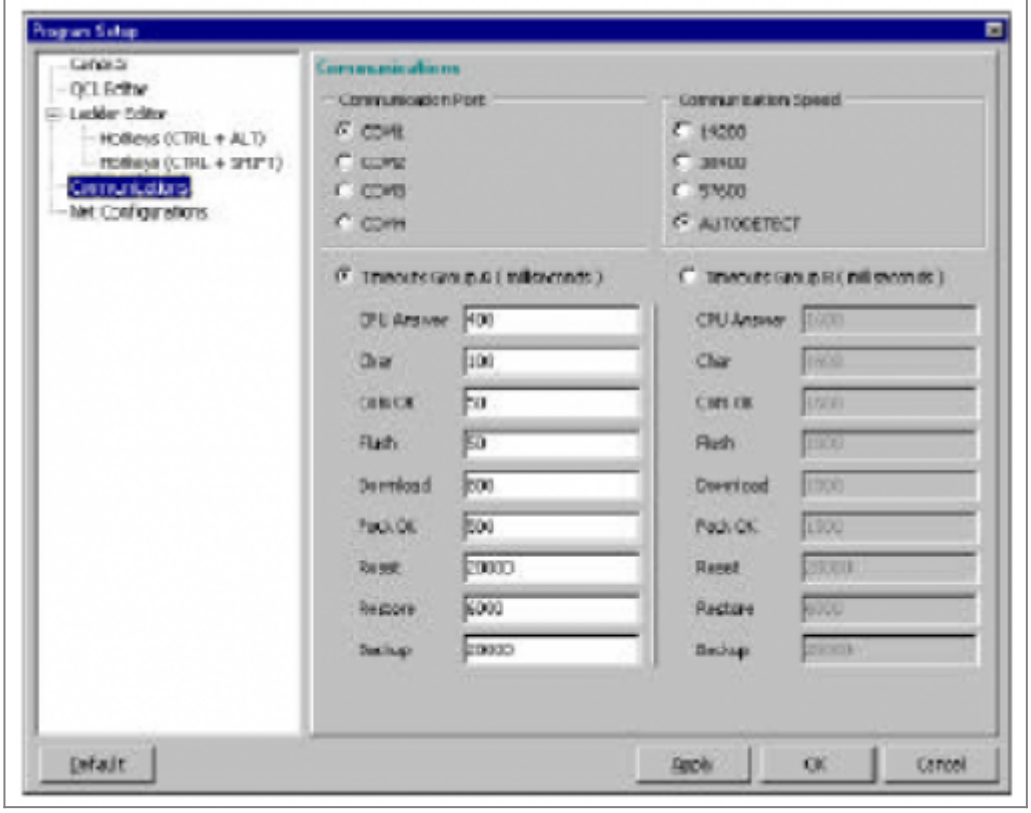


13.9.2.3 Cartella Communication

Per utilizzare le porte COM3 e COM4 è necessario adottare degli IRQ diversi da quelli utilizzati per altri dispositivi hardware (mouse, COM1, COM2, ...).

Permette di configurare la porta di comunicazione seriale (Figura 71).

Figura 71: impostazione parametri porta di comunicazione seriale.



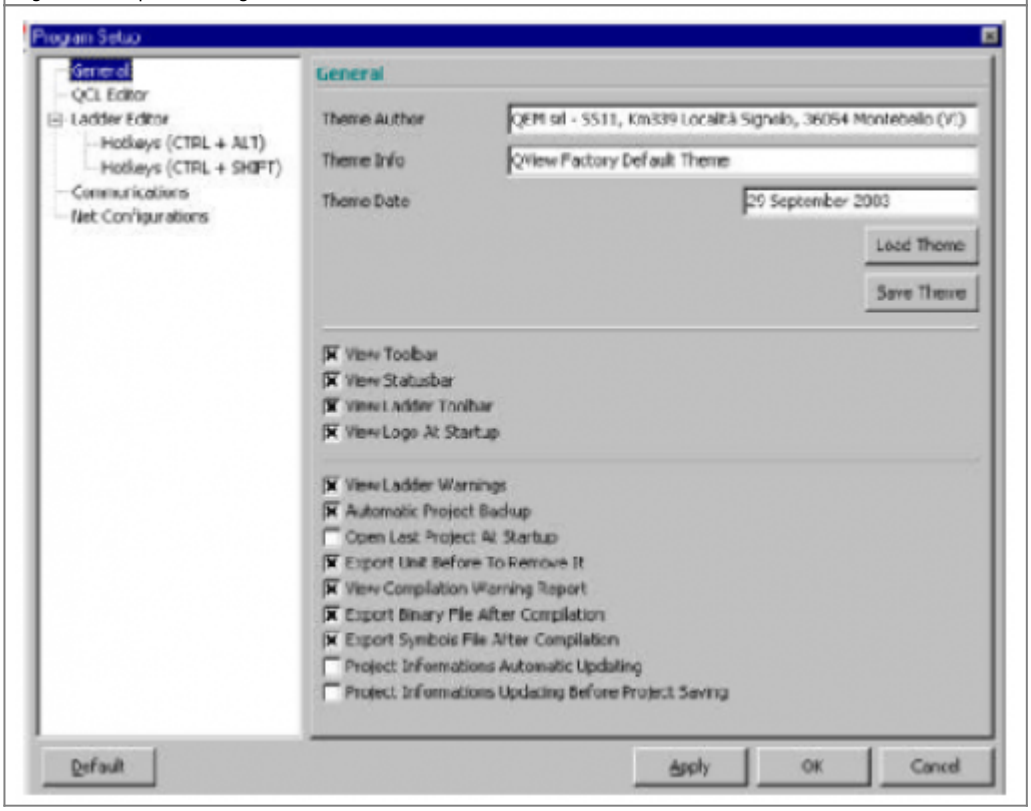
- COM Port: permette di selezionare la porta di comunicazione seriale da utilizzare. - Communication Speed: definisce la velocità di trasmissione della comunicazione seriale; selezionando l'opzione autodetect, QVIEW rileva automaticamente la velocità di trasmissione da adottare.

E' possibile selezionare tra due gruppi di impostazione dei tempi della comunicazione seriale (Timeouts Group A e Timeouts Group B).

13.9.2.4 Cartella General

Permette di personalizzare l'ambiente di sviluppo (Figura 72).

Figura 72: impostazioni generali.



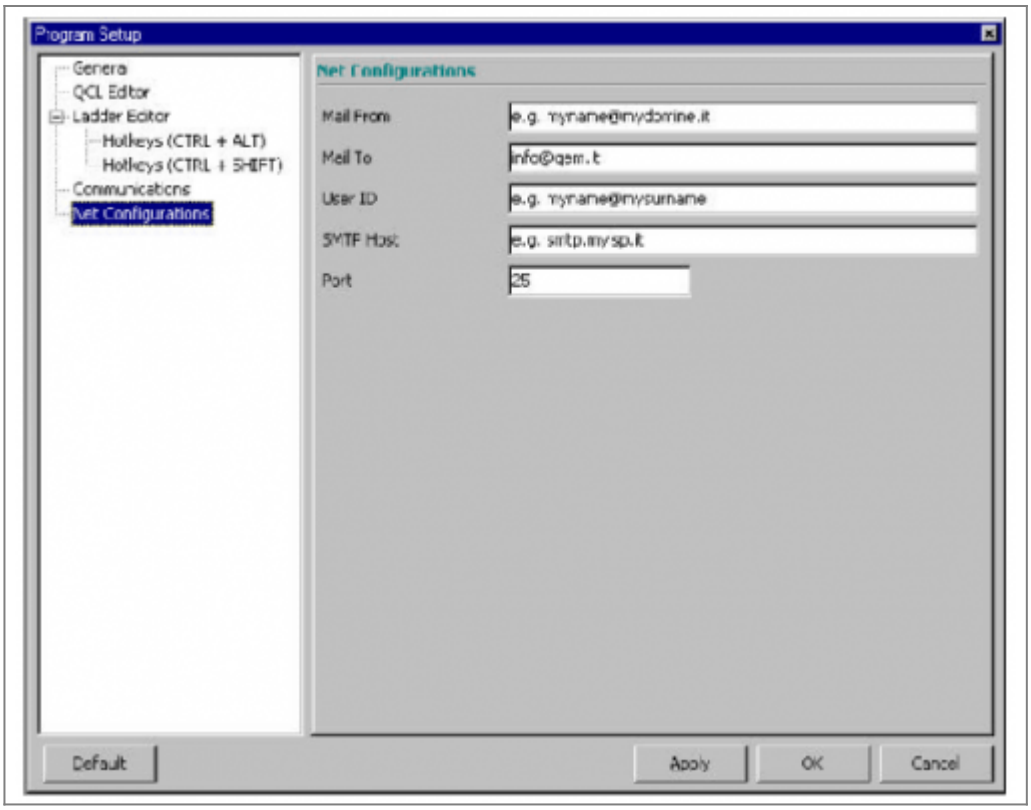
- View Toolbar: abilita la visualizzazione della toolbar del progetto.
- View Status Bar: abilita la visualizzazione della barra di stato.
- View Ladder Toolbar: abilita la visualizzazione della toolbar del ladder.
- View Logo At Startup: abilita la visualizzazione del logo Qview allo startup del programma.
- View Ladder Warnings: abilita la visualizzazione dei messaggi di warning nella finestra con i risultati della compilazione.
- Automatic Project Backup: abilita l'esecuzione di una copia di backup del progetto ogni volta che viene salvato.
- Open the last project at startup: abilita l'apertura dell'ultimo progetto salvato ad ogni apertura del Qview.
- Export unit before remove it: abilita la richiesta di esportazione dell'unità prima della cancellazione della stessa.
- View Compilation Warning report: abilita la visualizzazione dei messagi di warning durante la compilazione.
- Export Binary file After Compilation: abilita l'esportazione automatica del file binario risultato della compilazione.
- Export symbol file after compiling: abilita l'esportazione automatica del file simboli ad ogni compilazione andata a buon fine. Il file simboli verra' esportato nella stessa directory del file di progetto con lo stesso nome del file di progetto. Se il progetto è nuovo e non ancora salvato su disco, Qview genererà una finestra di dialogo per chiedere dove ubicare il file simboli.
- Automatic Project Information update: se selezionato appare un messaggio in cui si chiede all'utente se le informazioni di progetto contenute in "Project information" sono state aggiornate.
Questo messaggio appare se l'applicativo non viene salvato da almeno un'ora.
- Project Information update before project saving: se selezionato, ogni volta che si salva il progetto, appare un messaggio in cui si avvisa che i dati obbligatori (segnalati con un asterisco) da inserire in "Project information" non sono completi.

In questa finestra è possibile salvare le impostazioni in un file e specificare anche alcuni dati relativi all'autore.

13.9.2.5 Cartella Net Configurations

Permette di impostare i dati necessari ad inviare un'email direttamente da Qview4 senza l'ausilio di un mail client esterno (Figura 73).

Vedi figura 73: impostazioni generali.



- Mail From...: in questo campo è possibile impostare l'indirizzo da dove la mail dovra' partire. Tale indirizzo potrebbe essere il proprio indirizzo email.
- Mail To...: è l'indirizzo di chi riceverà la mail. Solitamente va impostato come support@qem.it, ma è possibile impostare liberamente anche altri indirizzi email.
- User ID: ID Utente necessario per effettuare il collegamento al provider di servizi internet (ISP).
- SMTP Host: è l'indirizzo del server di posta elettronica.
- Port: è la porta IP di comunicazione. Generalmente il valore 25 è corretto per la stragrande maggioranza dei casi.

13.10 Menu Help

Questo comando è sempre disponibile.

Dal menu help è possibile richiamare gli help in linea e alcune informazioni su QVIEW.

13.10.1 Contents

Richiama l'help in linea relativo al linguaggio di programmazione.

13.10.2 QCL Language guide

Richiama l'help in linea per la programmazione in QCL.

13.10.3 Ladder Functions info

Richiama l'help in linea relativo agli elementi LADDER.

13.10.4 Functions info

Richiama l'help in linea relativo alle funzioni QCL.

13.10.5 User Functions info

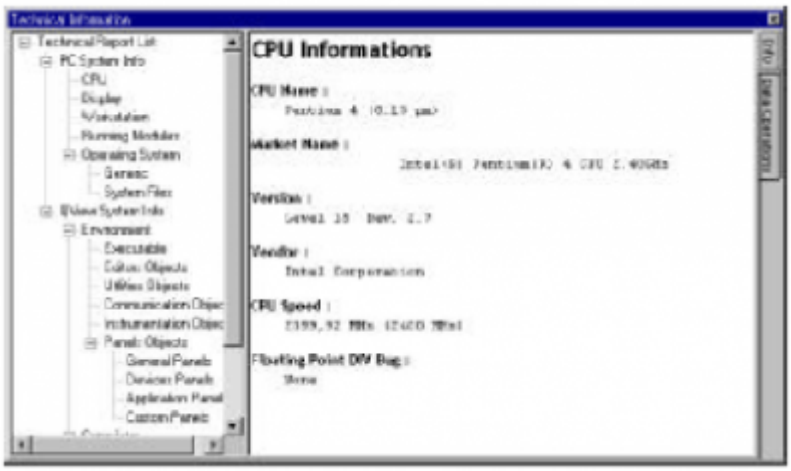
Richiama l'help in linea relativo alle funzioni QCL realizzate dall'utente.

13.10.6 Technical Info

Per accederte a Technical Info bata selezionare: TECHNICAL INFO. Viene visualizzata la finestra di figura 74 dove vengono raccolte tutte le informazioni del Personal Computer nel quale è istallato il Qview, complete di impostazioni e setup particolari , vengono

inoltre registrate le DLL utilizzate per permettere di avere il massimo numero di informazioni possibili.
Una volta raccolte tutte queste informazioni, è possibile trasferirle in modo automatico su un documento Word pronte per essere inviate per Email (Figura 76), oppure si può eseguire l'invio di queste informazioni via email in modo automatico direttamente da Qview 4 (Data Operations).

Figura 74: informazioni relative ai software e hardware installati.



Le informazioni sono suddivise in tre categorie.
- PC System Info: raccoglie tutte le informazioni Hardware e Software del Computer con il quale si stà lavorando.
- Qview System Info: raccoglie tutte le informazioni relative al programma Qview che si stà utilizzando.
- Project Information Info: raccoglie tutte le informazioni relative al progetto aperto in Qview.

13.10.7 About Qview

Viene visualizzata la versione di QVIEW e gli estremi per contattare la QEM srl.

14. Debug

Come è stato anticipato all'inizio di questo manuale, QVIEW è un indispensabile supporto per la programmazione in QCL e LADDER, sia per la stesura e la compilazione del codice, sia per il debugging del progetto realizzato.
Cos'è il debugging? Solitamente il tempo impiegato per la realizzazione di un progetto si suddivide in maniera equa tra il tempo per la programmazione e il tempo per la correzione degli errori. Il debugging è l'insieme di tutte le operazioni che permettono di rilevare questi errori che causano funzionamenti non desiderati. Gli strumenti che il QCL e LADDER mette a disposizione sono i seguenti:

- Esecuzione passo-passo del progetto (Step).
- Esecuzione passo-passo del singolo modulo (Step over).
- Inserimento di breakpoint.
- Watchpoint.

14.0.1 Esecuzione passo-passo

Una volta compilato e fatto il download, è possibile mettere in esecuzione l'applicativo scaricato selezionando il comando **Run** dal menu **Debug**; è anche possibile eseguire un'istruzione per volta, selezionando il comando **Step** dal menu **Debug**.
Ad ogni comando di **Step**, si può osservare lo scorrere di una istruzione; sull'editor viene evidenziata la riga di codice in esecuzione con una freccia sul bordo sinistro della finestra. Durante la scansione del codice, si può osservare l'effettivo flusso delle istruzioni ed il valore delle variabili.
Il comando **Step** esegue fedelmente la successione delle istruzioni QCL e LADDER e, la visualizzazione sull'editor, si sposta di unit in unit quando incontra un'istruzione di WAIT. Nel caso di unit scritte in LADDER l'esecuzione passo-passo consiste nel eseguire il codice un rung per ogni passo.

14.0.2 Esecuzione passo-passo della singola unit

Il comando **Debug > Step Over** ha lo stesso effetto del comando **Step**; la successione delle istruzioni visualizzate si limita però a quelle dell'unità visualizzata sull'editor. E' un comando utile per la scansionare il codice contenuto in una sola delle unità componenti il progetto.

14.0.3 Inserimento di breakpoint

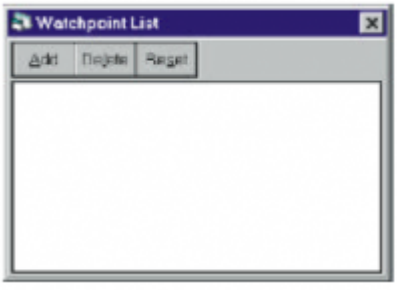
Il breakpoint è un punto preciso del programma in cui l'esecuzione del progetto si deve interrompere. Per fissare un breakpoint, posizionare con il cursore nel punto dell'editor in cui si desidera bloccare il programma e selezionare il comando **Debug > Toggle Breakpoint**; in corrispondenza dell'introduzione del breakpoint viene visualizzato il simbolo 

. Con comando di RUN il programma verrà eseguito fino all'istruzione marcata dal breakpoint (esclusa) e quindi si bloccherà; la CPU assumerà lo stato di STOP. Nella finestra CPU si accenderà il LED Breakpoint per indicare l'intervento di un breakpoint. Sull'editor la riga interessata dal breakpoint viene indicata con il simbolo  . Con un nuovo RUN il programma verrà eseguito ancora una volta fino ad incontrare di nuovo il breakpoint. Questa funzionalità è molto utile per osservare lo stato delle variabili in un certo punto del codice che non sarebbe possibile apprezzare in fase di esecuzione normale. Inoltre il sistema del breakpoint può essere usato per vedere se una parte di codice viene eseguita oppure no. E' possibile posizionare un numero massimo di 7 breakpoint contemporaneamente lungo il codice. Nel caso ci siano più di un breakpoint per visualizzare velocemente il punto del codice su cui si è bloccato il programma si può usare l'icona  (Go To PC).

14.0.4 Watchpoint

Il watchpoint è un breakpoint condizionato dal valore di una variabile, di un ingresso o di un'uscita digitale, di un array, di un Data Group o di un parametro di un device.
La domanda a cui risponde il watchpoint è: in che punto del programma un parametro o una variabile fissata acquisisce il valore impostato?
Selezionando **Debug > WatchPoint**, appare la finestra ***Watchpoint List*** (Figura 1). Questa finestra contiene il parametro su cui impostare il watchpoint e il valore a cui deve "scattare".

Figura 1:elenco watchpoint.



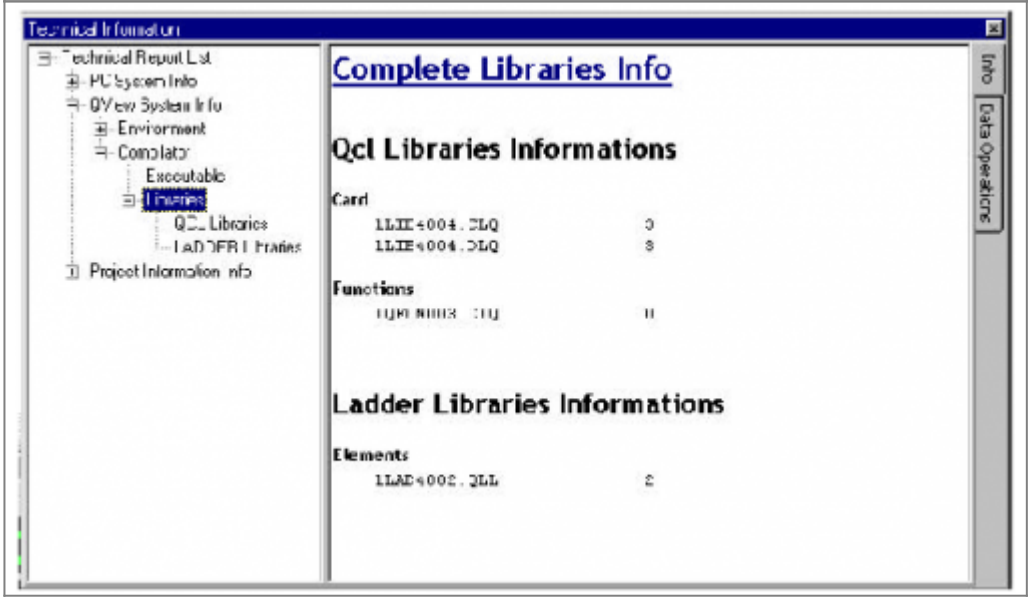
Con il tasto ***Add*** è possibile scegliere nome e tipo di parametro da usare per il watchpoint.
Una volta introdotte queste informazioni, digitare il valore al quale il watchpoint si deve attivare per bloccare l'esecuzione del programma.

15. Le librerie QCL

Le librerie QCL sono una componente fondamentale per il compilatore QCL, mettendo a disposizione - in fase di compilazione - una serie di informazioni riguardanti i device e l'hardware. Tali informazioni sono necessarie per sapere quali parametri utilizza un device interno, le loro caratteristiche e la loro collocazione in memoria; quali comandi sono disponibili per quel determinato device e la sua sintassi di configurazione.

Al momento della realizzazione di un nuovo progetto è quindi necessario verificare che la versione di QVIEW installata contenga le librerie necessarie per la gestione dei devices e dell'hardware da utilizzare (fare riferimento alle schede tecniche hardware).
Selezionare la voce *Technical Info* dal menu *Help*; Appare la finestra di figura 1.

Figura 1: componenti software installati.



Alla riga LIBRARY viene specificato l'identificatore della libreria in uso (nell'esempio 1LIB3004).
Nel caso non si sia in possesso delle librerie richieste è necessario fare l'upgrade delle stesse selezionando *Upgrade Library…* dal menu *Tools* e seguire le istruzioni proposte nelle finestre che appariranno.

16. Appendice A: Limitazioni QCL

16.0.1 Numero massimo di etichette

Durante la compilazione di alcune istruzioni QCL il compilatore genera internamente delle etichette che vengono utilizzate per successive elaborazioni. Esiste un limite di 999 etichette generabili per ogni unità, superato il quale non è possibile compilare il file sorgente. Per conoscere il numero di etichette interne generate bisogna sapere che le istruzioni IF, ELSE, CALL, SUB, o label (es. MAIN) generano una etichetta, mentre le istruzioni FOR e WHILE ne generano due. L'unica soluzione è quella di eliminare dal sorgente alcune delle istruzioni viste sopra e rientrare nei limiti ammessi.

16.0.2 Numero massimo di operandi annidati

Durante la compilazione di espressioni il massimo numero di operandi annidati è 6.
Ad esempio una istruzione di questo tipo è ammessa:
Variabile = 1+(1+(1+(1+(1+1))))
mentre questa provoca un errore:
Variabile = 1+(1+(1+(1+(1+(1+1)))))

16.0.3 Numero massimo di elementi

Il numero massimo di elementi e passi ammessi in un datagroup è di 65534. Il compilatore segnala errore se tale limite viene superato.

16.0.4 Dimensione massima di un Array

La dimensione massima di un array (sia ARRSYS che ARRGBL) è di 65535 elementi. Il compilatore segnala errore se tale limite viene superato.

16.0.5 Ciclo FOR

Il passo di incremento in un ciclo FOR deve essere numerico. Non è possibile usare variabili o espressioni.

16.0.6 Datagroup

Nella dichiarazione di un datagroup la sottosezione DATAPROGRAM è obbligatoria.

17. Appendice B: Conversione e promozione di tipo

17.1 Conversione di tipo

È importante ricordare che una conversione dal tipo intero (Flag, Byte Word o Long) al tipo Single non corrisponde ad un aumento di precisione ma cambia solamente il formato in cui il valore è rappresentato.

Per espressione si intende un insieme di operatori, costanti e variabili che, risolti, definiscono un valore numerico risultante. Il QCL mette a disposizione l'operatore di assegnamento "=" nella forma generale:

variabile = espressione
Non sono supportate le assegnazioni multiple tipo *variabile = variabile = espressione*
Per quanto riguarda l'istruzione di assegnamento, la regola di conversione è semplice: il QCL converte il valore alla destra del segno uguale nel tipo del dato sinistro.

Esempio:
Consideriamo le seguenti variabili:

```
SYSTEM
  sfFlag   F
  sbByte   B
  swWord   W
  slLong   L
  ssSingle S
```

sbByte = swWord
Viene eliminato il primo byte della variabile swWord assegnando a sbByte solo il byte meno significativo. Se il valore di swWord è compreso tra 127 e -128 i due valori risultano uguali e non viene eseguito alcun troncamento. Se il valore di swWord è esterno ai valori precedentemente considerati il valore di sbByte riflette solamente il valore del byte meno significativo di swWord.

ssSingle = slLong
Il valore di slLong viene convertito nel formato reale a singola precisione.

sfFlag = ssSingle
La variabile sfFlag viene assegnata al valore 1 se ssSingle rappresenta un valore diverso da zero.

17.2 Promozione di tipo

Quando in una espressione sono utilizzati dati di tipo differente, il compilatore QCL li converte tutti nello stesso tipo e, in particolare, nel tipo della dimensione che occupa più memoria, secondo quella che nei linguaggi viene definita *promozione di tipo*. Dopo che il compilatore ha applicato queste regole di conversione ogni coppia di operandi risulta della stessa dimensione che sarà anche la dimensione del risultato.

Ad esempio:

```
Variabile = (sbByte*sfFlag) + (swWord / sbByte) - (ssSingle+sfFlag)
```

Prima il compilatore converte sfFlag in BYTE e calcola il valore della moltiplicazione, poi il secondo sbByte in WORD e calcola il valore della divisione, poi sfFlag in SINGLE e calcola il valore della somma; il risultato di sbByte*sfFlag viene convertito in Word e ne calcola il valore. Quindi, questo risultato viene promosso a SINGLE per eseguire la sottrazione con il risultato di ssSingle+sfFlag.

Per quanto riguarda le costanti nelle espressioni, vengono convertite sempre nel tipo intero (FLAG, BYTE, WORD o LONG) della dimensione più indicata per contenere il valore (se nella costante non vi è il punto decimale). Se la costante contiene il punto decimale, viene convertita in tipo SINGLE.

Esempio:

```
  Variabile = swWord / sbByte
```

con swWord = 5 e sbByte = 2
Variabile risulta di tipo WORD con valore 2 perdendo così la parte decimale.

Se si riscrive l'espressione come:

```
  Variabile = (swWord * 1.0) / sbByte
```

con swWord = 5 e sbByte = 2
Variabile risulta di tipo SINGLE con valore 2,5. Questo perchè il prodotto della variabile swWord con una costante con parte decimale provoca la conversione del risultato in SINGLE.

18. Appendice C: Convenzioni di scrittura

Consigliamo i programmatori di seguire delle semplici convenzioni di scrittura del codice. Tali convenzioni non sono obbligatorie per il funzionamento, ma aiutano

- la leggibilità del codice scritto sia dallo stesso programmatore che da un secondo;
- a ridurre la possibilità di errori nella programmazione.

Le regole sono:

1. usare simboli in lingua inglese;
2. usare i caratteri maiuscoli per i nomi delle costanti e delle subroutine. Per separare le parole si può usare il carattere '_'.
3. Tutte le variabili locali (non visibili all'esterno della unit) devono avere un nome tutto minuscolo. Per separare le parole si può usare il carattere '_' (Esempio: *check_config_result*).
4. Le variabili locali che invece hanno visibilità all'esterno (IN, OUT o INOUT) hanno tutti i caratteri di cambio parola in maiuscolo. Non si usa in queste il carattere '_' (Esempio: *AccelerationTime*).

19. Appendice D: Parole chiave

In questo capitolo vengono riassunte le parole chiave del QCL.

ABS	valore assoluto
ACOS	arcocoseno
AND	AND logico
ANDB	And logico bit a bit
APPLICATION	Radice implicita di ogni simbolo utilizzato
ARRGBL	sezione file di configurazione
ARRSYS	sezione file di configurazione
ASIN	arcoseno
ATAN	arcotangente
B	byte
BEGIN	sezione codice operativo nella unit
BREAK	break
BUS	sezione file di configurazione
CALL	chiamata subroutine
CASE	istruzione SWITCH-CASE
CEIL	arrotondamento a intero non inferiore a quello dato
CONST	sezione file di configurazione
COS	coseno
COT	cotangente
D	double
DATAGROUP	sezione file di configurazione
DATAPROGRAM	sezione file di configurazione
DEVGROUP	inizio raggruppamento device
ELSE	“altrimenti” nell'istruzione IF
END	fine task
ENDDEVGROUP	fine raggruppamento device
ENDIF	fine istruzione IF
ENDSUB	fine subroutine
ENDSWITCH	fine SWITCH
ENDWHILE	fine while
EQ	uguale
EXP	esponenziale
EXTDEVICE	sezione file di configurazione
F	flag
FLOOR	arrotondamento a intero non superiore a quello dato
FOR	istruzione FOR
FPROG	istruzione FPROG
FSTEP	istruzione FSTEP
GE	maggiore uguale
GLOBAL	sezione file di configurazione
GT	maggiore
IF	istruzione IF
INPUT	sezione file di configurazione
INTDEVICE	sezione file di configurazione
ISFINITE	controlla se il valore è finito
ISINF	controlla se il valore è infinito
ISNAN	controlla se il valore è Nan (Not a Number)
ISNORMAL	controlla se il valore è normale
JUMP	istruzione JUMP
LE	minore o uguale
LN	logaritmo naturale
LT	minore
MULDIV	moltiplicazione e divisione
NEG	negazione (inversione del segno o complemento a 2)
NEQ	operatore
NEXT	istruzione NEXT
NOP	istruzione NOP
NOT	negazione
NOTB	negazione bit a bit (complemento a 1)
OR	OR logico
ORB	OR logico bit a bit

OUTPUT	sezione file di configurazione
POW	potenza
REFERENCE	proprietà di “riferimento” di un simbolo
REFERENCES	inizio dell'elenco dei riferimenti
RESOUT	reset uscite
RESTART	istruzione di restart
RESUME	istruzione di resume
RETURN	istruzione di return (su subroutine)
RMULDIV	resto di moltiplicazione e divisione
ROUND	arrotondamento a intero più vicino
S	single
SETOUT	setta uscita
SHLL	shift a sinistra
SHLR	shift a destra
SIN	seno
SQRT	radice quadrata
STEP	sezione file di configurazione
SUB	subroutine
SUSPEND	istruzione di sospensione
SWITCH	istruzione SWITCH-CASE
SYSTEM	sezione file di configurazione
TAN	tangente
TIMER	sezione file di configurazione
TRUNC	arrotondamento a intero non maggiore di grandezza
W	word
WAIT	istruzione di wait
WHILE	istruzione di while
XORB	OR esclusivo bit a bit

Oltre a queste vanno considerate le direttive del precompilatore

#DEFINE
#UNDEF
#IFDEF
#IFNDEF
#ELSE
#ENDIF
#ERROR

20. Appendice E: Tasti caldi

F1	Contenuti (Contents)
F2	-
F3	Prossima ricerca (Find Next)
F4	Prossima unità (Next Unit)
F5	Run
F6	Stop
F7	Restart
F8	Step
F9	Toggle breakpoint
F11	Vai a PC (Go to PC)
F12	Prossima unità selezionata (Next selected unit)
SHIFT + F2	Functions Info
SHIFT + F4	Unità precedente (Previous Unit)
SHIFT + F5	Move Rows Up (Editor LADDER)
SHIFT + F6	Move Rows Down (Editor LADDER)
SHIFT + F8	Step Over
SHIFT + F9	Clear All
SHIFT + F12	Unità selezionata precedente (Previous selected unit)
CTRL + A	Redo
CTRL + C	Copia
CTRL + E	Proprietà dell'elemento LADDER (Element Properties...)
CTRL + F	Trova (Find)
CTRL + G	Vai a (Go to)
CTRL + K	Compila (Compile)
CTRL + L	Download
CTRL + N	Salva progetto con altro nome (Save project As...)
CTRL + P	Stampa (Print)
CTRL + R	Sostituisci (Replace)
CTRL + S	Salva progetto (Save project)
CTRL + T	Ladder Network Checking
CTRL + V	Incolla
CTRL + X	Taglia
CTRL + Z	Undo
CTRL + F1	Ladder Function Info
CTRL + F2	Functions Info
CTRL + F3	Risultati della compilazione (View compilation results)

21. Appendice F: Files generati

21.1 File *.qm6: file di progetto

Viene creato da QVIEW dopo l'esecuzione del comando *New Project*. È un file unico per ogni applicazione.

21.2 File derivante dalla compilazione

21.2.1 File *.sym

E’ un file simboli utilizzato per la gestione dell'interfaccia operatore (terminale). Serve per l'applicazione realizzata per l'interfaccia operatore. Contiene tutte le informazioni per poter accedere alle variabili dichiarate nel progetto.

21.2.2 File *.xml

E’ un file simboli utilizzato per la gestione avanzata degli stessi da parte dei tools della QNET. Contiene tutte le informazioni per poter accedere alle variabili dichiarate nel progetto.

21.2.3 File *.bin

E’ un file risultato della compilazione che può essere utilizzato per essere trasferito nella CPU senza utilizzare la comunicazione seriale (per esempio con Multi Media Card).

22. Appendice G: Compatibilità con le versioni precedenti

L’ambiente di sviluppo è pienamente compatibile con le versioni e release precedenti.
Un progetto realizzato con Qview3, Qview 4 e Qview 5 potrà essere aperto con il Qview 6.

Al momento dell'apertura di un progetto realizzato con Qview precedenti, il Qview 6 applica delle trasformazioni automatiche di alcune parti di codice per adattarle alla nuova notazione. Queste trasformazioni sono:

Tipo di accesso	Versioni precedenti	Qview 6
Comandi ai device	<command name> <device name>	<device name>.<command name>
Parametri dei device	<device name>:<parameter/state>	<device name>.<parameter/state>
Variabili di sistema	QMOVE:<system variable name>	QMOVE.<system variable name>
Controlli schedulazione unit	T_RESTART <unit name>	<unit name>.RESTART
	T_RESUME <unit name>	<unit name>.RESUME
	T_SUSPEND <unit name>	<unit name>.SUSPEND
Stato di sospensione unit	QMOVE:is_suspend <unit name>	<unit name>.is_suspended
Dimensione array	<array name>:dim	<array name>.dim
Tempo rimanente timer	<timer name>:remain	<timer name>.remain

Al momento della compilazione di un progetto realizzato con Qview precedenti, è possibile che vengano notificati degli errori da Qview 6 che con i Qview precedenti non erano stati rilevati. Li elenchiamo di seguito:

- Non è possibile utilizzare lo stesso nomi per una *label* e per una *subroutine* o per una costante.
- Non è possibile scrivere le *subroutine* prima della istruzione *END* di una unit. Le *SUB* vanno sempre scritte alla fine della unit e dopo l'*END*.