Terminal Automate

SUPERVISOR
Guide d'utilisation

SERAD SA
271, route des crêtes
44440 TEILLE – France
☎ +33 (0)2 40 97 24 54
📠 +33 (0)2 40 97 27 04
💻 http://www.serad.fr
✉ info@serad.fr

SOMMAIRE

1- INTRODUCTION

1-1- Description

A) Généralités

Le système de supervision S640 ou S80 est un terminal intelligent, capable de gérer complètement les automatismes d'une machine.

Grâce à ses ports de communication, il peut dialoguer par liaison série ou bus de terrain avec les éléments d'automatismes tels que variateurs intelligents, entrées/sorties déportées, PC, automate, etc.

Facile à programmer grâce à l'atelier logiciel SPL, il possède un véritable cœur multi-tâches, de la mémoire RAM et FLASH, une horloge temps réel, plusieurs ports série (jusqu'à 3, RS232 ; RS485 ; CANopen).

Le SUPERVISOR est un système ouvert adapté à toutes les applications intégrant une interface homme/machine, un automate et de la communication.

B) Performances

- Processeur 32 bits à 33 MHz
- 4Mbits de RAM sauvegardée
- 8Mbits de mémoire Flash
- 2 ports séries - 1200 à 9600 bauds
- Gestion des d'entrées/sorties
- Horloge temps réel
- Chien de garde
- Afficheur rétro-éclairé
- 8 polices de caractères (S640 seulement)
- Clavier à effet tactile

C) Modularité

Le Supervisor offre un important choix de modules pour lui permettre de s'adapter à de nombreuses applications:

- Modules d'entrées/sorties TOR - 20 voies
- Carte de communication RS232, RS422 et RS485
- Carte de communication CANOpen

1-2- Description du logiciel SPL

1-2-1- Généralités

SPL est un logiciel de développement puissant sous environnement WINDOWS 95, 98, 2000, NT, ME et XT.

Il gère jusqu'à 28 tâches basic ou ladder, 20 000 variables utilisateurs et permet:

- Une configuration du système à l'aide de l'outil graphique
- Un accès facilité aux instructions évoluées par la boîte à outils
- Une programmation rapide à l'aide de l'outil Ladder
- Une aide en ligne et un éditeur pleine page
- Un mode de debug pour tester tout le système à partir du PC
- Un oscilloscope numérique qui affiche et mémorise 6 traces simultanées

2- INSTALLATION / MISE EN OEUVRE

2-1- Conditions d'utilisation

Le Supervisor doit être installée verticalement pour assurer un refroidissement naturel par convection.

Il doit être à l'abri de l'humidité, des projections de liquides quelconques, de la poussière.

Caractéristiques techniques :

- Alimentation 24VDC 15W
- Chien de garde : Contact NO libre de potentiel 48Vac maxi 2A maxi
- Température de service : 0 à 45°C
- Température de stockage : -20 à 70°C

2-2- Sécurités

- Les normes de sécurité imposent un réarmement manuel après un arrêt provoqué soit par coupure secteur, défaut chien de garde ou arrêt d'urgence.
- Le chien de garde du SUPERVISOR devra être relié en série dans la boucle d'arrêt d'urgence.
- Le chien de garde doit être activé au début du programme. Dans le cas d'un défaut (problème interne, micro-coupure, ...), la sortie chien de garde retombe.
- Relier l'information « Puissance armoire électrique OK » sur une entrée automate et la traiter dans une tâche basic non bloquante de sécurité ou dans une tâche ladder.

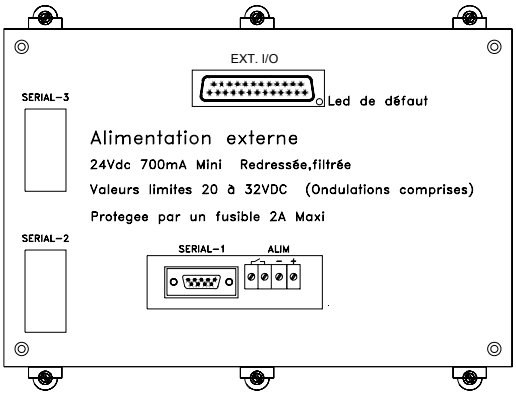
2-3- Raccordements

2-3-1- Explication générale

- Le câble de liaison SUPERVISOR / PC devra être blindé, la tresse étant reliée de chaque côté au chassis. Il devra être débranché du SUPERVISOR lorsqu'il n'est plus utilisé. Tous ces câbles, ainsi que les câbles d'entrées-sorties, devront être séparés et éloignés des circuits de puissance.
- Prévoir impérativement des diodes de roue libre sur les charges inductives en continu et des filtres RC sur les charges inductives en alternatif. Ces diodes et filtres doivent être placés le plus près possible de la charge. Les conducteurs d'alimentation et de signaux ne doivent pas être le siège de surtensions.

2-3-2- Supervisor

A) S640:



SERIAL-1 SUBD 9PTS MALE	
PIN	RS232
1	
2	RXD
3	TXD
4	
5	GND
6	
7	
8	
9	

NON ISOLEE

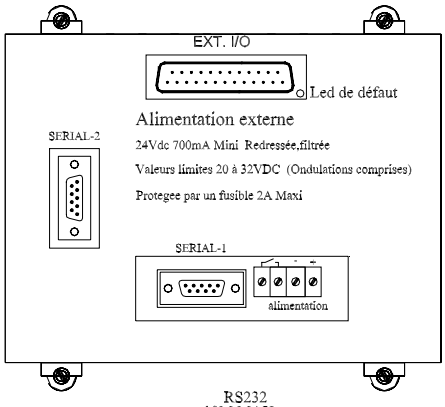
OPTION SERIAL-2 SUBD 9PTS FEMELLE		
PIN	RS422	RS485
1		
2		
3	RX−	
4	RX+	
5	GND	GND
6		
7	TX−	TRX−(B)
8	TX+	TRX+(A)
9		

OPTION SERIAL-3 SUBD 9PTS FEMELLE
CANBUS
GND
CANL
CANH

ISOLEE

IO SUBD 25PTS FEMELLE			
PIN		13	+24V
1	INP1	14	INP7
2	INP2	15	INP8
3	INP3	16	INP9
4	INP4	17	INP10
5	INP5	18	INP11
6	INP6	19	INP12
7	0V	20	OUT5
8	OUT1	21	OUT6
9	OUT2	22	OUT7
10	OUT3	23	OUT8
11	OUT4	24	0V
12	+24V	25	0V

B) S80:



SERIAL-1 SUBD 9PTS MALE	
PIN	RS232
1	
2	RXD
3	TXD
4	
5	GND
6	
7	
8	
9	

NON ISOLEE

OPTIONAL SERIAL-2 SUBD 9PTS FEMELLE		
PIN	RS422	RS485
1		
2		
3	RX-	
4	RX+	
5	GND	GND
6		
7	TX-	TRX-(B)
8	TX+	TRX+(A)
9		

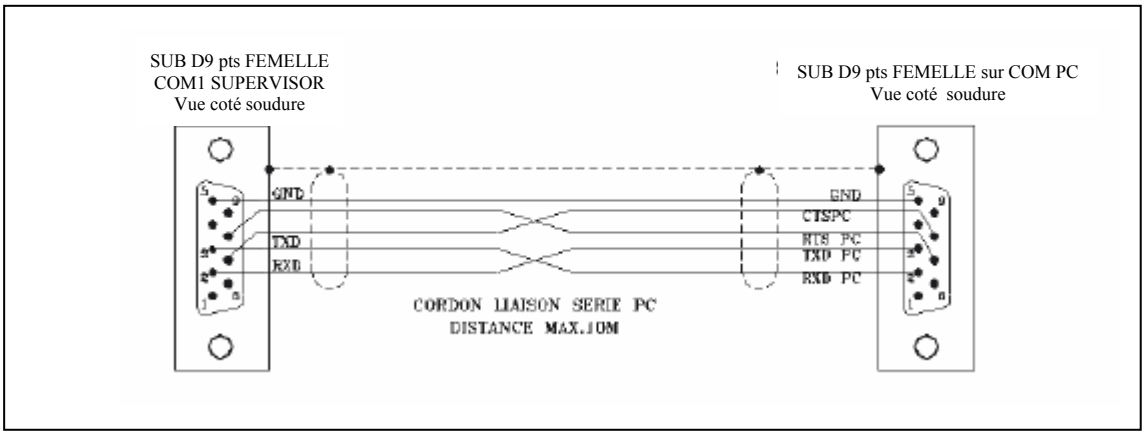
CANBUS
GND
CANL
CANH

ISOLE

IO SUBD 25PTS FEMELLE			
PIN		13	+24V
1	INP1	14	INP7
2	INP2	15	INP8
3	INP3	16	INP9
4	INP4	17	INP10
5	INP5	18	INP11
6	INP6	19	INP12
7	0V	20	OUT5
8	OUT1	21	OUT6
9	OUT2	22	OUT7
10	OUT3	23	OUT8
11	OUT4	24	0V
12	+24V	25	0V

WATCHDOG est un contact normalement ouvert libre de potentiel de 48 VACmaxi et de 2A maxi.

C) Connections PC <-> SUPERVISOR:



2-4- Mise en route

2-4-1- Mise en route

La mise en route du SUPERVISOR s'effectue de la façon suivante :

- On commence par définir l'emplacement des cartes avec l'écran de configuration du SPL.
- On choisit ensuite la configuration de chacune des cartes en cliquant dessus.
- On charge la configuration ainsi définie en utilisant la fonction "Envoyer la configuration" à partir du menu communication du SPL.
- On définit les données globales.
- On transfère la valeur initiale des variables sauvegardées en utilisant "Envoyer les données" à partir du menu communication du SPL.
- On écrit les différentes tâches.
- On transfère les tâches après les avoir compilées.

Par la suite chaque modification apportée aux tâches nécessite une nouvelle compilation et un transfert des tâches dans le SUPERVISOR. Une modification de la configuration nécessite un nouvel envoi de la configuration dans le SUPERVISOR.

- Il est fortement conseillé de sauvegarder les données dans la mémoire flash du SUPERVISOR à partir du menu de communication et de conserver une copie du projet sur disquette.

3- LOGICIEL SPL

3-1- Installation du logiciel SPL

3-1-1- Configuration du système

Configuration minimale :

- PC 486 DX2 66
- RAM 8 Mo
- Disque dur (35 Mo disponibles)
- Microsoft® Windows™ 95 ou Microsoft® Windows™NT 4.0 (service pack 3)
- Lecteur de CD-ROM (2X)
- Ecran SVGA
- Souris ou autre périphérique de pointage

Configuration recommandée :

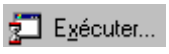
- PC Pentium® 75 ou plus
- RAM 16 Mo
- Disque dur (35 Mo disponibles)
- Microsoft® Windows™ 95 ou Microsoft® Windows™NT 4.0 (service pack 3)
- Lecteur de CD-ROM (4X)
- Ecran SVGA
- Souris ou autre périphérique de pointage

Ce logiciel peut aussi fonctionner sous Microsoft® Windows NT™. Cette application ne travaille pas sous Unix, Mac, MS-DOS et Microsoft® Windows 3.11.

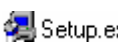
3-1-2- Procédure d'installation du logiciel SPL

Le logiciel Supervisor Programming Language est fourni de CD-ROM. L'installation du logiciel se fait comme suit :

- Vérifier la configuration requise pour installer le logiciel
- Insérer le CD-ROM dans le lecteur approprié.
- Dans le menu déroulant 

, sélectionner 

.
- Dans la boîte de dialogue « Exécuter », sélectionner 

.
- Dans la boîte de dialogue « Parcourir », sélectionner le lecteur où se situe le CD-ROM.
- Sélectionner 

puis 

dans la boîte de dialogue « Parcourir ».
- Sélectionner 

dans la boîte de dialogue « Exécuter ».
- ⇨ Le programme d'installation du logiciel SPL débute.
- Le début de l'installation est marquée par une série de boîte de dialogue guidant l'utilisateur :

- répertoire de destination
- type d'installation (Typique, compacte ou personnalisée)
- sélection du dossier programme

' Attention : seul un niveau de répertoire peut-être crée.

⇨ **L'installation des fichiers débute et est indiquée par l'évolution d'un barre graphe.**

⇨ **L'installation se termine par l'ajout de l'icône du SPL dans le dossier programme.**

3-1-3- Mise à jour d'une version antérieure

Un programme décrit avec une version antérieure peut fonctionner avec la nouvelle version à condition de recompiler les tâches. Le logiciel SPL ne fonctionne qu'avec le système d'exploitation fourni dans le répertoire OS de l'arborescence d'installation du logiciel. Ce répertoire se situe dans l'arborescence spécifiée lors de l'installation. Par défaut celui-ci est « **C:\Program Files\Serad\Spl**».

La procédure d'installation du système d'exploitation est la suivante :

⇩ Relier le port SERIAL1 du SUPERVISOR sur COM1 ou COM2 du PC

⇩ Lancer le logiciel SPL, aller dans le menu Projet -> Nouveau Projet puis dans le menu Options -> Système d'exploitation -> Mise à jour (si vous préférez faire la mise à jour sous DOS, suivez les intructions suivantes :)

⇩ Sous Windows 95 ou plus, il faut ouvrir une fenêtre DOS

⇩ Se placer sous le répertoire OS à l'aide des commandes DOS

⇩ Exécuter la commande : INSTALL < Port série du PC>

Pour un câble de liaison sur le COM1 : INSTALL COM1

⇨ L'installation commence par effacer l'ancien système d'exploitation. Le message « Waiting for erasure » apparaît sur le PC ainsi que sur le SUPERVISOR.

⇨ Ensuite, la programmation.s'effectue.

⇨ Lorsque la programmation est terminée, le SUPERVISOR redémarre en indiquant erreur 23 sur No user tasks à l'écran car aucun programme utilisateur n'est présent.

⇩ Recompiler les tâches du projet puis les transférer dans le SUPERVISOR.

3-2- Architecture du logiciel SPL

3-2-1- Les répertoires

⇩ Gfx: contient tous les graphiques.

⇩ Lib : contient tous les fichiers avec extensions DLL nécessaire au bon fonctionnement du logiciel

⇩ Str : contient les fichiers concernant les différentes langue que gère le logiciel

⇩ OS : contient une copie du système d'exploitation du SUPERVISOR

⇩ Help : contient les fichiers d'aide pour le SUPERVISOR et le SPL.

⇩ Project : contient les différents fichiers et répertoires des projets développés par l'utilisateur

3-2-2- Contenu d'un projet

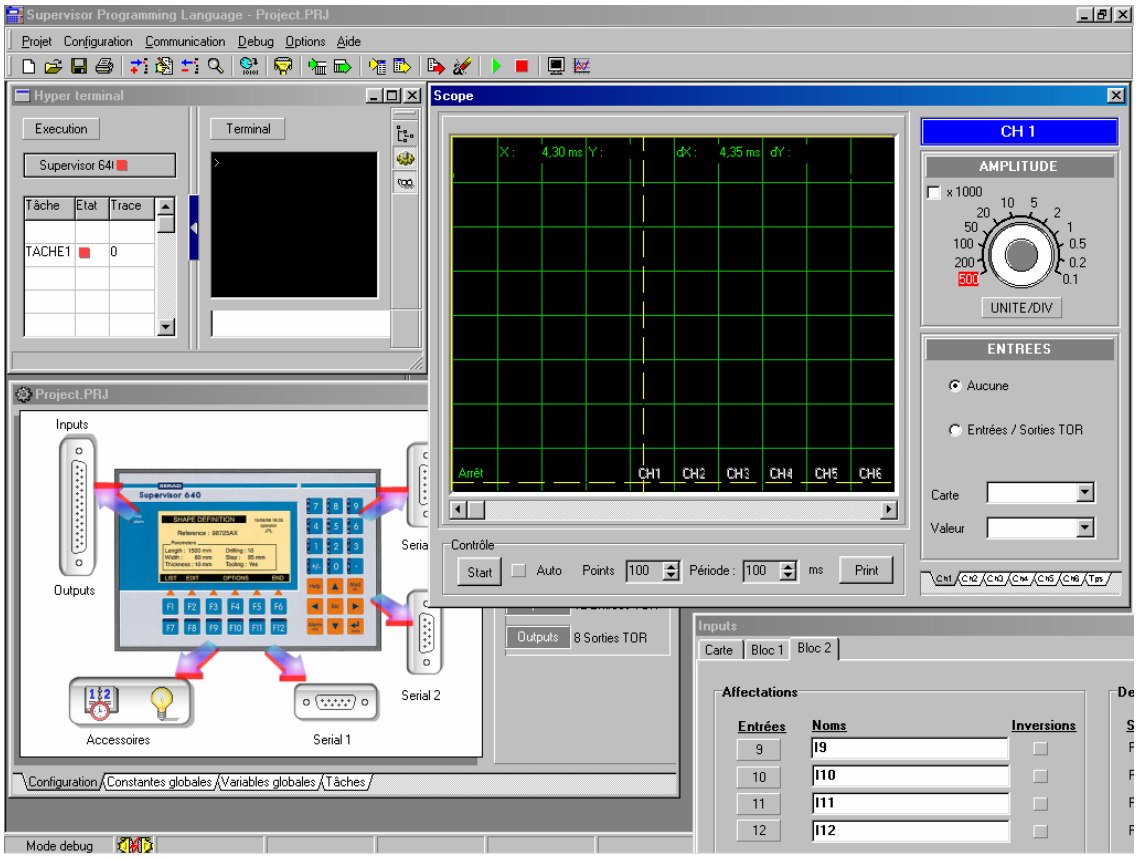
Le répertoire “project” est l’emplacement réservé par défaut pour stocker les projets utilisateurs. Chaque projet développé se décompose en un fichier “Nom Projet.prj” et un répertoire “Nom Projet.rep”. Ce répertoire comporte les fichiers suivants:

- un fichier de configuration (Nom Projet.cfg)
- un fichier de définition des variables globales (Nom Projet.var)
- un fichier de définition des constantes globales (Nom Projet.cst)
- un fichier par tâche (Nom Tâche.tsk)
- un fichier supplémentaire par tâche ladder (Nom Projet.lad)
- le résultat de la compilation donne des fichiers binaires (Nom Projet.bin et Nom Projet.b00 à Nom Projet.b007). La somme des fichiers de type b0* permet de connaître la taille du projet compilé.
- autres fichiers (.map, .uti) pour la gestion interne du SPL

3-3- Présentation

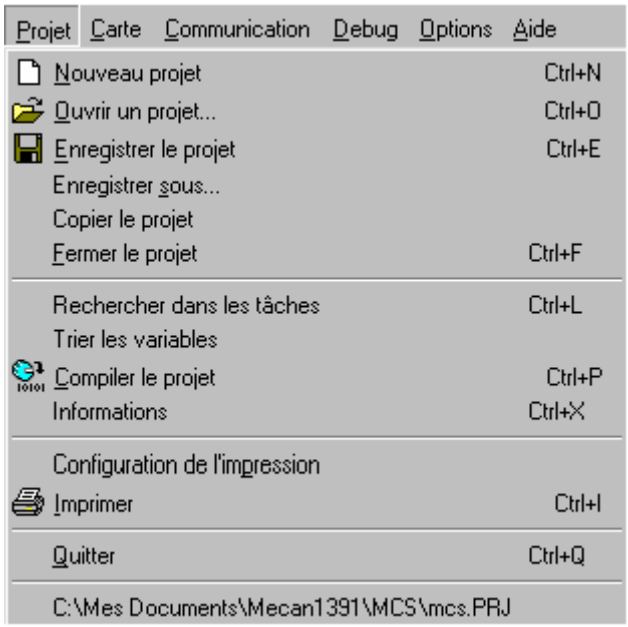
3-3-1- Ecran initial

Le logiciel SPL est caractérisé par une fenêtre principale comportant le menu principal, une barre d’icônes et le multi-fenêtrage. Les propriétés du multi-fenêtrage permettent à l’utilisateur de pouvoir passer d’une fenêtre à une autre sans avoir à la modifier.



3-4- Menus et icônes

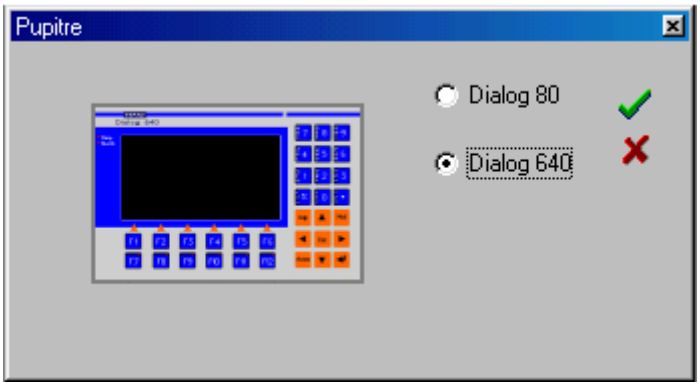
3-4-1- Menu Projet



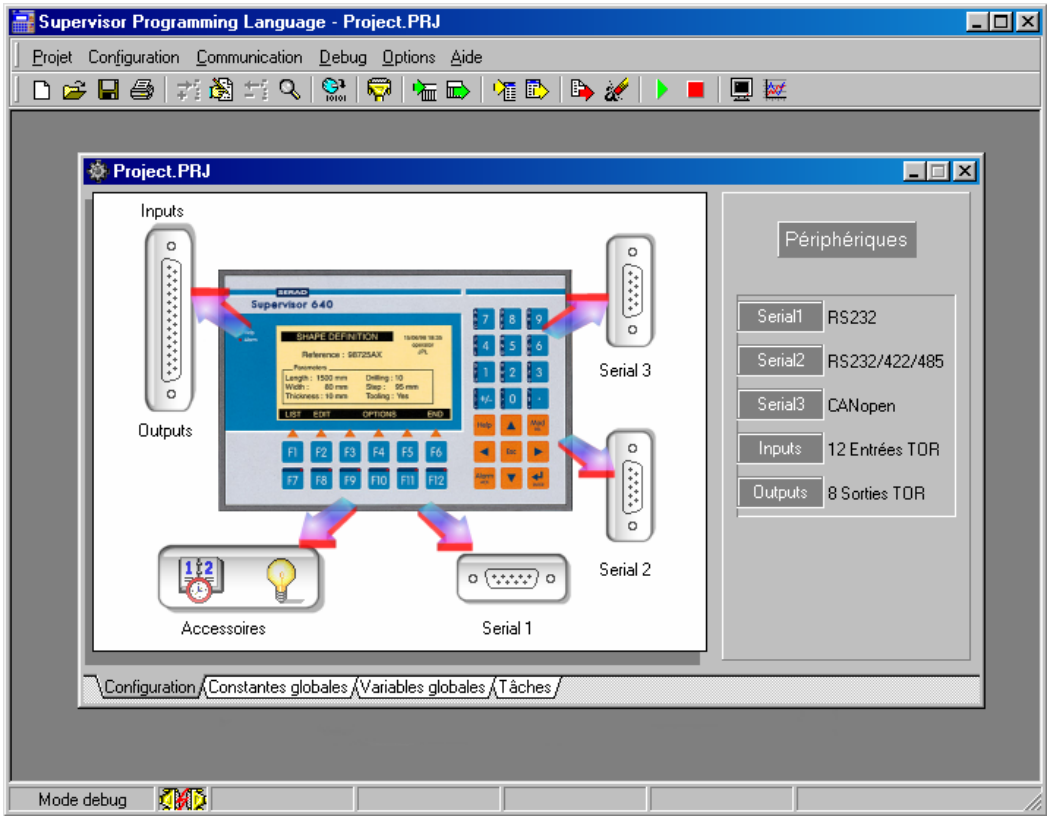
Nouveau Projet

Icône: 

Action: Cette commande permet à l'utilisateur de définir un nouveau projet. Le dernier projet en cours se trouve alors fermé et une fenêtre s'affiche pour choisir le modèle de Supervisor :



La fiche de configuration est la seule à être présente et paraît de façon vierge (comme ci-dessous).



Ouvrir un projet

Icône: 

Action: Cette commande ouvre la boîte de dialogue “Ouvrir un Projet”. elle permet à l’utilisateur de spécifier le chemin et le nom du projet à charger. Cette commande à pour effet de fermer le dernier projet en cours dès la validation du projet à ouvrir.

Enregistrer un projet

Icône: 

Action: Cette commande permet la sauvegarde complète du projet en cours sous le nom spécifié.

Enregistrer le projet sous…

Action: Cette commande ouvre la boîte de dialogue “Enregistrer sous” et permet à l’utilisateur de spécifier le nom du projet de sauvegarde. Cette commande a pour effet de créer un fichier et un répertoire portant le nom spécifié avec pour le premier l’extension “prj” et pour le second l’extension “rep”.

Copier le projet

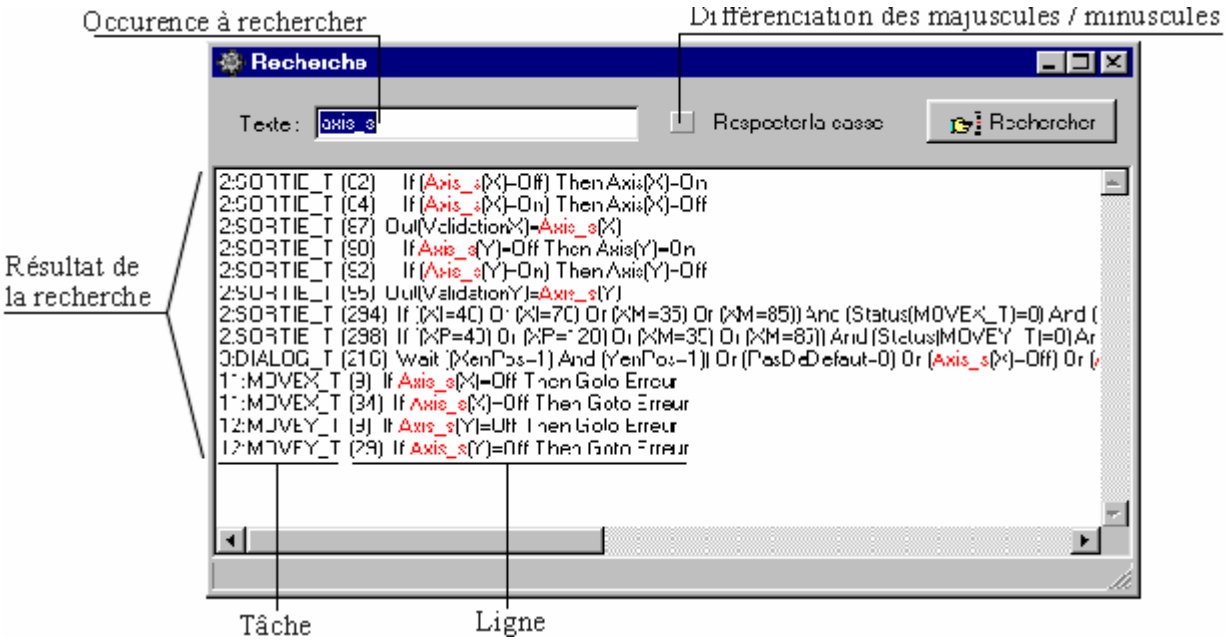
Action: Cette commande a les mêmes fonctionnalités que Enregistrer le projet sous… . L'intérêt de cette commande est qu'elle ne modifie pas les dates de création du projet.

Fermer le projet

Action: Cette commande ferme le projet en cours.

Rechercher dans les tâches

Action: Cette commande permet de rechercher un texte, un mot ou une partie d'un mot dans toutes les tâches du projet. Une boîte de dialogue est liée à cette commande. Elle regroupe toutes les fonctions nécessaires pour déterminer, lancer et visualiser la recherche. Un double clique sur une des lignes de la fenêtre résultat lance l'éditeur de tâche et vient se placer sur la ligne de la tâche concernée.



Trier les variables

Action : Cette commande trie les variables globales. On trouve d'abord les variables globales sauvegardées triées dans l'ordre croissant de leur adresse. Ensuite viennent les variables globales non sauvegardées triées par type (bit, octet … chaîne de caractères). A l'intérieur d'un même type, un tri par ordre alphabétique est effectué.

Compiler le projet

Icône:

Action: Cette commande réalise la compilation du projet. Une phase d'analyse des tâches permet de vérifier la syntaxe de chaque tâche, la définition de la configuration des variables, etc…. Lors d'une erreur de syntaxe, la tâche concernée est éditée et l'erreur est mise en évidence par la position du curseur.

Il est possible de compiler une seule tâche. Pour cela, pointer la tâche concernée à partir de la liste des tâches puis sélectionner « Vérifier la syntaxe » par le click droit de la souris.

Informations

Action : Cette commande permet d'avoir des informations détaillées sur le projet, sur la mémoire programme et mémoire de données utilisées

Configuration de l'impression

Action: Cette commande permet à l'utilisateur de définir sont type d'impression (imprimante, papier, etc…). L'orientation du papier peut-être modifiée mais n'est pas prise en compte lors de l'impression (impression de type portrait).

Impression

Icône: 

Action: Cette commande réalise une impression totale ou personnalisée du projet. Dans cette impression, on retrouve la configuration du SUPERVISOR, les différents programmes basic des tâches ainsi que le ladder correspondant.

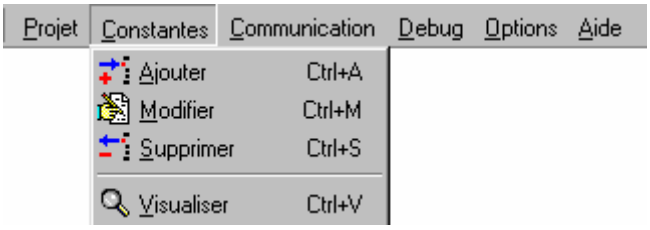
Quitter

Action: Cette commande permet à l'utilisateur de quitter le logiciel.

Liste de projets

Action : En cliquant sur l'un de ces éléments de la liste, le projet indiqué est ouvert.

3-4-2- Menu Constantes/Variables/Tâches



Les 4 commandes définies dans ce sous-menu agissent toutes sur la fenêtre principale du projet. Leur action est différente suivant l'onglet de cette fenêtre.

- Tout ajout, suppression ou modification d'éléments implique une recompilation du projet.
- Toute modification sur une valeur de paramètre d'une carte implique un envoi de configuration.
- Toute modification sur une valeur de variable globale sauvegardée implique un envoi de variables.

Ajouter

Icône: 

Action : Cette commande permet d'ajouter une carte, une constante globale, une variable globale ou une tâche suivant l'onglet sélectionné.

Modifier

Icône: 

Action : Cette commande permet de modifier les paramètres d'une carte, d'une constante globale, d'une variable globale ou d'une tâche suivant l'onglet sélectionné.

Supprimer

Icône: 

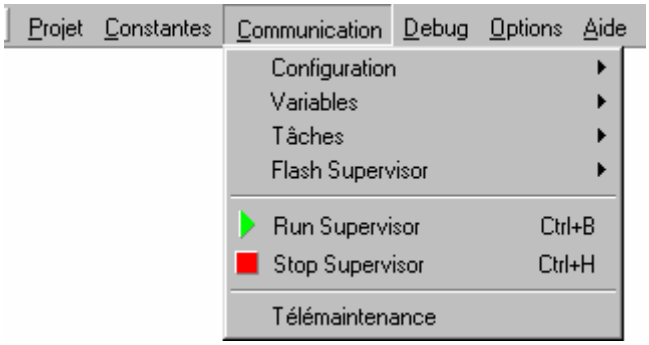
Action : Cette commande permet de supprimer une carte, une constante globale, une variable globale ou une tâche suivant l'onglet sélectionné.

Visualiser

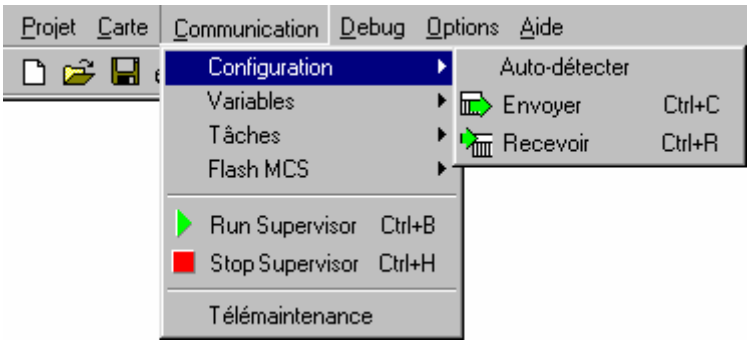
Icône: 

Action : Cette commande permet de visualiser les paramètres d'une carte, d'une constante globale, d'une variable globale ou d'une tâche suivant l'onglet sélectionné.

3-4-3- Menu Communication



Configuration



Auto-détecter

Action: Cette commande permet lors d'un nouveau projet de créer automatiquement la configuration si le PC est relié par le lien série au SUPERVISOR.

Envoyer la configuration

Icône: 

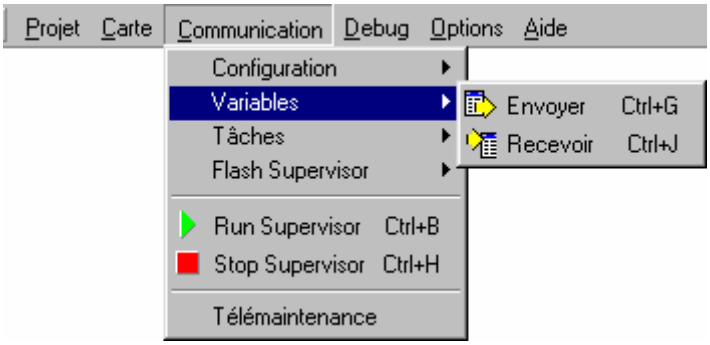
Action: L'envoi de la configuration permet d'initialiser dans le SUPERVISOR les paramètres définis dans les écrans de configuration des cartes. Le transfert commence par le test de la configuration du SUPERVISOR par rapport à la configuration indiquée sur le PC. Si une différence est détectée le transfert est stoppé et un message apparaît en indiquant le slot dont le contenu est incorrect. Elle est nécessaire suite à un ajout, suppression de cartes…

Recevoir la configuration

Icône: 

Action: La réception de la configuration permet de mettre à jour les paramètres indiqués dans les écrans de configuration. Le transfert commence par le test de la configuration de du SUPERVISOR par rapport à la configuration indiquée sur le PC. Si une différence est détectée le transfert est stoppé et un message apparaît en indiquant le slot dont le contenu est incorrect. Si l'on souhaite coserver cette configuration dans le projet, il est nécessaire d'effectuer la commande «Enregistrer le projet sous».

Variables



Envoyer les variables

Icône: 

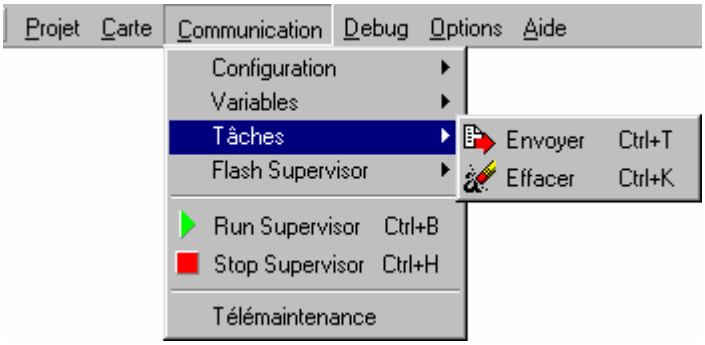
Action: L'envoi des variables sauvegardées permet d'initialiser dans le SUPERVISOR, les valeurs affectées à ces variables. Ainsi il n'est pas nécessaire de les initialiser dans le programme. Le transfert commence par le test de la configuration du SUPERVISOR par rapport à la configuration indiquée sur le PC. Si une différence est détectée le transfert est stoppé et un message apparaît en indiquant le slot dont le contenu est incorrect.

Recevoir les variables

Icône: 

Action: La réception des variables sauvegardées permet de conserver une copie de leur valeur sur le PC. Le transfert commence par le test de la configuration du SUPERVISOR par rapport à la configuration indiquée sur le PC. Si une différence est détectée le transfert est stoppé et un message apparaît en indiquant le slot dont le contenu est incorrect. Si l'on souhaite conserver cette configuration dans le projet, il est nécessaire d'effectuer la commande «Enregistrer le projet sous».

Tâches



Envoyer les tâches

Icône: 

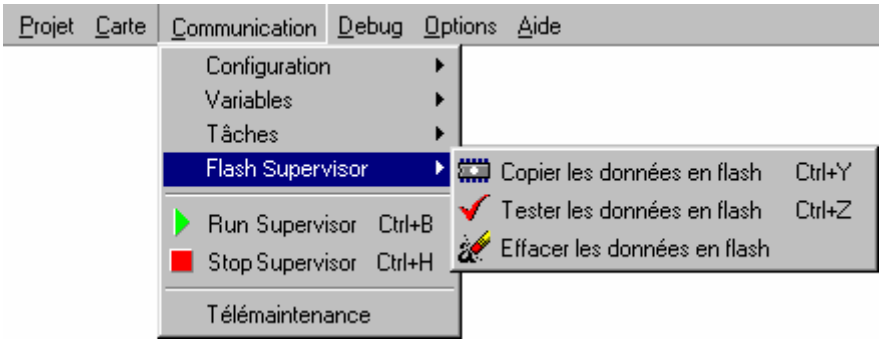
Action: Cette commande permet d'envoyer la totalité des tâches compilées dans le SUPERVISOR. L'exécution des tâches est lancée automatiquement dès la fin du transfert. Le transfert commence par le test de la configuration du SUPERVISOR par rapport à la configuration indiquée sur le PC.

Effacer les tâches

Icône: 

Action: Cette commande permet d'écraser les tâches présente dans le SUPERVISOR.

Flash Supervisor



Copier les données en flash

Action: Cette commande permet de créer une copie de sauvegarde en mémoire flash des paramètres de configuration et des 10000 premières variables sauvegardées stockées en mémoire RAM soutenue par pile. A chaque mise sous-tension du SUPERVISOR, un calcul de checksum est effectué pour tester la cohérence des données en RAM. Si une erreur est détecée, le

SUPERVISOR transfère la copie stockée en mémoire flash dans la mémoire RAM puis lance l'exécution des tâches. S'il n'y a pas de copie, le SUPERVISOR passe en erreur 20.

Tester les données en flash

Action: Cette commande indique si la mémoire flash contient une copie des données en RAM.

Effacer les données en flash

Action: Cette commande efface la copie des données contenues dans la mémoire flash.

Run Supervisor

Icône: 

Action: Cette commande lance l'exécution des tâches du SUPERVISOR.

Stop Supervisor

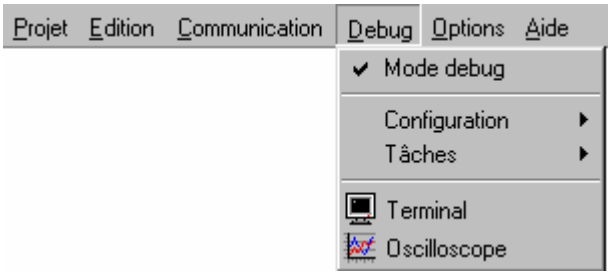
Icône: 

Action: Cette commande permet d'arrêter l'exécution des tâches du SUPERVISOR. Le WatchDog passe à l'état OFF. Toutes les cartes servo passent en boucle ouverte (consigne analogique=0).

Telemaintenance

Action: Avec cette commande, on accède au mode de télémaintenance. On peut alors piloter le SUPERVISOR à distance via modems et une ligne téléphonique (se reporter au chapitre Telemaintenance).

3-4-4- Menu Debug



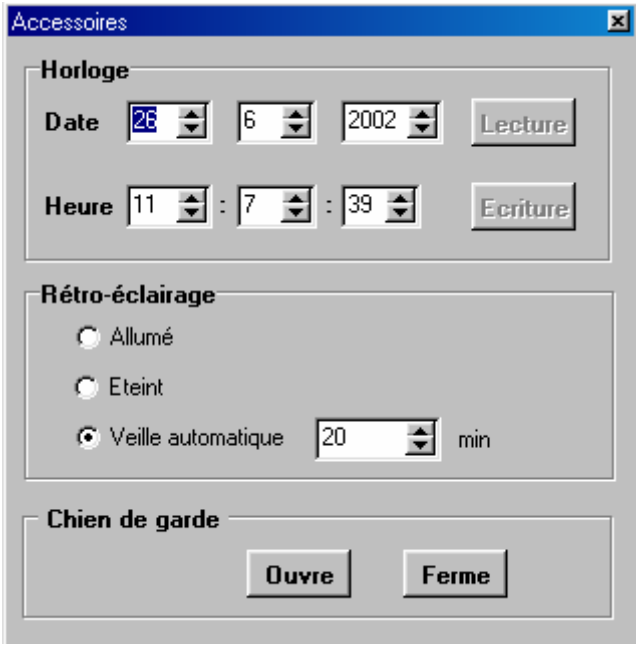
Mode Debug

Action: Cette commande autorise le fonctionnement en mode Debug. L'activation de cette commande a pour effet d'autoriser toutes les autres actions de ce menu.

Configuration

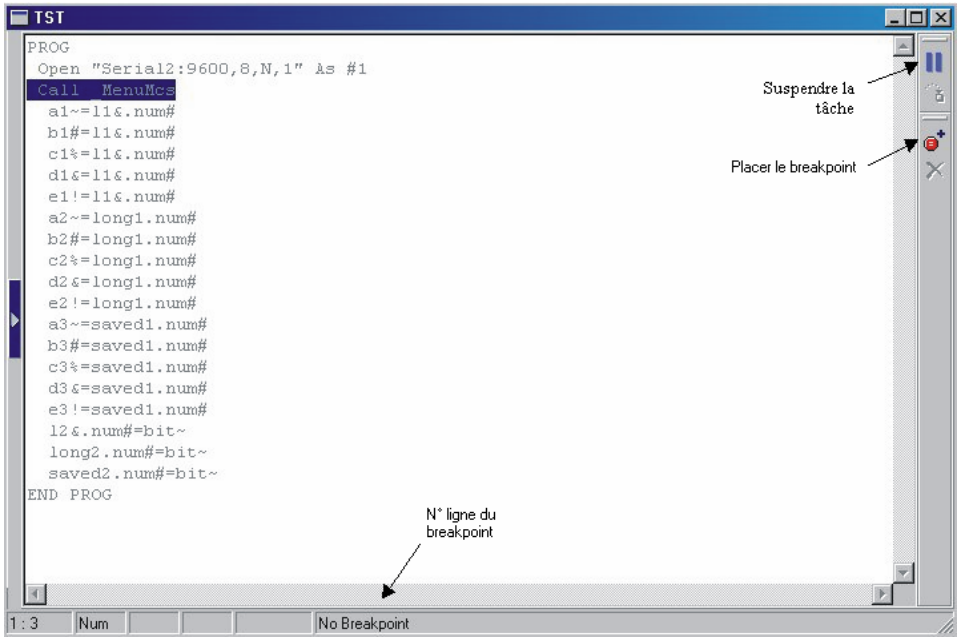
Action : Ce sous-menu permet d'afficher la fenêtre de debug de la SB32EX ou du slot qui a été validé. Suivant la carte du slot, la fenêtre est différente :

↳ Une boîte de dialogue comportant l'état de l'afficheur, du chien de garde et les informations de date et heure du SUPERVISOR apparaît. Ces différents paramètres peuvent-être modifiés. Il faut aussi noter que pendant une exécution des tâches du SUPERVISOR, si l'un de ces éléments est piloté, sa manipulation par l'utilisateur risque de ne pas être pris en compte ou de manière fugitive.



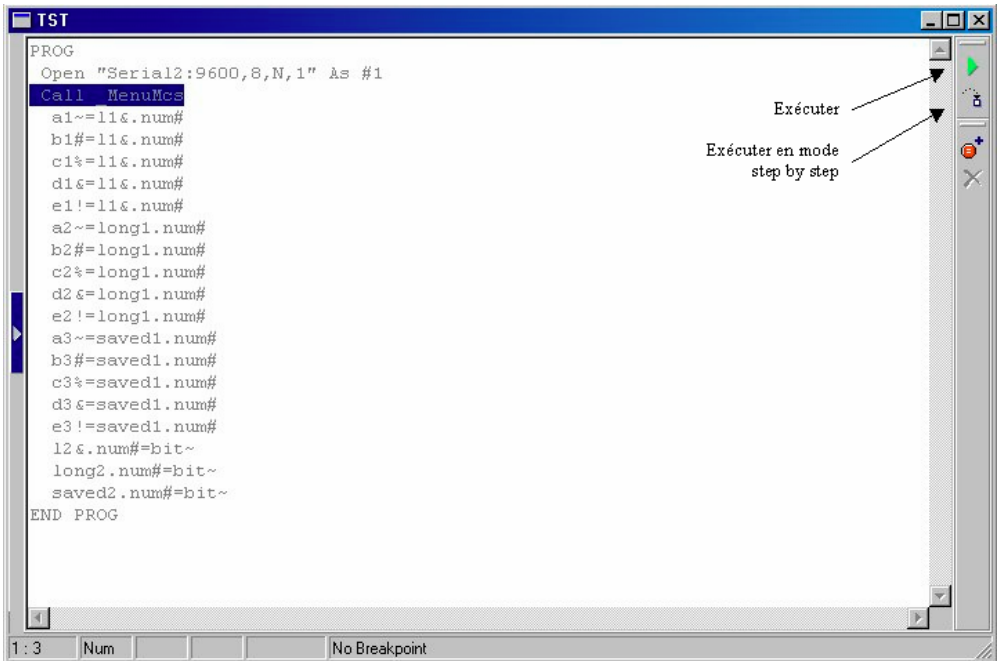
↳ La fenêtre de Debug du module entrées / sorties TOR permet de visualiser l'état des entrées ou des sorties de la carte par des leds. Un clique sur l'une des leds de visualisation permet de modifier l'état de l'entrée ou de la sortie.

Tâches



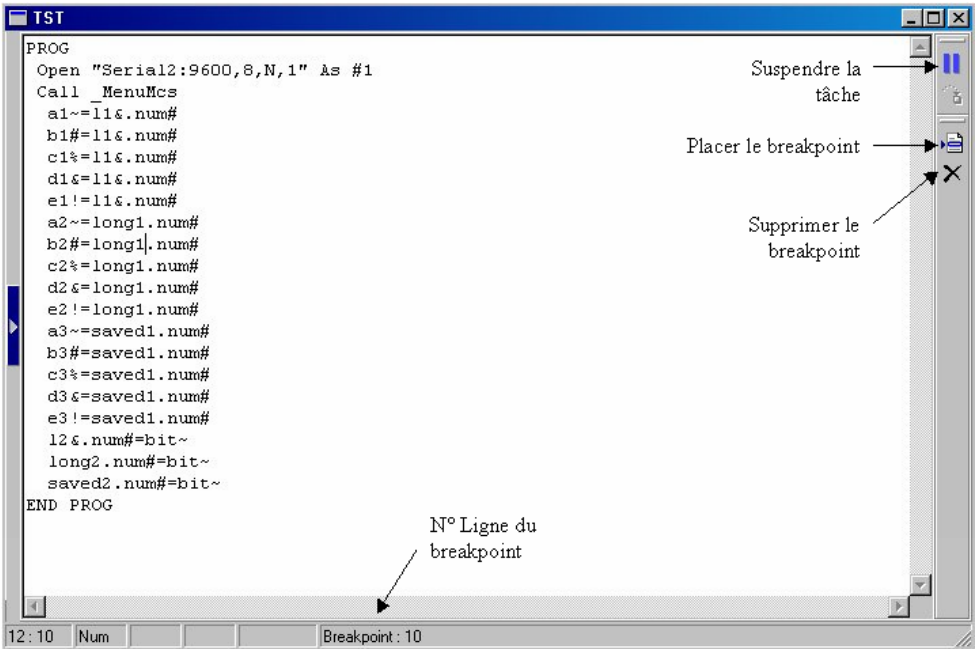
Action : Ce sous-menu regroupe tous les noms des tâches définies dans l'onglet Tâches. La validation d'une tâche ouvre l'éditeur basic en mode debug. C'est à dire que le code de la tâche apparaît mais ne peut pas être modifié. Ce mode permet entre autre de visualiser la trace d'évolution si elle a été validée (se référer au menu option->projet->compilateur).

Stepbystep :



Action : Commande de déboggage qui permet d'exécuter le programme en pas à pas ce qui permet de controler le bon fonctionnement de chaque ligne basic écrite dans la tâche.

Breakpoint :

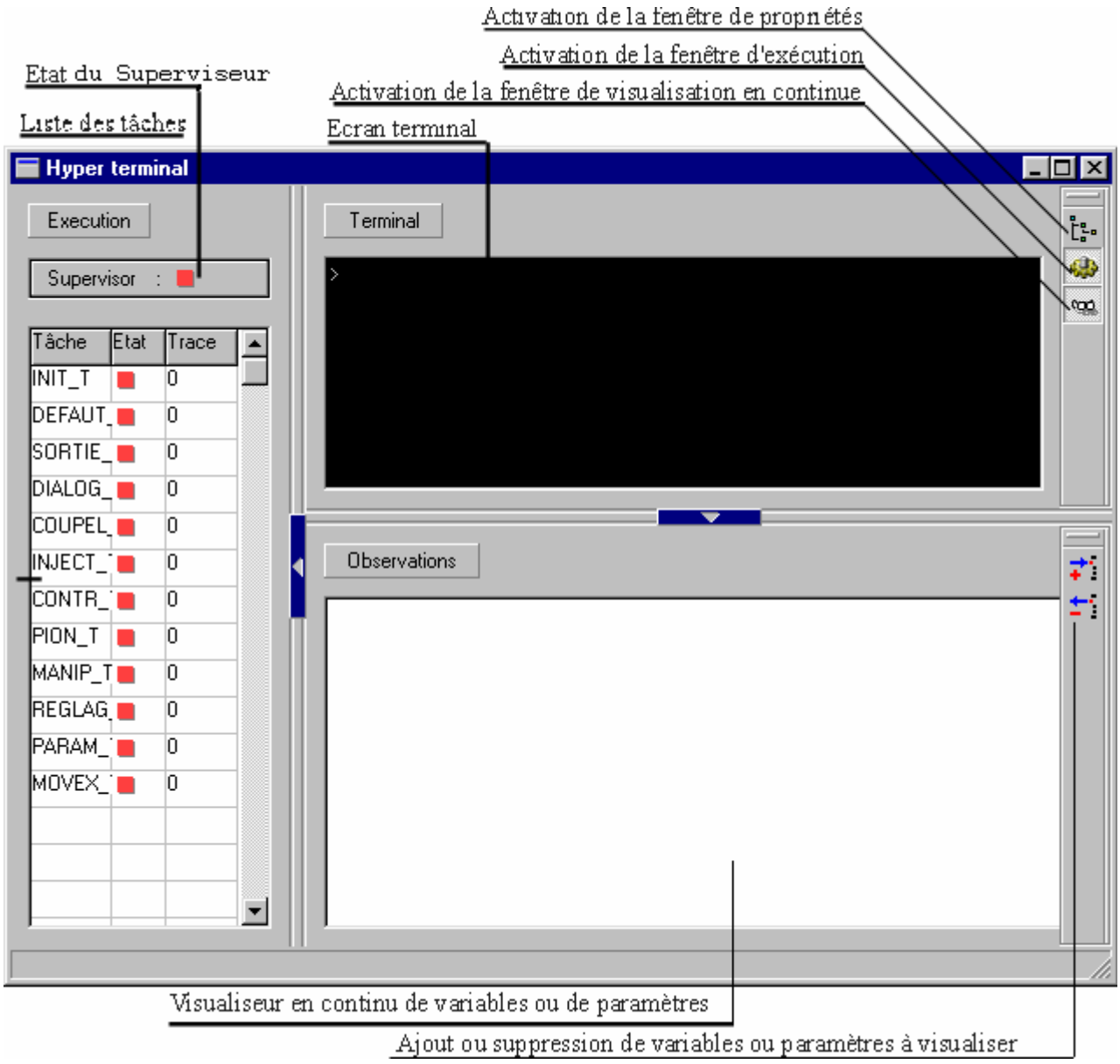


Action : Permet de choisir une ligne dans la tâche où vous voulez que le programme s'arrête lors du mode debuggage. L'exécution du programme est interrompue et vous pouvez alors examiner l'état de votre programme à ce stade de l'exécution (états des variables, tempos …).

Terminal

Icône: 

Action: Cette commande ouvre l'hyper terminal. Cet outil d'aide à la mise en œuvre permet d'interroger l'état du SUPERVISOR, de visualiser et de modifier les variables locales ou globales, les paramètres, les entrées et les sorties.



La fenêtre Terminal est composée de l'écran principal et de deux autres fenêtres optionnelles : la fenêtre « observations » et la fenêtre « status ».

⇨ L'écran principal de la fenêtre Terminal permet la lecture et l'écriture de toutes les variables et paramètres en temps réel dans le SUPERVISOR. Pour accéder à ces différentes informations, le terminal est muni de fonctions :

↳ Print <Nom de variable ou paramètre> : affichage d'une variable sauvegardée ou d'un paramètre.

↳ <Nom de variable ou paramètre>=<Valeur> : affectation d'une valeur à une variable sauvegardée ou un paramètre

↳ STATUS : état des tâches

↳ RUN <Nom d'une tâche> : exécution d'une tâche

↳ HALT <Nom d'une tâche> : arrêt d'une tâche

↳ SUSPEND <Nom d'une tâche> : suspend l'exécution d'une tâche

↳ CONTINUE <Nom d'une tâche> : continue l'exécution d'une tâche

↳ CLS : efface la zone de dialogue

↳ RESTART : redémarre le SUPERVISOR

↳ EXIT : ferme le terminal

Pour faciliter l'édition des variables ou paramètres, un éditeur des propriétés de la configuration du SUPERVISOR peut-être mis à disposition. Cette fenêtre regroupe les différentes cartes configurées et leurs paramètres associées, les différentes variables globales et les tâches et leurs variables locales. En double cliquant sur une variable ou un des paramètres de cette fenêtre, le nom associé apparaît alors dans l'écran du terminal.

⇨ La fenêtre « observations » permet la visualisation de paramètres ou de variables en continue. Le nombre de variables ou paramètres à visualiser est limité à 40. Deux commandes permettent d'ajouter ou de supprimer une variable. L'ajout exécute la fenêtre de propriétés de la configuration du SUPERVISOR. La variable ou le paramètre doit être choisi parmi les cartes, variables globales ou tâches de la configuration. On peut sauver ou charger ces 40 variables sous forme de fichiers.

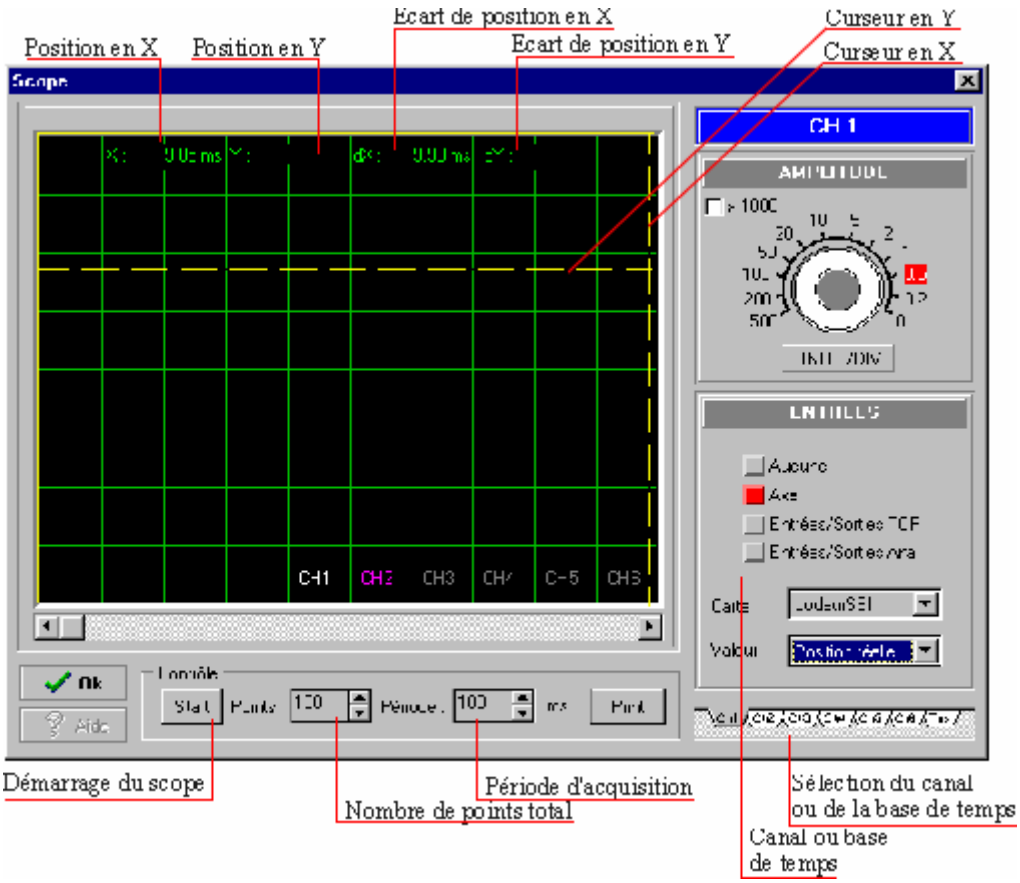
⇨ La fenêtre « status » du terminal permet la visualisation de l'état du SUPERVISOR et des tâches du projet. Le SUPERVISOR peut-être pilotée à distance en cliquant sur l'icône lecture ou stop qui est affiché. Le clique provoque le basculement d'un état à l'autre. De même les tâches peuvent être pilotées à distance, les états peuvent être « arrêt », « départ », « suspendue » ou « continue ». Le changement est obtenu en cliquant sur le bouton de droite et en préselectionnant la tâche à modifier. La colonne « trace » permet de connaître la ligne en cours d'exécution dans la tâche. Il faut au préalable que le code trace est été validé et que le projet soit recompilé et envoyé à le SUPERVISOR. On peut également avoir une notion des ressources prises au système pour chacune des tâches.

Oscilloscope

Icône: 

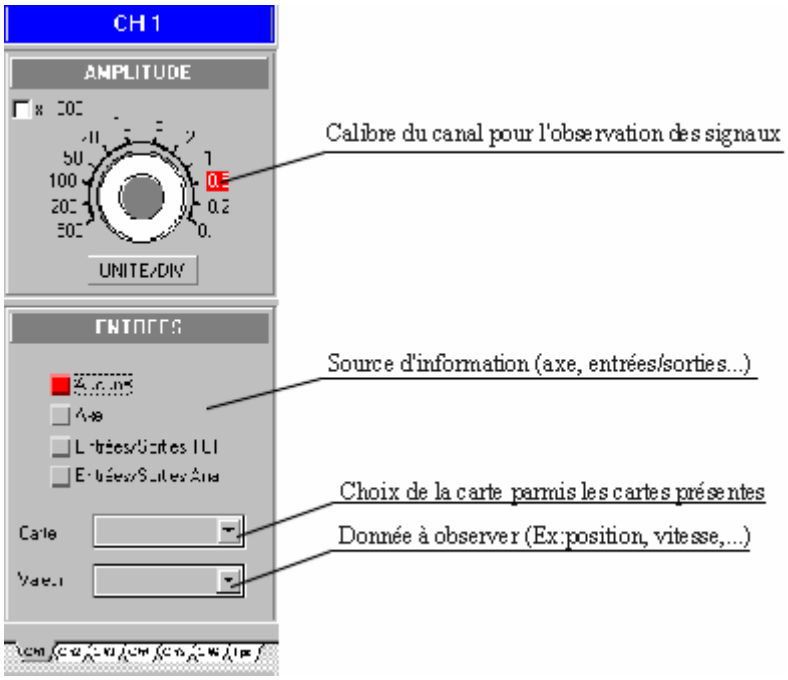
Action: Cette commande ouvre l'oscilloscope. Cet outil d'aide à la mise en œuvre permet d'enregistrer et de visualiser toutes les informations de cartes d'axes ou d'entrées / sorties. Il est capable d'enregistrer jusqu'à six variables différentes.

L'oscilloscope est configuré en trois partie : l'écran de visualisation, la zone de configuration de l'acquisition et la zone de configuration des voies.

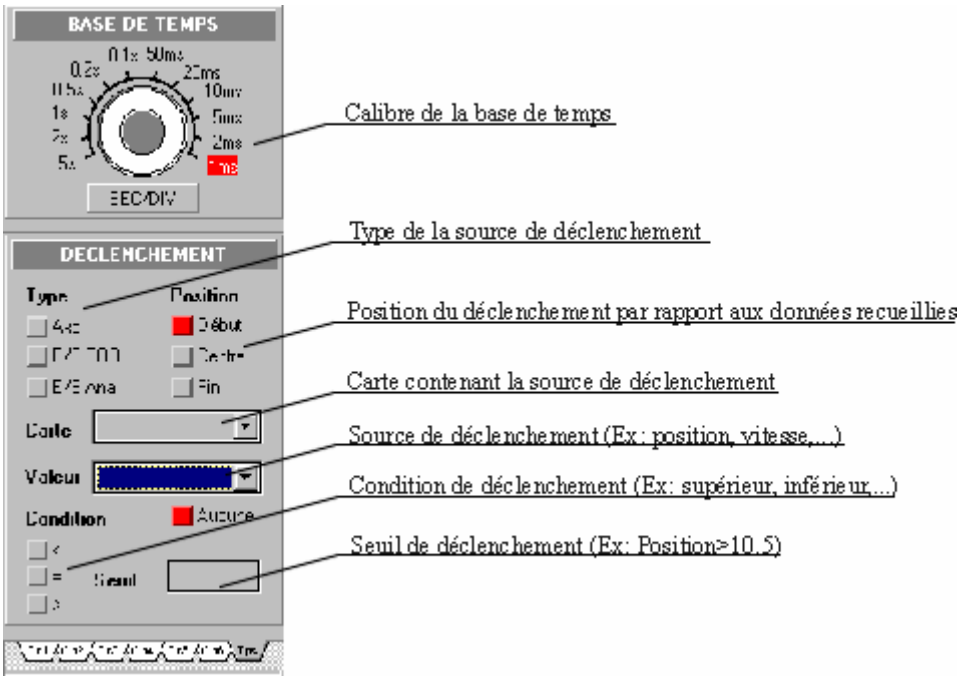


↳ La zone de configuration de l'acquisition permet de spécifier le nombre d'échantillon pendant la période d'acquisition. Elle permet aussi de lancer l'acquisition et d l'imprimer.

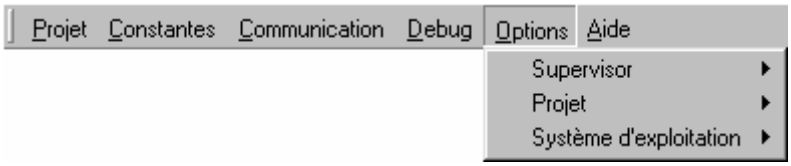
↳ La zone de configuration des voies comporte 6 onglets relatifs à chacune des voies et un autre relatif à la base de temps. Pour les onglets relatif aux voies, on peut définir le type de carte , la carte et le paramètre que l'on veut modifier. Par exemple pour une carte d'axe, on choisit le paramètre d'erreur de poursuite.



↳ La zone de visualisation affiche les six voies prédéfinies. En double-cliquant sur cette zone, la zone se place en plein écran. Il permet de plus de donner des informations en X et en Y (haut de l'écran) de la position du curseur. On peut aussi définir des références en X et en Y. Pour les définir, il faut cliquer sur dX ou dY (haut de l'écran). Les barres d'intersection deviennent pleine et mobile. Il suffit de cliquer à l'endroit où l'on veut déposer cette référence. Les positions dX et dY seront donner par rapport à cette référence.



3-4-5- Menu Options



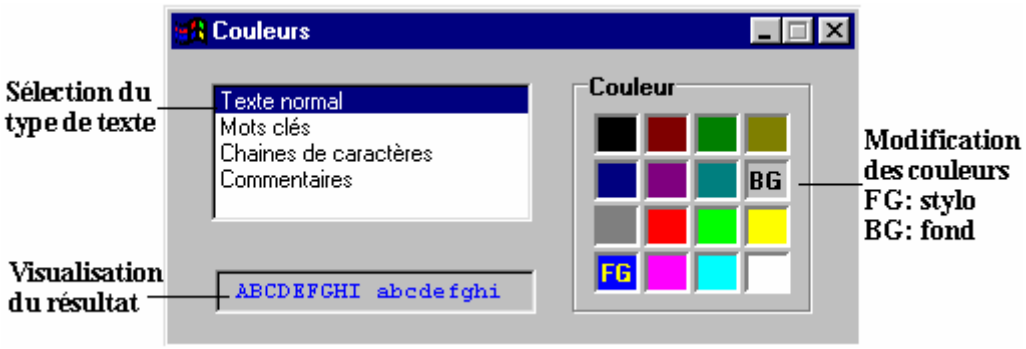
Supervisor 640

Langue

Ce sous-menu permet de choisir la langue dans laquelle le logiciel SPL sera exploité.

Editeur

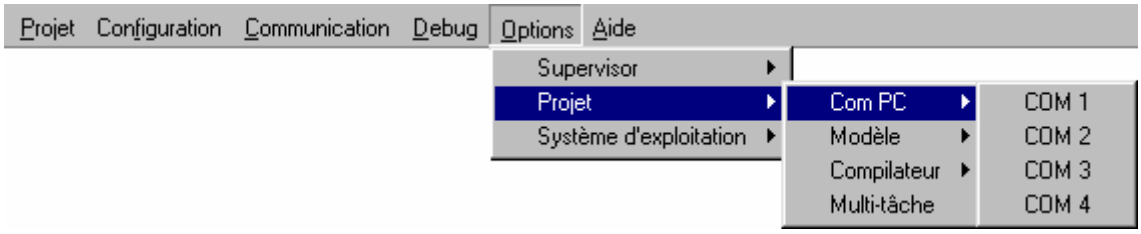
Ce sous-menu permet de personnaliser la couleur et le fond des textes, mots clés… de l'éditeur de tâche.



La procédure de modification est la suivante : sélectionner l'un des types de texte, modifier la couleur du texte par un clique sur le bouton de gauche de la souris et la couleur du fond par un clique sur le bouton de droite de la souris. Un écran de visualisation permet d'observer les modifications.

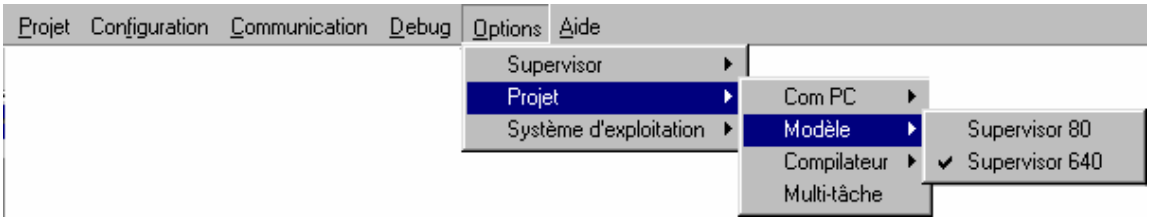
Projet

Com PC



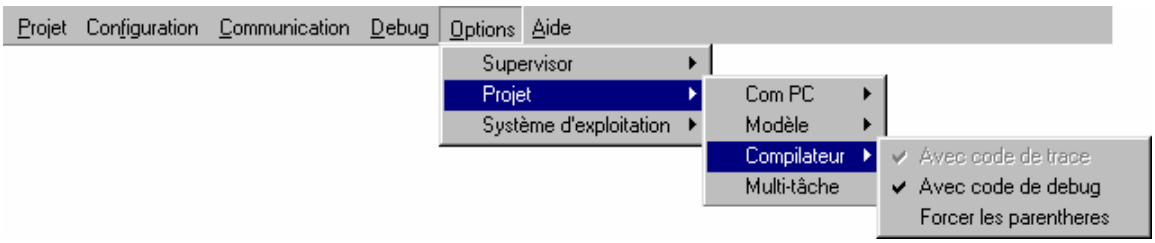
Ce sous-menu permet de sélectionner le port de communication du PC qui sera en liaison avec le SUPERVISOR

Modèle



Permet de sélectionner le modèle de SUPERVISOR utilisé.

Compilateur



Avec code de trace

Action: Cette commande permet de rajouter ou non lors de la compilation les informations nécessaires pour visualiser la trace des tâches en mode débogage. Cette commande est intéressante pour les tests mais elle augmente la taille du fichier compilé et ralentit légèrement l'exécution des tâches. Lors de sa validation ou dévalidation, elle nécessite une recompilation du projet pour que celle-ci soit prise ou non en compte.

Forcer les parenthèses

Action : Cette commande permet de renforcer les tests de compilation sur les parenthèses.

Multitâches

Action : Cette commande autorise les modifications des paramètres du multitâche. Une boîte de dialogue apparaît et permet la modification des temps de vieillissement et de tranche normale des tâches.

Système d'exploitation

Action : Ces commandes permettent de mettre à jour ou d'effacer l'operating system. Attention : cette procédure est réservée à des utilisateurs avertis.

3-4-6- Menu Aide

Assistant

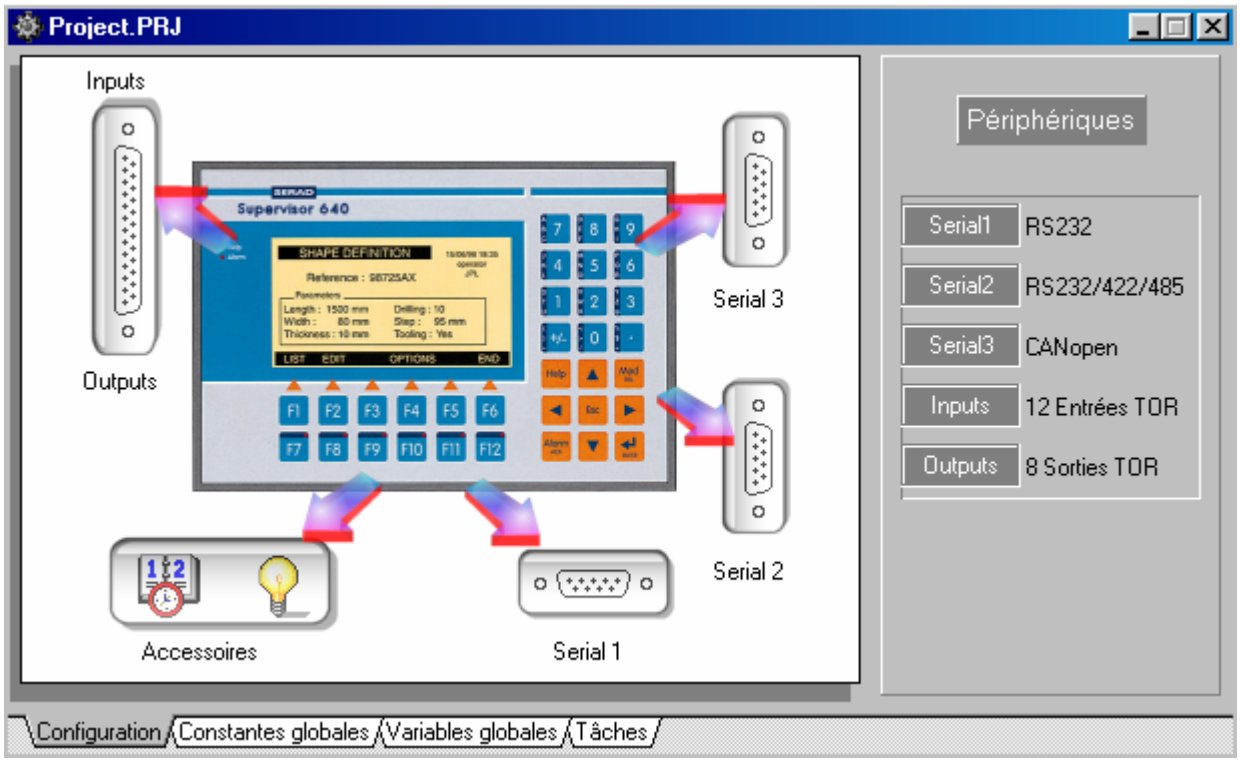
Action: Cette commande valide ou non l'affichage des informations bulle sur les icônes ainsi que des messages complémentaires d'avertissement.

A propos …

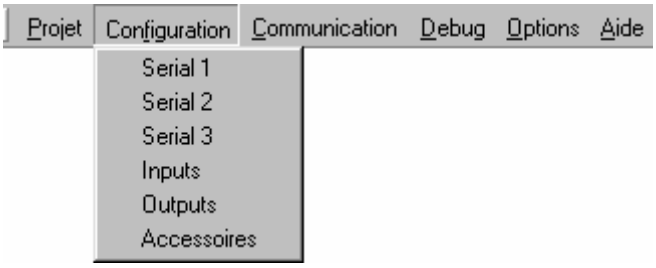
Action: Cette commande ouvre une boîte de dialogue indiquant la version du logiciel, sa date de création, etc…

3-4-7- Onglet et Menu Configuration

La fenêtre configuration se présente sous la forme de deux zones distinctes. La première située sur la gauche visualise la face avant du SUPERVISOR ainsi que ses différents connecteurs. C'est sur celle-ci que l'on vient réaliser la configuration du SUPERVISOR. Parmi les éléments paramétrable, on trouve l'horodateur, la SERIAL1, la SERIAL2, le SERIAL3, les INPUTS et les OUTPUTS. La deuxième zone intitulée "Périphériques" récapitule l'affectation des différents connecteurs.



Le choix de cet onglet, active une nouvelle option « CONFIGURATION » dans le menu principal :



A) Serial 1

Action : Permet de configurer le port série 1 du SUPERVISOR en paramettrant :

- la vitesse
- le nombre de bits de données
- la parité
- le bit de stop

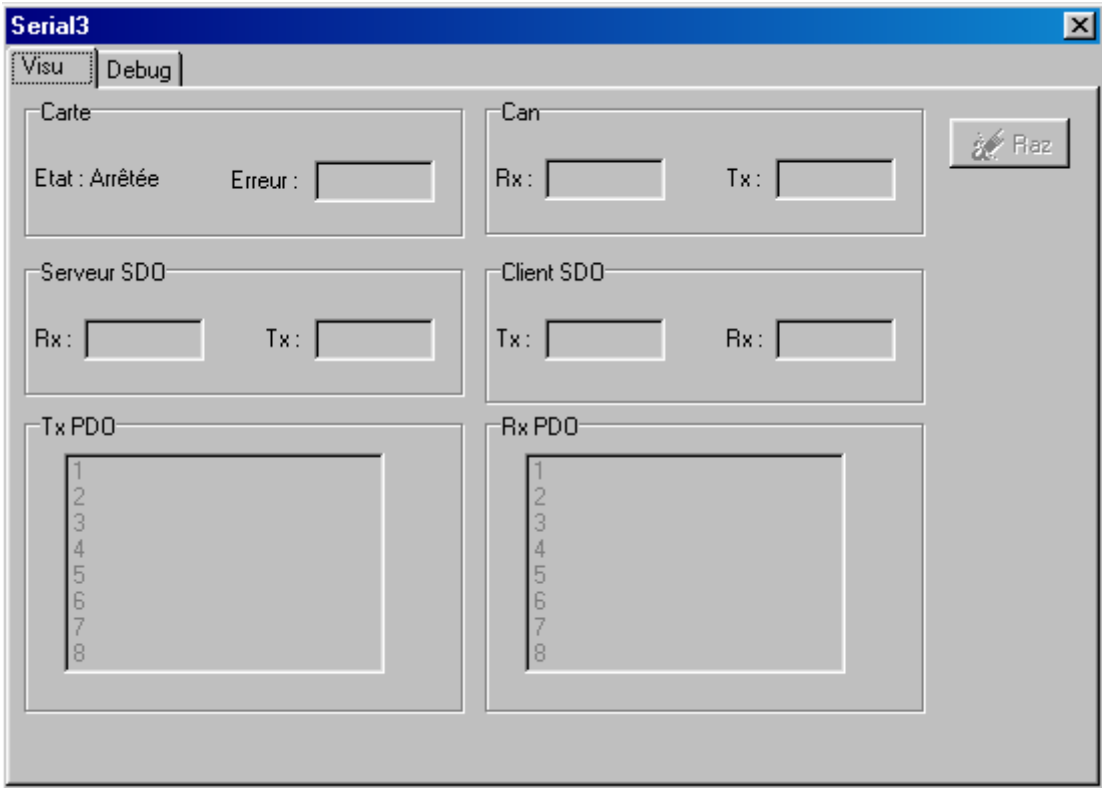
B) Serial 2

Action : Permet de configurer le port série 2 du SUPERVISOR en paramettrant :

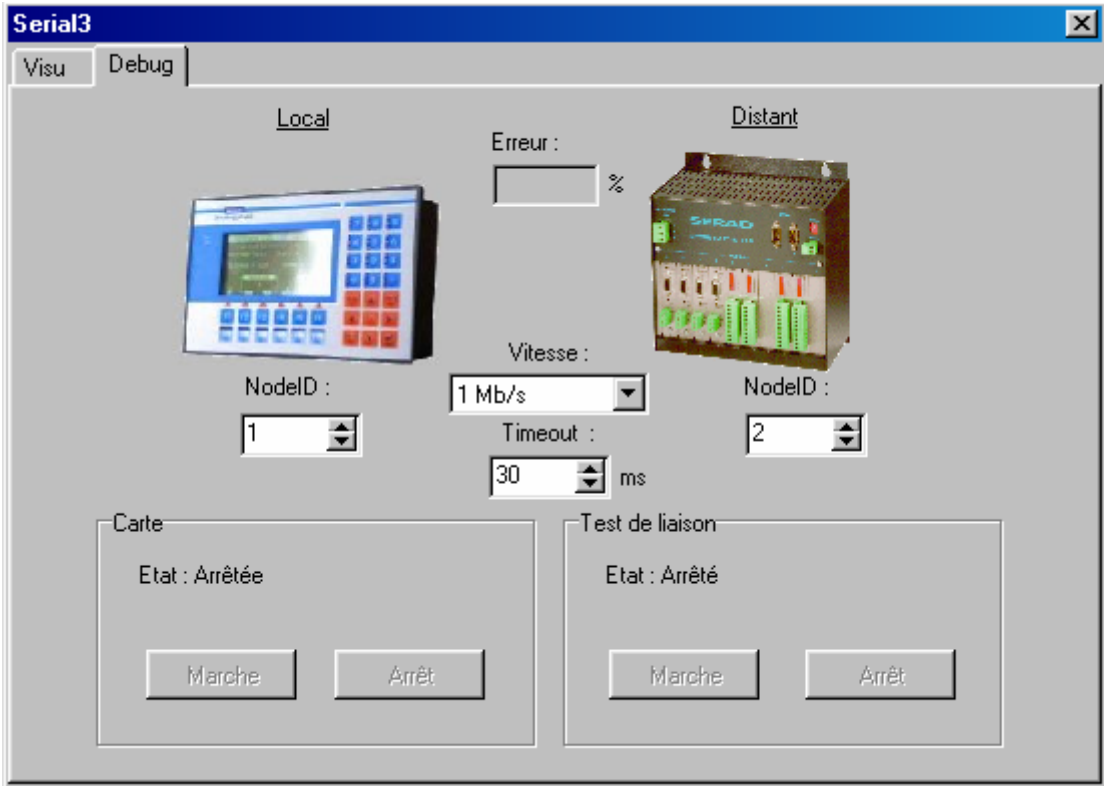
- la vitesse
- le nombre de bits de données
- la parité

- le bit de stop

C) Serial 3



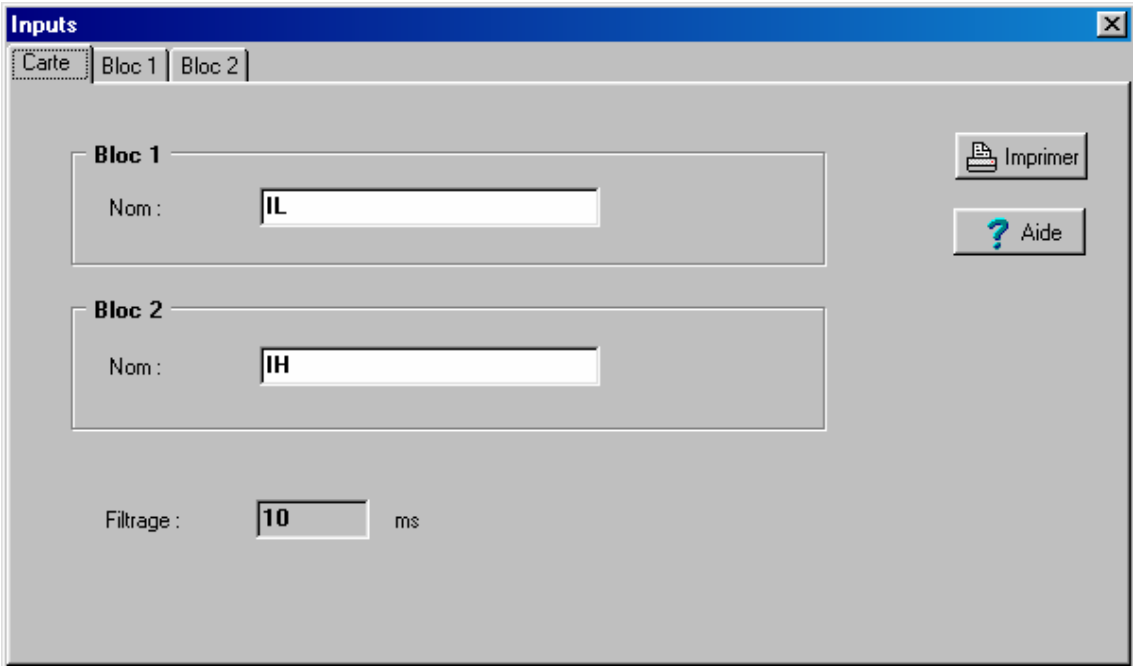
Action : Permet de paramètrer la liaison CANOpen entre le SUPERVISOR et un autre périphérique CANOpen.



Action : Permet de controler la liaison CANOpen entre le SUPERVISOR et un autre périphérique CANOpen.

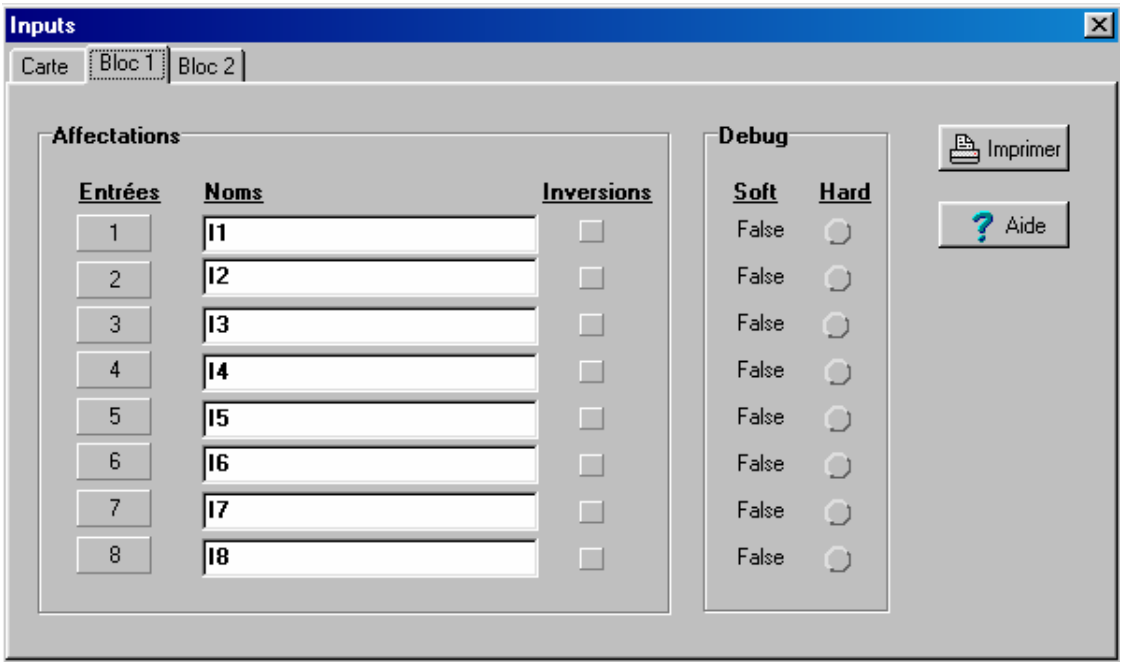
D) Inputs

a) Carte :



Action : Permet de donner un nom aux blocs d'entrées et de leur fixer un filtrage

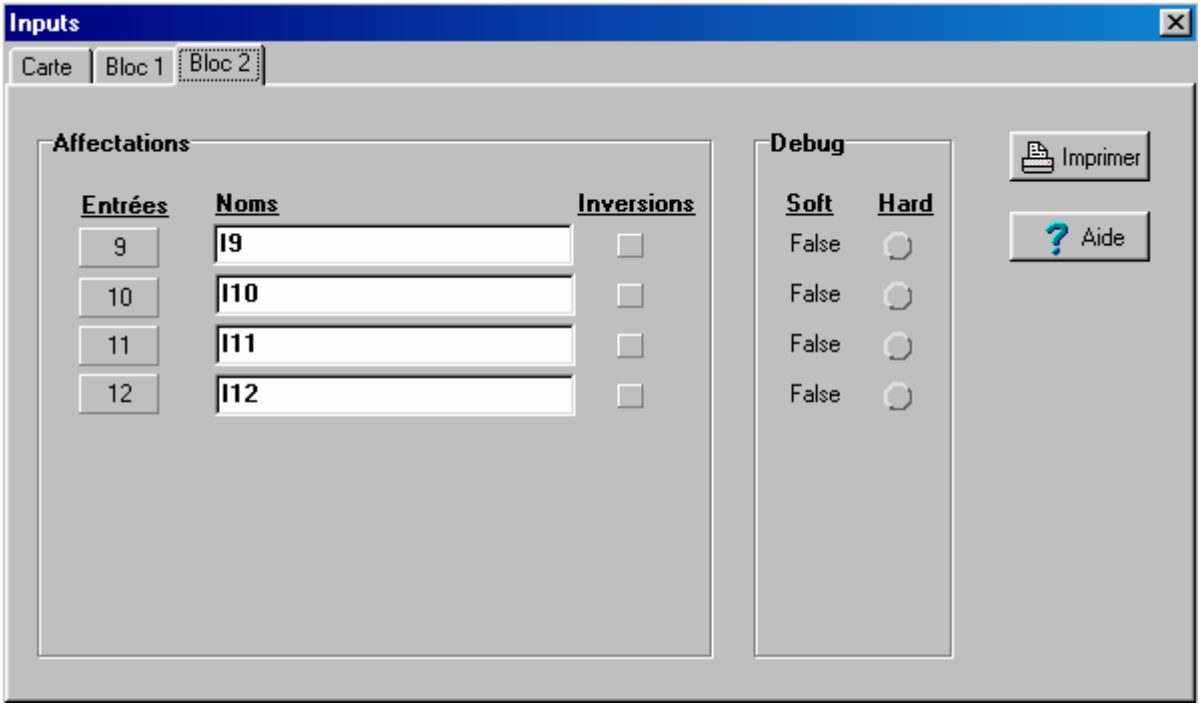
b) Bloc 1 :



Action : Permet de donner un nom à chaque bit d'entrées , de les inverser

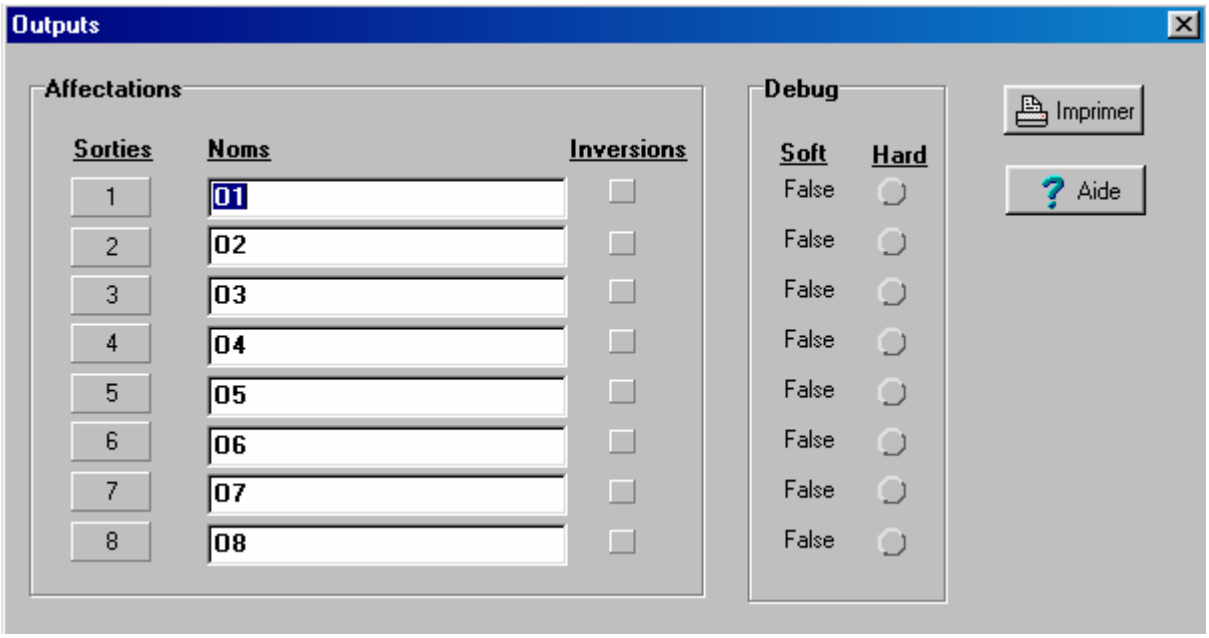
Si on est en mode debug, on peut visualiser leur état et le modifier.

c) Bloc 2 :



Action : Idem que Bloc 1

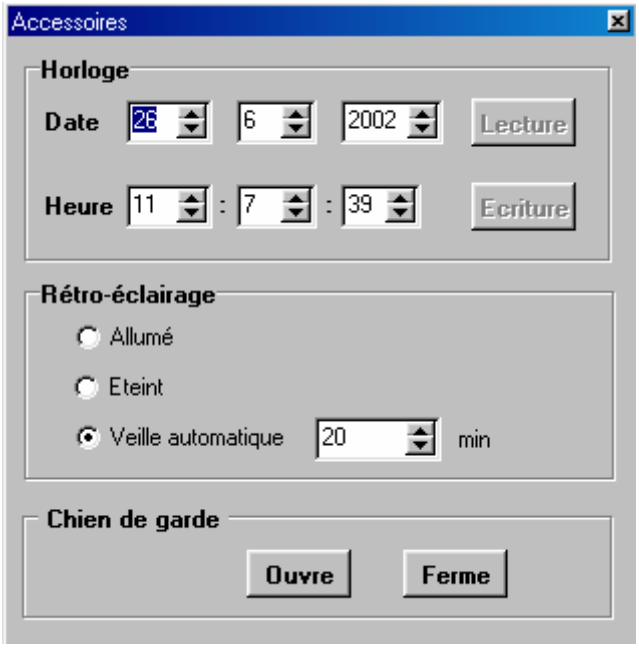
E) Outputs



Action : Permet de nommer chaque bit de sorties, de choisir type de sortie (inverser ou non).

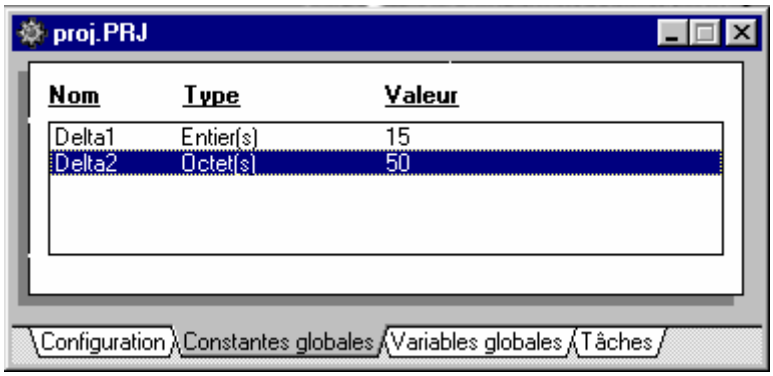
En mode debug, permet de visualiser l'état de sorties.

F) Accessoires



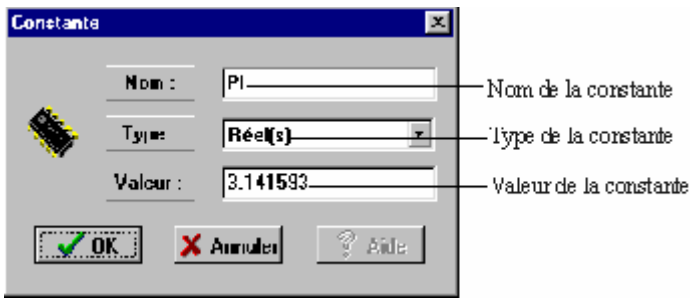
Action : Permet de régler l'horloge, la date et le mode de rétro-éclairage (seulement sur le S640) ainsi que de changer l'état du chien garde.

3-4-8- Onglet Constantes globales



Dans l'onglet « Constantes globales », ce sont toutes les constantes globales qui sont énumérées avec leurs caractéristiques (nom, type et valeur). Sur cet onglet, on peut ajouter, modifier, supprimer ou visualiser une constante. Les commandes d'ajout, de modification, de suppression ou de visualisation nécessitent une recompilation du projet et l'envoi des tâches dans le SUPERVISOR.

La commande « Ajouter » permet de définir une nouvelle constante globale au projet. Le paramétrage de cette constante se fait au moyen d'une boîte de dialogue.



La commande ajouter (une constante) peut-être obtenue de trois manières différentes :

- Par le menu CONSTANTE
- Un clique sur le bouton droit ouvre un menu qui comprend la commande « Ajouter »

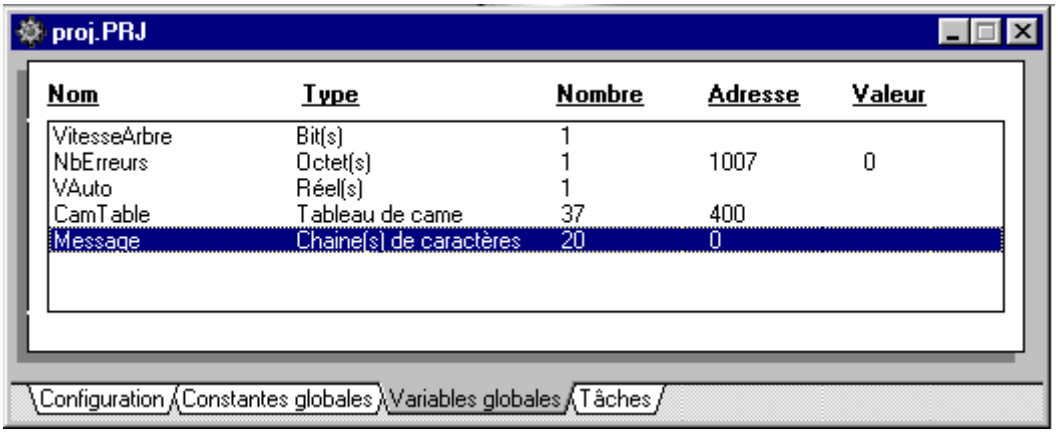
La commande « Modifier » ou « Visualiser » permet de redéfinir les caractéristiques d'une constante globale, sauf son type. La boîte de dialogue est la même que pour l'ajout d'une nouvelle constante globale. La commande modifier (une constante) s'obtient des façons suivantes :

- Par le menu CONSTANTE et en ayant sélectionné au préalable la constante à modifier
- En double cliquant sur la constante à modifier.
- Un clique sur le bouton droit ouvre un menu qui comprend la commande « Modifier » (après avoir sélectionné la constante à modifier).

La commande « Supprimer » permet de détruire une constante globale au projet. Elle s'obtient des façons suivantes :

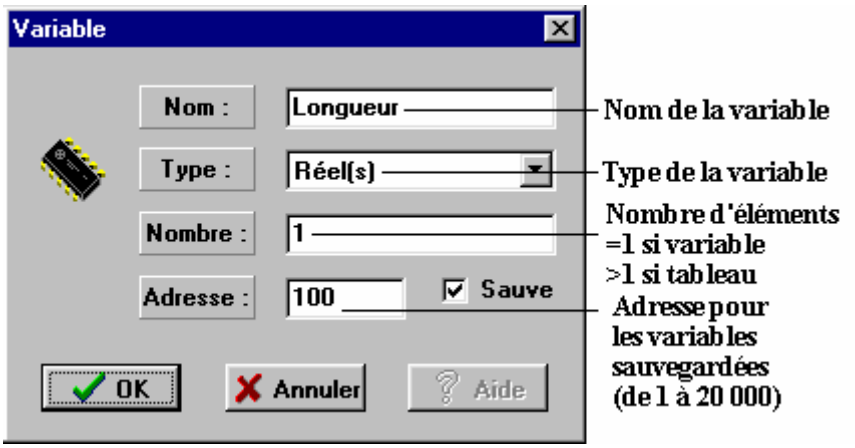
- Par le menu CONSTANTE et en ayant sélectionné au préalable la constante à supprimer
- Un clique sur le bouton droit ouvre un menu qui comprend la commande « Supprimer » (après avoir sélectionné la constante à supprimer).

3-4-9- Onglet Variables globales



Dans l'onglet « Variables globales », ce sont toutes les variables globales qui sont énumérées avec leurs caractéristiques (nom, type, nombre, adresse et valeur). La notion « nombre » permet de définir une variable globale de type tableau. Sa valeur correspond au nombre d'éléments du même type réservé. L'adresse doit être fixée lorsque l'on désire que la variable soit de type sauvegardé (adresse de 1 à 20000). Sur cet onglet, on peut ajouter, modifier, supprimer ou visualiser une variable. Les commandes d'ajout, de modification, de suppression ou de visualisation nécessitent une recompilation du projet et l'envoi des tâches dans le SUPERVISOR. Dans le cas d'une variable sauvegardée, contenant une valeur, il faut également envoyer les variables dans le SUPERVISOR.

La commande « Ajouter » permet de définir une nouvelle variable globale au projet. Le paramétrage de cette variable se fait au moyen d'une boîte de dialogue.



La commande ajouter (une variable) peut-être obtenue de trois manières différentes :

- Par le menu VARIABLES
- Un clique sur le bouton droit ouvre un menu qui comprend la commande « Ajouter »

La commande « Modifier » permet de redéfinir les caractéristiques d'une variable globale, sauf son type. La boîte de dialogue est la même que pour l'ajout d'une nouvelle variable globale. La commande modifier (une variable) s'obtient des façons suivantes :

- Par le menu VARIABLES et en ayant sélectionné au préalable la variable à modifier
- Un clique sur le bouton droit ouvre un menu qui comprend la commande « Modifier » (après avoir sélectionné la constante à modifier).

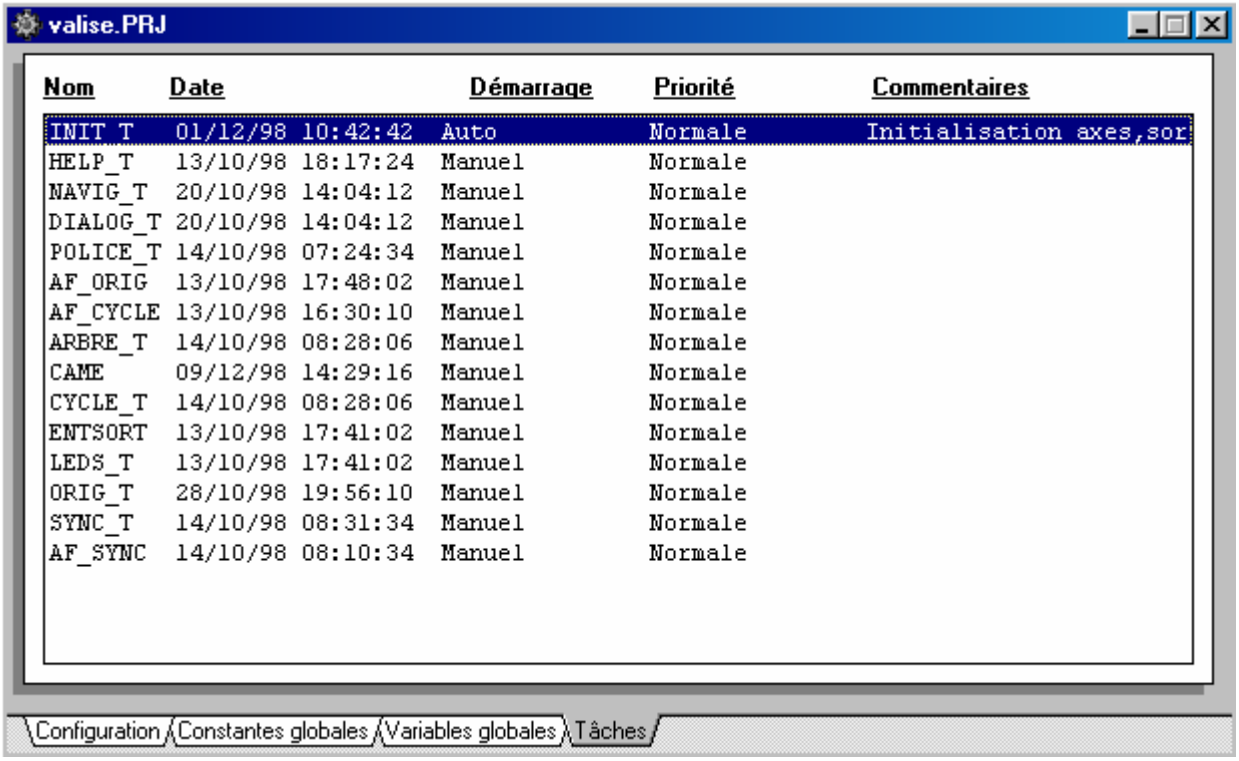
La commande « Supprimer » permet de détruire une variable globale du projet. Elle s'obtient des façons suivantes :

- Par le menu VARIABLES et en ayant sélectionné au préalable la variable à supprimer
- Un clique sur le bouton droit ouvre un menu qui comprend la commande « Supprimer » (après avoir sélectionné la variable à supprimer).

La commande « visualiser » permet de visualiser l'état d'une variable. Elle est intéressante pour les variables de type tableau car elle permet de visualiser tous les éléments de celui-ci. Elle s'obtient des façons suivantes :

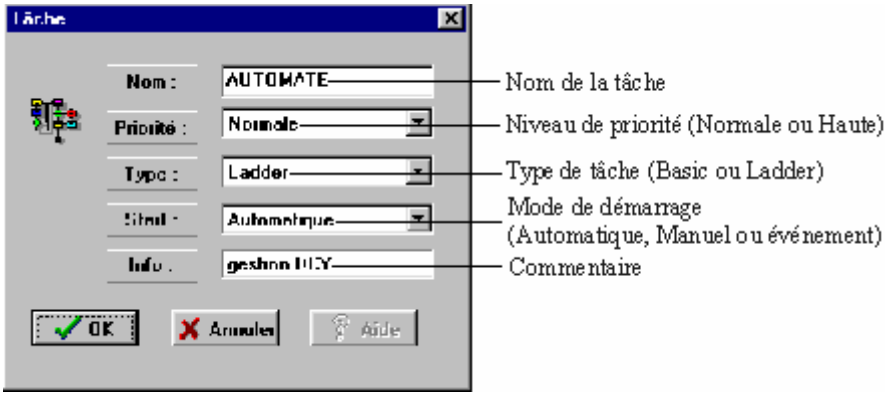
- Par le menu VARIABLES et en ayant sélectionné au préalable la variable à visualiser
- Par un double-clique sur la variable à visualiser
- Un clique sur le bouton droit ouvre un menu qui comprend la commande « Visualiser » (après avoir sélectionné la constante à visualiser).

3-4-10- Onglet Tâches



Dans l'onglet « Tâches », ce sont toutes les tâches qui sont énumérées avec leurs caractéristiques (nom, date de création, type de démarrage et commentaire associé). Sur cet onglet, on peut ajouter, modifier, supprimer ou visualiser une variable. Les commandes d'ajout, de modification, de suppression ou de visualisation nécessitent une recompilation du projet et l'envoi des tâches dans le SUPERVISOR.

La commande « Ajouter » permet de définir une nouvelle tâche au projet. Le paramétrage de cette constante se fait au moyen d'une boîte de dialogue. Une tâche est caractérisée par sa priorité (normale ou haute), son type (basic ou ladder), son mode de démarrage (manuel, automatique ou événementiel) et un commentaire. Le type de la tâche permet de définir quel est le mode d'édition de celle-ci.



La commande ajouter (une tâche) peut-être obtenue de trois manières différentes :

↳ Par le menu TACHES

↳ Un clique sur le bouton droit ouvre un menu qui comprend la commande « Ajouter »

La commande « Modifier » permet de redéfinir les caractéristiques de la tâche de la même manière que lorsque l'on fait un ajout. La commande modifier (une tâche) s'obtient des façons suivantes :

↳ Par le menu TACHES et en ayant sélectionné au préalable la variable à modifier

↳ Un clique sur le bouton droit ouvre un menu qui comprend la commande « Modifier » (après avoir sélectionné la tâche à modifier).

La commande « Visualiser » permet d'exécuter l'éditeur de tâche associée à son type : basic ou ladder. L'exécution peut aussi être obtenue par un double clique sur la tâche à éditer. Elle s'obtient des façons suivantes :

↳ Par le menu TACHES et en ayant sélectionné au préalable la tâche à visualiser

↳ Par un double-clique sur la tâche à visualiser

↳ Un clique sur le bouton droit ouvre un menu qui comprend la commande « Visualiser » (après avoir sélectionné la tâche à visualiser).

La commande « Supprimer » permet de détruire une tâche du projet. Elle s'obtient des façons suivantes :

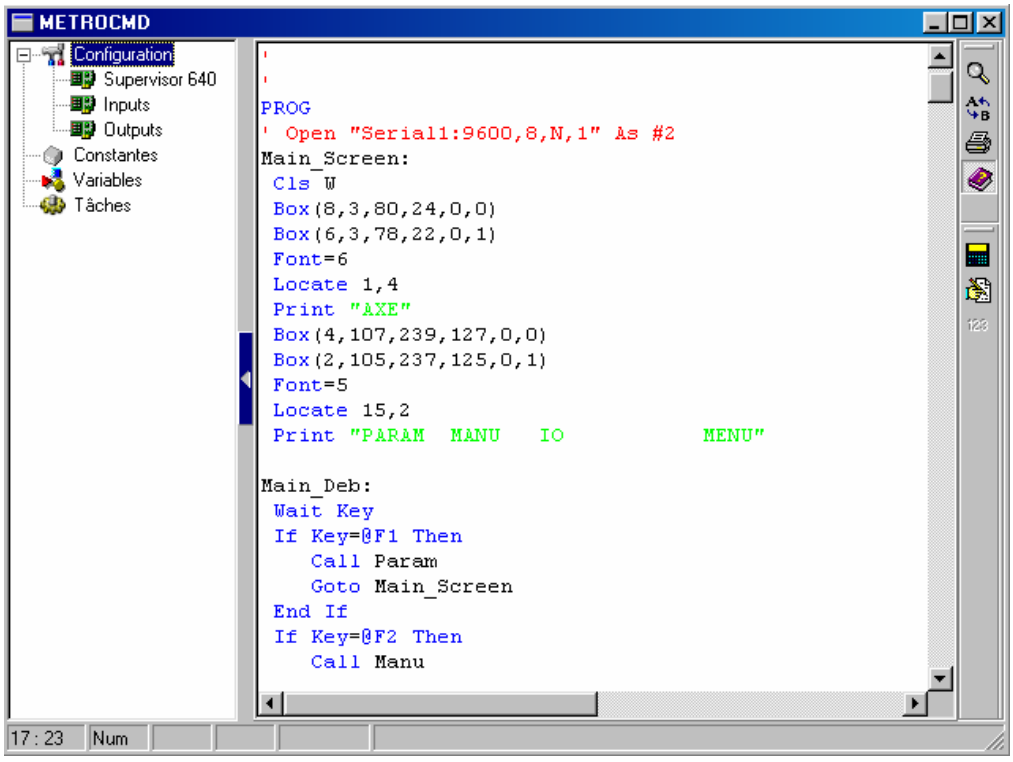
↳ Par le menu TACHES et en ayant sélectionné au préalable la tâche à supprimer

↳ Un clique sur le bouton droit ouvre un menu qui comprend la commande « Supprimer » (après avoir sélectionné la tâche à supprimer).

3-5- Editeurs

3-5-1- Editeur de tâche Basic

L'éditeur basic se décompose en une zone d'édition de texte dans lequel l'utilisateur vient entrer le code basic associé à son programme, une barre d'outil permettant l'aide à l'édition du code et une zone indiquant la configuration du SUPERVISOR. La configuration du SUPERVISOR comprend tous les paramètres des cartes du projet, les constantes et variables globales définies et les variables locales des tâches.



Les outils de l'éditeur permettent de simplifier la mise en œuvre de certaines instructions du basic. Les outils sont réunis en sous groupe :

⇨ Les outils propres à l'éditeur

↳ L'outil de recherche d'un mot ou d'un groupe de mot . Cette recherche s'effectue dans la tâche et peut se faire en confondant ou dissociant (cocher la case respecter la casse) majuscule ou minuscule.

↳ La commande remplacer permet de faire une recherche d'occurrence et de la remplacer dans la tâche.

↳ La commande d'impression

↳ L'icône suivant permet d'afficher une syntaxe rapide de l'instruction qui est pointée par le curseur de l'éditeur de tâches.

↳ On peut aussi intégrer le copier (CTRL+C), le coller (CTRL+V) et le couper (CTRL+X).

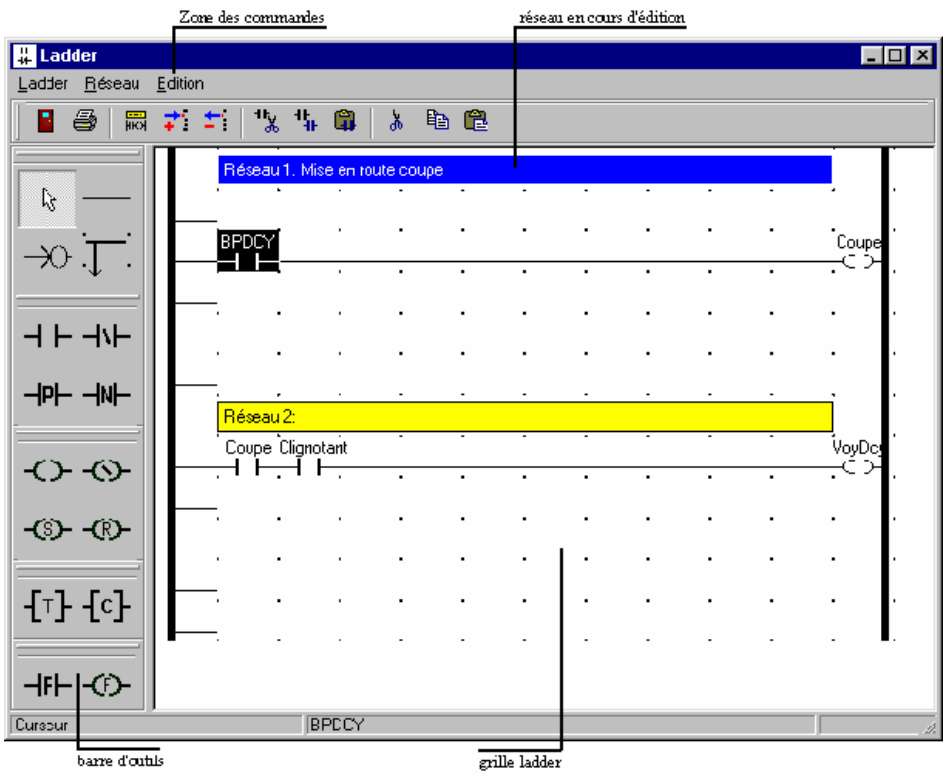
⇨ Les outils liés aux instructions de communications

↳ L'outil Terminal permet de définir tout le code concernant le texte à afficher sur un pupitre. La boîte de dialogue est définit par un écran avec le pupitre sélectionné dans le menu Option et quelques boutons de commande. Pour générer du code, il suffit d'inscrire sur l'écran sa configuration d'affichage et d'exécuter la commande Ecrire. Pour visualiser ce que donne du code sur l'écran, il faut sélectionner le code dans la tâche et exécuter la commande Lire. Une commande Effacer permet d'initialiser l'écran du pupitre.

↳ La commande d'édit permet de définir une saisie numérique ou alphanumérique sur un pupitre.

↳ La commande Format permet de réaliser le formatage d'une variable.

3-5-2- Editeur de tâche Ladder



L'éditeur Ladder se décompose en une zone d'édition du programme ladder (la grille), une barre des outils qui peuvent être inséré et une zone de commande. Un programme ladder est constitué de plusieurs réseaux limités à 50 par tâche. Chaque réseau est la constitution d'un en-tête ou commentaire et d'une seule expression associée à une ou plusieurs sorties.

↳ La barre des outils permet de définir l'élément qui sera déposé sur la grille. La sélection de l'outil se fait par un clique sur le bouton gauche de la souris de l'élément désiré.

↳ La grille permet de déposer les éléments et donc de définir le programme.

La sélection de la case de la grille est matérialisée par une focalisation de l'élément (fond noir).

Le dépôt d'un contact, bobine, lien ou bloc sur la grille se fait par un clique sur le bouton de gauche de la souris. Le lien parallèle s'insère sur le coin gauche de la case sélectionnée et se dirige vers le bas.

La configuration de l'élément déposé peut intervenir à tout moment par un double clique sur le bouton gauche de la souris :

⇨ Pour les bobines et les contacts, un écran de configuration du SUPERVISOR apparaît sur la barre des outils. Celui-ci regroupe les différentes variables déclarées du système (entrées, sorties et variables globales) ainsi qu'un nombre limité de 64 bits et d'un groupe de bits système. Les bits apparaissent par défaut avec un nom de la forme <bit>+n°du bit. Néanmoins, ils peuvent être reconfiguré par un clique simple sur le bouton gauche de la souris ou à l'aide du menu qui apparaît sur le clique du bouton de droite de la souris. Les bits systèmes sont au nombre de trois : Un bit d'initialisation qui reste à un durant l'exécution du premier cycle du programme et deux bits clignotants dont l'un ayant une demi-période de 500ms et l'autre de 1s.

⇨ Pour les blocs une boîte de dialogue permet de configurer son nom. Pour les blocs tempos, on définit la durée de temporisations et sa base de temps ou la variable temps qui lui sera associée. Pour les blocs compteurs, on peut définir le type compteur ou décompteur ainsi que la valeur de présélection ou sa variable de présélection.

⇨ Pour les blocs libres, une boîte de dialogue permet de saisir le code basic qui lui sera associé. Pour les contacts libres, le code ressemblera en particulier à un test et pour les bobines à une action. Une erreur dans l'édition du code basic ne sera détectée qu'à la compilation du projet.

La sélection du réseau en cours d'édition est matérialisée par un commentaire en bleu. Le commentaire d'un réseau peut être édité en double cliquant sur celui-ci.

↳ la zone des commandes

⇨ Une commande pour quitter le logiciel  .

⇨ Une commande d'impression  .

⇨ Les commandes d'ajout d'un réseau  , les commandes d'insertion du réseau avant le réseau détenant la focalisation  et la commande de suppresion du réseau détenant la focalisation  .

⇨ Les commandes pour couper  , copier  et coller un élément  . L'élément couper ou copier est l'élément détenant la focalisation. L'élément coller est inséré sur une case vide détenant la focalisation.

⇨ Les commandes pour couper  , copier  et coller un réseau  . Le réseau couper ou copier est le réseau détenant la focalisation. Le réseau coller est inséré avant le réseau détenant la focalisation.

⇨ La touche SUPPR permet d'effacer l'élément de la grille possédant la focalisation. Un lien parallèle ne peut être enlever qu'en ce type de lien sélectionnant dans la boîte à outil et en cliquant sur la case correspondante de la grille ladder.

⇨ La fonction «Aller au réseau» de la barre des tâches permet de se positionner sur un réseau particulier.

Attention : Une tâche ladder est automatiquement traduite en basic de façon interne. Il est déconseillé d'écrire une tâche ladder longue ou complexe afin d'éviter les fortes dégradations du temps de cycle et les limitations de la traduction basic.

4- LANGAGE DE PROGRAMMATION

4-1- Introduction

4-1-1- Description

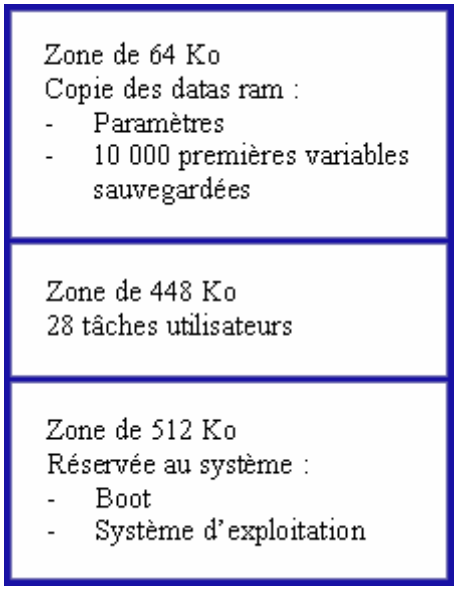
Supervisor Programming Language est un outil de programmation puissant et simple à utiliser. Il offre une architecture structurée rencontrée sur les langages de haut niveau. Pour une programmation flexible, ce langage est géré par un noyau temps réel multitâches, utilisant des instructions pseudo-basic et contenant également toutes les fonctions d'automate.

Le langage intègre aussi la gestion de données sous la forme de constantes ou de variables globales, locales, sauvegardées...

Un projet développé à partir du SPL peut contenir jusqu'à 28 tâches fonctionnant en parallèle. Chaque tâche possède un niveau de priorité et peut être écrite en basic ou en ladder. Une tâche spéciale permet de traiter les événements rapides.

4-1-2- Affectation du plan mémoire du SUPERVISOR

Mémoire Flash 1 Mo



Mémoire Ram 512 Ko sauvegardée par pile

Cette RAM comporte une zone de 128 Koctets correspondant au fichier du SUPERVISOR. Ce fichier permet à l'utilisateur de venir y déposer des données pendant l'exécution du SUPERVISOR. Ces données sont sauvegardées par pile et donc ne sont donc pas perdues. L'utilisateur gère cette zone mémoire à l'aide des fonctions suivantes : OPEN, PRINT, INPUT$ et SEEK

4-2- Les données

4-2-1- Constantes globales

Les constantes sont déclarées à partir de l'onglet constantes globales du logiciel SPL. Elles acceptent les types bit, octet, entier, entier long, réel, chaîne de caractères.

Les constantes définies dans un projet sont des données pouvant seulement être lues. Elles sont stockées en mémoire flash car elles sont intégrées dans le code des tâches lors de la compilation. Une constante globale peut être utilisée par plusieurs tâches.

4-2-2- Variables globales

Les variables sont déclarées à partir de l'onglet variables globales du logiciel SPL.

Une variable globale et une constante ne peuvent pas porter le même nom dans un projet car la distinction ne pourra pas être effectuée lors de la compilation. Les types des variables globales et des constantes sont les mêmes.

Une variable globale peut être utilisée par plusieurs tâches et est accessible à tout moment.

Elle sera traitée comme un tableau si lors de sa création, *"nombre"* est supérieur à 1. L'index du tableau commence à 1. L'index peut être une valeur immédiate, une variable octet ou entière.

Exemple :

```
Longueur = TableCote[5]      '  le cinquième élément de TableCote
                             ' est stocké dans la variable Longueur
```

' Attention : Une écriture dans un tableau à l'index 0 est interdite : ceci peut entraîner un dysfonctionnement de l'application

Lors de la création de la variable, on peut la déclarer en variable ou tableau de variables sauvegardé.

Sur une coupure secteur, la valeur de la variable sera conservée. On dispose de 20 000 variables sauvegardées stockées aux adresses 1 à 20 000. Ainsi, chaque variable sauvegardée doit être affecté à une adresse : quel que soit le type, une variable sauvegardée occupe une adresse.

' Attention : c'est à l'utilisateur de s'assurer qu'il n'y a pas de chevauchement entre variables lors des affectations d'adresses. Par exemple, si un tableau de 50 éléments est déclaré à l'adresse 100, les variables suivantes devront être à des adresses supérieures ou égale à 150.

Le chevauchement des variables peut être utilisé dans un seul cas : pour permettre l'accès à une même adresse mémoire à partir de noms différents.

Exemple :

TableModbus : tableau de 50 entiers déclarés à l'adresse 100

DecompteurPiece : variable entière déclarée à l'adresse 100

```
If TableModbus[1]=0 Then Goto FinProduction
If DecompteurPiece=0 Then Goto FinProduction
```

Ces deux lignes programmes ont la même signification mais la deuxième ligne est plus explicite.

Contrairement aux variables locales, vous devez déclarer une variable globale avant de pouvoir l'utiliser. Non sauvegardée, elle sera utilisée pour effectuer des liens entre tâches, tandis que sauvegardée, elle permettra de conserver des paramètres de réglage propres à l'application.

Type	Value	Memory size	Example
Bit	1/0, On/Off ou True/False	Unstored variable: 1 byte Stored variable : 6 bytes	State=On
Byte	0 to 255	Unstored variable: 1 byte Stored variable: 6 bytes	Middle=128
Integer	0 to 65535	Unstored variable: 2 bytes Stored variable: 6 bytes	NumLoop= 1000
Long integer	0 to +/- 2 147 483 647	Unstored variable: 4 bytes Stored variable: 6 bytes	VelMax= 100000
Real	$+/-2.9 \times 10^{-39}$ to $+/-1.7 \times 10^{+38}$	Unstored variable : 6 bytes Stored variable: 6 bytes	PositionMax=1256.152
String char	0 to 255	Length string +1	Error1= «Limit reached »

4-2-3- Variables locales

Ces variables ne sont accessibles que dans la tâche où elles sont déclarées (programme principal et sous programmes). Elles acceptent les types bit, octet, entier, entier long, réel, chaîne de caractères. Leurs valeurs ne sont pas conservées entre chaque mise sous tension.

Lorsque vous effectuez des calculs, il est souvent nécessaire de stocker temporairement des valeurs. Vous avez besoin de conserver les valeurs pour les comparer mais sans les stocker dans une variable globale.

Les variables locales n'ont pas besoin d'être explicitement définies avant d'être utilisées. Elles possèdent un caractère d'identification à la fin du nom pour indiquer le type de donnée. Les

variables locales d'une tâche ne peuvent pas être utilisées par une autre tâche. Deux variables avec le même nom, utilisées dans deux tâches, sont deux variables différentes. Dans une tâche, la variable peut être utilisée dans le programme principal et dans les sous programmes.

Le traitement d'une variable locale est plus rapide que celui d'une variable globale. On ne peut pas déclarer un tableau de variables locales.

' Attention : n'utilisez pas trop de variables locales de type chaîne de caractères car chacune occupent 256 octets en mémoire ram !

Exemple :

```
a%=10                                   ' variable entière
If Position  !>1000 Then Position  !=0   ' variable réelle
Compteur&=Compteur&+1                   ' variable entier long
FormFeed$=Chr$(10)+Chr$(13)             ' variable chaîne de caractères
```

Tableau récapitulatif des différents types :

Type	Value	Memory size	Declaration	Example
Bit	1/0, On/Off or True/False	1 byte	~	Sensor~=On
Byte	0 to 255	1 byte	#	Data#=128
Integer	0 to 65535	2 bytes	%	Nb%= 2000
Long integer	0 to +/- 2 147 483 647	4 bytes	&	Velocity&= 120000
Real	$+/-2.9 \times 10^{-39}$ to $+/-1.7 \times 10^{+38}$	6 bytes	!	Pos!=122.245
String	0 to 255	256 octets	$	Mes$= 'Error'

4-2-4- Conversion de types de données

Pour convertir un type de données en un autre, utilisez les fonctions ci-dessous :

Destination / Source	Bit	Byte	Integer	Long integer	Real	String
Bit	X	Directe	Directe	Directe		Str$ or Format$
Byte	'.1' to '.8'		Directe	Directe	Directe	Str$ or Format$
Integer	'.1' to '.16'	'.L' or '.H'	X	Directe	Directe	Str$, Mki$, Mkir$ or Format$
Long integer	'.1' to '.32'	X	LongToInteger	X	Directe	Str$, Mkl$, Mklr$ or Format$
Real	X	RealToByte	RealToInteger	RealToLong	X	Str$ or Format$
String	X	X	Cvi, Cvir	Cvl, Cvlr	VAL	X

On peut extraire un bit d'une variable octet ou entière grâce à la fonction « .NuméroBit ».

Pour un octet, NuméroBit est compris entre 1 et 8, 1 représentant le bit de poids faible.

Pour un entier, NuméroBit est compris entre 1 et 16, 1 représentant le bit de poids faible.

NuméroBit peut être une valeur ou une variable de type octet.

On peut extraire un octet d'une variable entière grâce à la fonction « .L » ou « .H ».

La fonction « .L » extrait l'octet de poids faible alors que « .H » l'octet de poids fort.

Exemples :

```
VarOctet=4
VarBit=VarOctet.3                  ' VarBit=1
VarOctet=16
Index=5
VarBit=VarOctet.Index              ' VarBit=1
VarBit=1
VarOctet=VarBit                    ' VarOctet=1
VarEntier=259
VarOctet=VarEntier.L               ' VarOctet=3
VarOctet=VarEntier.H               ' VarOctet=1
VarLong=261
VarReel=38.15
VarOctet=RealToByte(VarReel)       ' VarOctet=38
VarBit=1
VarEntier=VarBit                   ' VarEntier=1
VarOctet=128
VarEntier=VarOctet                 ' VarEntier=128
VarLong=45200
VarEntier=LongToInteger(VarLong)   ' VarEntier=45200
VarReel=54200.65
VarEntier=RealToInteger(VarReel)   ' VarEntier=54200
VarBit=1
VarLong=VarBit                     ' VarLong=1
VarOctet=128
VarLong=VarOctet                   ' VarLong=128
VarEntier=45200
VarLong=VarEntier                  ' VarLong=45200
VarReel=154200.65
VarLong=RealToLong(VarReel)        ' VarLong=154200
VarOctet=128
VarReel=VarOctet                   ' VarReel=128
VarEntier=45200
VarReel=VarEntier                  ' VarReel=45200
VarEntier=154200
VarReel=VarEntier                  ' VarReel=154200
VarChaîne=  «  -125.45  »
VarReel=Val(VarChaîne)             ' VarReel=-125  .45
VarReel=1510.55
VarChaîne=Str$(VarReel)            ' VarChaîne=  «   1510.55  »
```

4-2-5- Notations numériques

Les valeurs numériques peuvent être exprimées en décimal, en hexadécimal, en binaire.

Exemple :

```
VarOctet=254                       ' notation décimale
VarOctet=0FEh                      ' notation hexadécimale
VarOctet=11111110b                 ' notation binaire
```

4-3- Les tâches

4-3-1- Principe du multitâches

Le moniteur temps réel multitâches gère jusqu'à 32 tâches en parallèle :

- 4 tâches internes réservées au système
- 27 tâches utilisateurs pseudo-basic ou ladder
- 1 tâche spécifique pour la gestion d'événements

Le multitâches bascule de la tâche courante vers la tâche suivante si :

↳ Le temps passé dans la tâche dépasse le « temps de vieillissement ». Ce temps est paramétrable à partir du menu Options. Il est nécessaire de recompiler les tâches après une modification.

↳ rencontre d'une instruction bloquante :

⇨ Wait, Delay

⇨ Beep, Edit

⇨ ClearFlash, FlashToRam, RamToFlash

↳ rencontre d'une instruction de saut, structure de boucle :

⇨ Call

⇨ Goto, Case

⇨ For...Next

⇨ Repeat...Until

⇨ While...End While

⇨ End Prog

L'instruction Jump permet d'effectuer un saut sans basculement.

En règle générale, une tâche courte permettra de traiter des événements plus rapides qu'une tâche longue.

4-3-2- Priorité des tâches

Chaque tâche utilisateur possède un niveau de priorité : priorité haute, priorité normale.

Le moniteur multitâche alloue deux tranches d'exécution : la tranche de priorité haute pour les tâches de priorité haute, une tranche de priorité normale pour les tâches de priorité normale.

L'enchaînement des tranches pendant l'exécution est le suivant :

| tranche priorité haute | tranche priorité normale | tranche priorité haute | tranche priorité normale | ...

↳ Tranche priorité haute :

Toutes les tâches de priorité haute sont effectuées l'une après l'autre dans cette tranche. Chaque tâche exécute ces instructions jusqu'à la condition de basculement (rencontre d'une instruction bloquante, vieillissement atteint ...).

Temps maxi exécution tranche priorité haute = nombre de tâches priorité haute * temps de vieillissement

Le « temps de vieillissement » est paramétrable à partir du menu Options et est identique pour les tâches de priorité haute et normale. Il est nécessaire de recompiler les tâches après une modification.

↳ Tranche priorité normale :

Les tâches de priorité normale sont effectuées l'une après l'autre dans cette tranche. Chaque tâche exécute ces instructions jusqu'à la condition de basculement (rencontre d'une instruction bloquante, vieillissement atteint ...).

Temps exécution tranche normale = temps tranche normale

Temps maxi exécution tranche normale = temps tranche normale + temps de vieillissement

Le « temps tranche normale » est paramétrable à partir du menu Options. Il est nécessaire de recompiler les tâches après une modification.

Si la durée d'exécution de toutes les tâches normales est inférieure à « temps tranche normale », toutes les tâches sont exécutées puis on enchaîne sur la tranche priorité haute.

Dans le cas contraire, le système redonne la main à la tranche priorité haute alors que toutes les tâches normales n'ont pas été exécutées. Elles seront traitées dans la tranche normale suivante.

Exemple :

```
T1, T2  : tâches priorité haute
T3, T4, T5, T6  : tâches de priorité normale
Temps de vieillissement = 2 ms
Temps tranche normale = 6 ms
L'exécution sera de la forme| T1,T2 | T3,T4,T5 | T1,T2 | T6,T3,T4 | T1,T2 |
T5,T6,T3 | ...
```

4-3-3- Gestion des tâches

Chaque tâche possède un mode de démarrage qui a été paramétré lors de sa création :

- Démarrage automatique : à chaque démarrage du SUPERVISOR, la tâche est lancée automatiquement.
- Démarrage manuel : la tâche n'est pas lancée automatiquement.

Un projet doit au moins contenir une tâche avec démarrage automatique. Il est conseillé d'avoir une seule tâche dans laquelle on écrit toute la partie initialisation de l'application et ensuite on lance les autres tâches.

On dispose de 5 instructions pour gérer les tâches :

- Run : lancement d'une tâche qui est à l'arrêt.
- Suspend : suspension (pause) d'une tâche en cours d'exécution
- Continue : reprise de l'exécution d'une tâche suspendue là où elle c'était arrêtée
- Halt : arrêt d'une tâche en cours d'exécution
- Status : indique l'état de la tâche

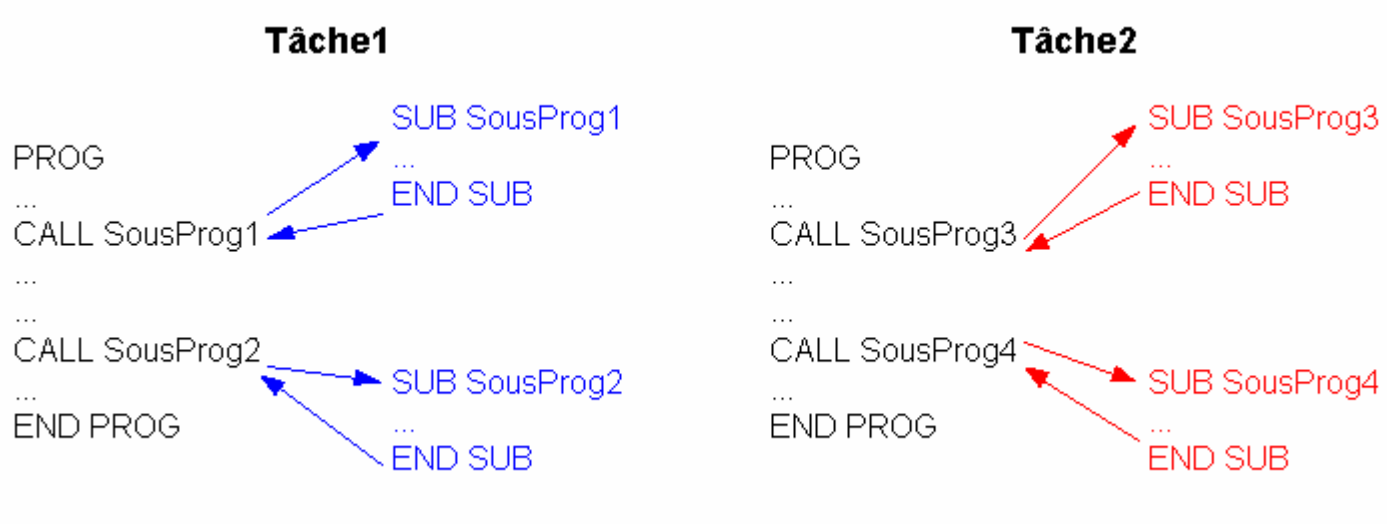
```
Exemple  :
Tâche Menus1                Tâche Menus2
Prog                        Prog
.....                       .....
Run Menus2                  If Key = @ESC Then Halt Menus2
Wait Status(Menus2)=0       .....
....                        End Prog
End Prog
```

Pour synchroniser les tâches entre elles, on peut utiliser les instructions d'événements Signal et Wait Event ou les variables globales.

```
Exemple :
ProcessEnable  : variable globale de type bit
Tâche Process1              Tâche Process2
Prog                        Prog
.....                       .....
ProcessEnable=1             Wait ProcessEnable=1
Wait ProcessEnable=0        .....
....                        ProcessEnable=0
End Prog                    .....
                            End Prog
```

4-3-4- Structure d'une tâche basic

Chaque tâche est constituée d'un programme principal défini par les mots clé PROG et END PROG et par des sous programmes sous forme de structure SUB .. END SUB. Par exemple :



A) Programme principal

Le programme principal d'une tâche peut appeler tous les sous programmes de la tâche mais ne peut pas appeler les sous programmes d'une autre tâche. Une tâche correspond à un fichier. Dans l'exemple précédent, la tâche1 peut appeler les sous-programmes SousProg1 et SousProg2 mais ne peut pas appeler les sous- programmes SousProg3 et SousProg4. Un sous programme d'une tâche peut également appeler un autre sous- programme de la même tâche.

Une seule structure PROG ... END PROG doit être utilisée par tâche. Elle peut apparaître à n'importe quel endroit.

Pendant l'exécution de la tâche, la rencontre du mot clé END PROG provoque un branchement de celle-ci en PROG.

B) Sous-programmes

Un sous-programme doit être déclaré par une procédure SUB...END SUB. Il peut être placé avant ou après le programme principal.

Pour appeler un sous-programme, vous devez utiliser la fonction CALL. Le sous-programme appelé doit être dans la même tâche.

Après l'appel du sous-programme, son exécution et son retour, la tâche continue automatiquement à l'instruction qui suit l'appel du sous-programme. Le système sort d'un sous programme lorsqu'il rencontre l'instruction END SUB ou EXIT SUB. Par exemple :

```
SUB Calcul
Resultat%=0
IF b%=0 THEN EXIT SUB ' Si b% est égal à zéro la division est impossible
Resultat%=a% DIV b%   ' Division
END SUB
```

Un sous-programme peut être appelé partout dans le programme mais ne peut s'appeler lui-même. Si des données sont utilisées dans le programme et dans des sous programmes, il est recommandé d'utiliser des variables bien spécifiques. En fait, toutes les variables peuvent être modifiées par un sous-programme, vous pouvez donc utiliser ces variables spécifiques dans chaque sous-programme en les affectant simplement avant l'appel. Par exemple :

```
...
Diviseur%=a%
Dividende%=b%
CALL Divise
IF Resultat!>10 THEN ...
...

SUB Divise
Resultat!=0
```

```
IF Diviseur%= 0 THEN EXIT SUB
   Resultat!= Dividende% / Diviseur%
END SUB
```

Suite au branchement au sous-programme, le multitâche temps réel passe automatiquement à la tâche suivante.

L'instruction ICALL permet également de se brancher à un sous-programme mais sans basculemnt automatique à la tâche suivante.

C) Branchement à une étiquette

L'instruction GOTO sert à effectuer un saut à une adresse représentée par une étiquette. Une étiquette est composée d'un nom terminé par ":". Si l'instruction GOTO se trouve à l'intérieur d'une structure de sous-programme SUB...END SUB, l'étiquette doit se trouver dans cette même structure.

Un branchement avec l'instruction GOTO peut être effectué indifféremment vers l'avant ou l'arrière du programme. Par exemple:

```
GOTO Label1
...
Label1:
...
```

Suite à un branchement, le multitâche temps réel passe automatiquement à la tâche suivante.

L'instruction JUMP permet également d'effectuer un saut à une étiquette sans passage à la tâche suivante.

D) Opérateurs

Les expressions sont composées d'opérateurs et d'opérandes. En Basic presque tous les opérateurs sont binaires, c'est à dire qu'ils utilisent deux opérandes. Les opérateurs n'utilisant qu'un opérande sont qualifiés d'unaires. Les opérateurs binaires utilisent les formes algébriques communes, par exemple A + B. Les opérateurs unaires s'écrivent toujours avant leurs opérandes, par exemple : `NOT A`. Dans des expressions complexes les règles de priorité suivantes enlèvent toute ambiguïté sur l'ordre des opérateurs.

Operators	Priority	Type
NOT	First (High)	Unary
*, /, DIV, MOD, AND, <<, >>	Second	Multiplication
+, -, OR, XOR	Third	Addition
=, <>, <, >, <=, >=	Fourth (Low)	Comparison

Trois règles fondamentales sur les priorités :

↳Un opérande placé entre deux opérateurs dont l'un est prioritaire, sera traitée avec l'opérateur prioritaire.

↳Un opérande placé entre deux opérateurs dont la priorité est la même, sera traitée avec l'opérateur de gauche.

↳Les expressions entre parenthèses sont évaluées séparément, les résultats des parenthèses sont considérés comme des opérandes.

Les opérateurs ayant la même priorité sont habituellement utilisés de gauche à droite.

Il est fortement conseillé d'utiliser les parenthèses pour séparer chaque expression afin de mettre en évidence les priorités. Par exemple :

```
IF ((INP(E1)=1) AND (FlagRun=1)) OR (InitOk=0) Then ...
```

a) Opérateurs arithmétiques

L'opérateur 'NOT' est un opérateur unaire. Les opérateurs + et - sont employés comme des opérateurs unaires ou des opérateurs binaires. Les autres sont uniquement binaires.

Un opérateur unaire ne possède qu'un paramètre.

Par exemple : NOT <Expression>

Un opérateur binaire demande deux paramètres.

Par exemple : <Expression1> * <Expression2>

b) Opérateurs binaires

Operator	Operation	Operand type	Type
+	Addition	Byte, Integer, Long integer or real	Operand type
-	Substraction	Byte, Integer, Long integer or real	Operand type
*	Multiplication	Byte, Integer, Long integer or real	Operand type
/	Division	Byte, Integer, Long integer or real	Operand type
DIV	Integer division	Byte, Integer, Long integer or real	Operand type
MOD	Modulus	Byte, Integer, Long integer or real	Operand type

c) Opérateurs unaires

Opérateur	Opération	Type de l'opérande	Type
+	Même signe	Octet, Entier, Entier long ou réel	Type de l'opérande
-	Inversion de signe	Octet, Entier, Entier long ou réel	Type de l'opérande

d) Opérateurs logiques

Opérateur	Opération	Type de l'opérande	Type
NOT	Négation binaire	Octet, Entier	Type de l'opérande
AND	ET logique	Octet, Entier	Type de l'opérande
OR	OU logique	Octet, Entier	Type de l'opérande
XOR	OU exclusif	Octet, Entier	Type de l'opérande
>>	Décalage à droite	Octet, Entier	Type de l'opérande
<<	Décalage à gauche	Octet, Entier	Type de l'opérande

e) Opérateurs sur bits

Operator	Operation	Operand type	Result type
+	Same sign	Byte, Integer, Long integer or real	Operand type
-	Invert sign	Byte, Integer, Long integer or real	Operand type

f) Opérateurs sur chaîne de caractères

Opérateur	Opération	Type de l'opérande	Type
+	Concaténation	Chaîne de caractères	Chaîne de caractères

g) Opérateurs de relation

Opérateur	Opération	Type de l'opérande	Type
=	Egal	Octet, Entier, Entier long, Réel ou chaîne de caractères	Bit
<>	Différent de	Octet, Entier, Entier long, Réel ou chaîne de caractères	Bit
<	Inférieur à	Octet, Entier, Entier long, Réel ou chaîne de caractères	Bit
>	Supérieur à	Octet, Entier, Entier long, Réel ou chaîne de caractères	Bit
<=	Inférieur ou égal à	Octet, Entier, Entier long, Réel ou chaîne de caractères	Bit
>=	Supérieur ou égal à	Octet, Entier, Entier long, Réel ou chaîne de caractères	Bit

E) Tests

a) Tests simples

Les instructions conditionnelles sont un moyen pratique d'exécuter ou non un groupe d'instructions selon qu'une condition est vraie ou fausse. Les deux syntaxes possibles de l'instruction IF sont :

```
IF <Expression> THEN <Instruction1> [ELSE <Instruction2>]
```

ou

```
IF <Expression> THEN
    <Instruction1>
    ...
[ELSE
   <Instruction2>
    ...]
END IF
```

<Expression> doit être une valeur de type bit. Si <Expression> est vraie alors <Instruction1> et les instructions suivantes sont exécutées. Si <Expression> est fausse <Instruction2> et les instructions suivantes sont exécutées. Dans la seconde forme de syntaxe, une instruction seulement est exécutée pour chaque condition, toutes les instructions sont sur la même ligne et l'on n'emploie pas de END IF. Il est possible d'imbriquer des instructions IF, mais une terminaison ELSE se réfère toujours à l'instruction IF la plus proche.

b) Tests multiples

Les tests multiples sont obtenus avec l'instruction CASE.

La syntaxe de l'instruction CASE est décrite comme telle :

```
CASE <Expression> [ GOTO | CALL ] <Identif. Sous-prog. 1> [ { , <Identif. Sous-prog. 2> } ]
```

<Expression> doit être de type octet ou entier. Avec cette instruction, des sous-programmes seront appelés selon la valeur d'<Expression>. Pour <Expression>=1 le premier sous-programme est appelé, pour <Expression>=2 le second programme 2 est appelé... Par exemple :

```
INPUT #1,Choix%              ' Lecture sur le lien série
CASE Choix% GOTO Choix1, Choix2, Choix3
GOTO Fin                     ' Choix%=0 ou Choix%>3
Choix1:                      ' Traitement choix1
  ....
  GOTO Fin
Choix2:                      ' Traitement choix2
  ....
  GOTO Fin
Choix3  :                    ' Traitement choix3
  ....
```

```
  GOTO Fin
  ....
Fin:
```

c) Les boucles

Si le nombre de fois à effectuer la boucle est déjà connu en écrivant le programme, il est recommandé d'utiliser la structure FOR, dans un autre cas, utiliser les structures WHILE ou REPEAT.

L'instruction FOR

L'instruction FOR permet l'exécution répétée d'une ou plusieurs instructions tandis qu'une variable (index) parcourt pas à pas un domaine de valeurs.

La syntaxe de l'instruction FOR est la suivante :

```
FOR <Compteur>=<Début> TO <Fin> [STEP <Incrément>]
<Instructions>
NEXT <Compteur>
```

<Compteur>doit être une variable locale de type octet , entier ou entier long. <Début>, <Fin> et <Incrément> sont des expressions dont le type est compatible avec celui de <Compteur> . Les expressions <Début>, <Fin> et <Incrément> sont calculées avant l'exécution de la boucle.

La valeur <Début> est affectée à <Compteur> au début de la boucle. A chaque boucle la valeur <Incrément> est ajoutée à <Compteur>, jusqu'à ce que <Compteur> atteigne ou dépasse <Fin>, et la boucle s' arrête.

Par exemple :

```
FOR a%=0 TO 15
OUT(IO1)=1<<a%
NEXT a%
```

A chaque NEXT, le multitâche temps réel passe automatiquement à la tâche suivante.

L'instruction WHILE

L'instruction WHILE permet l'exécution répétée d'une ou plusieurs instructions selon la valeur d'une expression.

La syntaxe de l'instruction WHILE est la suivante :

```
WHILE <Expression> DO
<Instructions>
END WHILE
```

Dans cette instruction, si la valeur de <Expression> est FAUX avant la structure WHILE, on n'entre pas dans la boucle. Tant que <Expression> est VRAIE <Instructions> sont exécutées.

A chaque END WHILE, le multitâche temps réel passe automatiquement à la tâche suivante.

L'instruction REPEAT

L'instruction REPEAT permet l'exécution répétée d'une ou plusieurs instructions selon la valeur d'une expression.

La syntaxe de l'instruction REPEAT est la suivante :

```
REPEAT
<Instructions>
UNTIL <Expression>
```

Dans cette instruction, si <Expression> est VRAIE avant la structure REPEAT, la boucle est effectuée une fois. <Instructions> sont exécutées jusqu'à ce que <Expression> soit vraie.

A chaque UNTIL, le multitâche temps réel passe automatiquement à la tâche suivante.

4-3-5- Structure d'une tâche ladder

Elle se présente sous forme graphique et se programme par des réseaux qui contiennent des contacts, des bobines, des compteurs et des temporisations.

On peut également insérer n'importe quelle instruction basic dans les « contacts libres » et dans les « bobines libres ».

Lors de la compilation, la tâche ladder est traduite en tâche basic. Celle-ci est visualisable à partir d'un éditeur quelconque de Windows : fichier « NomTâcheLadder.tsk ».

4-3-6- Structure de la tâche événementielle

Cette tâche spéciale permet de gérer jusqu'à 16 événements standards: 7 entrées automates, 8 entrées capture, 1 timer.

Par projet, on ne peut en déclarer qu'une seule : lors de la création de la tâche, on l'affecte avec un mode de démarrage événementiel.

A) Configuration des événements

A chaque démarrage du SUPERVISOR, aucun événement n'est configuré. Celle-ci se fait à partir d'une tâche basic normale (ex : tâche d'initialisation) grâce à l'instruction MODIFYEVENT.

Syntaxe : MODIFYEVENT (<Condition>,<Seuil compteur1>, <Seuil compteur2>, <Masque>, <Durée>)

Limites : <Durée> : de 10ms à 30000ms

Types acceptés : <Condition> : Entier

<Seuil Compteur1> : Entier

<Seuil Compteur2> : Entier

<Durée> : Entier

Description : Elle permet de configurer les événements souhaités.

Remarques : <Condition> :

↳ Bits 0...7 : activation des entrées n° 1 à n° 8 de la carte entrée. Un front montant générera l'événement. L'entrée tient compte du filtre et de l'inversion paramétrés dans l'écran de configuration de la carte.

↳ Bit 8 : Seuil du compteur 1 atteint

↳ Bit 9 : Seuil du compteur 2 atteint

↳ Bit 10 : SDOEvent

↳ Bit 11 : PDOEvent

↳ Bit 12 : Base de temps

<Durée> :

Durée de la base de temps entre 10 ms et 30000 ms. Si la base de temps n'est pas utilisée, la valeur de durée rentrée ne sera pas traitée.

Après l'affectation de la configuration, la tâche événementielle est exécutée dès qu'au moins un événement est détecté. Le temps maxi entre l'apparition de l'événement et son traitement est égal au « temps de vieillissement » d'une tâche.

Si l'on désire par la suite modifier la configuration des événements, l'instruction MODIFYEVENT doit être traitée dans une tâche basic normale ou dans la tâche événementielle à condition qu'elle soit placée après GETEVENT.

B) Lecture des événements détectés

L'instruction GETEVENT permet de consommer et de lire quels événements ont été détectés.

Syntaxe : <Variable>=GETEVENT

Avec <Variable> : de type entier avec la même définition des bits que <Masque> de MODIFYEVENT.

Chaque bit associé à un événement est à l'état 1 si l'événement a été détecté.

Si un nouvel événement apparaît pendant l'exécution de la tâche événementielle, il est mémorisé et traité dès que possible.

C) Dévalidation des événements

Elle se fait en utilisant MODIFYEVENT(0,0).

D) Mises en garde

Les instructions RUN, HALT, SUSPEND, CONTINUE, STATUS n'ont aucun effet sur la tâche événementielle.

Elle ne rend pas la main aux autres tâches tant qu'elle n'a pas été entièrement exécutée. Il ne faut donc pas qu'elle soit très longue.

En aucun cas cette tâche ne doit comporter d'instructions bloquantes (ex : WAIT, ...).

Elle ne doit pas être rebouclée : l'instruction END PROG doit être rencontrée en fin de tâche afin de relancer la détection d'événements.

Si l'instruction MODIFYEVENT est utilisée dans la tâche événementielle, elle peut altérer tout nouvel événement apparu pendant l'exécution de celle-ci.

E) Exemple

```
Tâche Init
      PROG
      ....
      MODIFYEVENT(0183H,1000)      'événements E1, E2, base de temps 1s,
      .....                        'Capture1 carte d'axe n°1
      END PROG

Tâche EVENEM
      PROG
      Event%=GETEVENT
      IF Event%.1=1 THEN    'événement E1
         .
         .
      END IF
      IF Event%.2=1 THEN    'événement E2
         .
         .
      END IF
      IF Event%.8=1 THEN    'événement base de temps
         .
         .
      END IF
      IF Event%.9=1 THEN    'événement capture 1
         .
         .
         capture1(…         'relance la capture
      END IF
      END PROG
```

5- PROGRAMMATION DE L'AUTOMATE

5-1- Tâche pseudo-basic

5-1-1- Entrées/Sorties logiques

A) Lecture des entrées

La fonction INP est utilisée pour lire 1 bit, INPB un bloc de 8 bits et INPW un bloc de 16 bits.

Les syntaxes sont : INP(<NomEntrée>), INPB(<NomEntrée>), INPW(<NomEntrée>)

<NomEntrée> doit représenter l'identificateur d'une entrée, d'un bloc de 8 entrées ou d'un bloc de 16 entrées. Cet identificateur peut être soit un nom symbolique utilisé dans le module de configuration ou le nom par défaut de cette même entrée. Le type de données retourné est :

- Bit pour une entrée

- Octet pour un bloc de 8 entrées

- Entier pour un bloc de 16 entrées

Par exemple :

```
A~ = INP(Capteur)     'lecture d'une entrée
B1# = INPB(Bloc1)     'lecture du premier bloc de 8 entrées
B2# = INPB(Bloc2)     'lecture du deuxième bloc de 8 entrées
C%= INPW(A)           'lecture d'un bloc de 16 entrées
```

B) Ecriture des sorties

La fonction OUT est utilisée pour écrire 1 bit , OUTB un bloc de 8 bits et OUTW un bloc de 16 bits .

Les syntaxes sont : OUT(<NomSortie>), OUTB(<NomSortie>), OUTW(<NomSortie>)

<NomSortie>doit représenter l'identificateur d'une sortie, d'un bloc de 8 sorties ou d'un bloc de 16 sorties. Cet identificateur peut être soit un nom symbolique utilisé dans le module de configuration ou le nom par défaut de cette même sortie. Le type de données utilisé est :

- Bit pour une sortie

- Octet pour un bloc de 8 sorties

- Entier pour un bloc de 16 sorties

Par exemple :

```
OUT(Verin)=On         'écriture d'une sortie
OUT(LAMP)=Defaut.5
OUTB(Data)=00110000b  'écriture d'un bloc de 8 sorties sous forme binaire
OUTW(B)=0FFFFh        'écriture d'un bloc de 16 sorties
                      'sous forme hexadécimal
```

C) Lecture des sorties

Toutes les sorties peuvent également être lues. La valeur lue est la dernière valeur écrite. Cette caractéristique est très utile quand plus d'un programme utilise le même bloc de sorties. Donc, il est possible d'écrire seulement les sorties désirées dans une opération sans changer les autres.

Par exemple :

Pour mettre à 1 le quatrième bit d'un bloc de 8 bits nommé Bloc1 :

```
OUTB(Bloc1)=OUTB(Bloc1) OR 00001000b      'mise à 1 du quatrième bit
                                          'd'un bloc de 8 bits nommé Bloc1
```

D) Attente d'un état

Il est possible d'attendre un changement d'état sur une entrée grâce à l'instruction WAIT.

La syntaxe est : WAIT <Condition>

La fonction WAIT est utilisée pour attendre une condition de changement durant une exécution normale. L'exécution est stoppée aussi longtemps que la condition est fausse. Quand l'état devient vrai, l'exécution continue. Cette fonction est très utile pour attendre la fin des mouvements ou une butée logicielle...

Exemple :

```
WAIT (Lim_S(Coupe))=On              'Attente erreur de butée soft
Stop(Coupe)                         'Arrêt de l'axe
WAIT (Inp(Bouton Depart))=On        'Attente bouton de départ pressé
```

E) Test d'un état

Il est possible de tester l'état d'une entrée grâce à l'instruction IF…THEN…ELSE.

La syntaxe est : IF (<Condition>) THEN <Action1> ELSE <Action2>

La structure IF…THEN…ELSE est utilisée pour tester une condition à un instant donné. La validation de la <Condition> permet d'exécuter l'<Action1>. Dans le cas contraire, c'est l'<Action2> qui est exécuter.

Exemple :

```
IF (Inp(Bouton Depart)=On) THEN     'Test de l'état de l'entrée
                                    'Bouton Depart
           Out(LedMarche)=On        'Action associée à Bouton Depart=On
           RUN Cycle
ELSE
           Out(LedMarche)=Off       'Action associée à Bouton Depart=Off
           HALT Cycle
ENDIF
```

5-1-2- Temporisations

A) Attente passive

La fonction DELAY est utilisée pour établir une attente passive. Sa syntaxe est :

DELAY <Durée>

<Durée> est un entier exprimé en milliseconde. Il est recommandé d'utiliser cette fonction pour une longue attente passive car le programme en attente ne prend pas de temps processeur.

Avec cette fonction, le programme attend la durée indiquée.

Par exemple:

```
Debut:
WAIT Inp(Start)=ON
...
DELAY 5000            ' Délai de 5 secondes
...
GOTO Debut
```

B) Attente active

5-1-3- TIME

La variable globale interne TIME peut être utilisée pour établir des attentes actives. C'est un entier long qui représente le nombre de millisecondes écoulées depuis la dernière mise sous tension. Cette variable peut donc être utilisée comme base de temps. Elle convient en particulier au machine qui sont sous-tension moins de 24 jours. En effet à la mise sous-tension, TIME est initialisé à 0. Au-delà de 24 jours, la variable atteint sa valeur maximum 2^31 et passe ensuite à

2^-31. Cette transition appelée débordement peut provoquer dans certain cas des erreurs de temporisations. Il est donc préférable d'utiliser la variable globale TIMER.

Par exemple :

```
FinDelay& = TIME+5000        'chargement d'une temporisation de 5s
WHILE TIME<FinDelay& DO
          ...                'Traitement pendant 5s
END WHILE
FinTimeOut& = TIME+200
WAIT (Inp(Capteur)=On) Or (Time>FinTimeOut&)     'Attente d'un capteur ou
                                                 'd'un time-out de 200ms
```

5-1-4- TIMER

La variable globale interne TIMER peut être utilisée pour établir des attentes actives. C'est un réel qui représente le nombre de millisecondes écoulées depuis la dernière mise sous tension. Cette variable peut donc être utilisée comme base de temps. Elle convient en particulier aux machines qui sont toujours sous-tension. La partie entière représente les secondes et la partie décimale (3 chiffres après la virgule) représente les millisecondes.

Par exemple :

```
FinDelay! = TIMER+5.250      'Chargementd'une temporisation de 5.25s
WHILE TIMER<FinDelay! DO
          ...                'Traitement pendant 5.25s
END WHILE
FinTimeOut! = TIMER+0.200
WAIT (Inp(Capteur)=On) Or (TIMER>FinTimeOut&)    'Attente d'un capteur ou
                                                 'd'un time-out de 200ms.
```

5-1-5- Evénements

A) Evénements

Dans un système multitâche, les mécanismes d'événements sont utiles pour la communication entre tâches. La gestion d'événements peut aussi fournir des fonctions de contrôle de process. Plusieurs programmes peuvent attendre ou envoyer le même événement. Le langage de programmation offre deux mécanismes pour déclarer ces opérations d'événements entre tâches.

5-1-6- Signal ou Diffuse et Wait Event

↳ Pour envoyer un même événement à un seul programme, il existe la fonction SIGNAL. Pour l'envoyer à tous les programmes, il existe la fonction Diffuse.

Syntaxe : SIGNAL <NomEvenement> ou DIFFUSE <NomEvenement>

Le <NomEvenement> peut être n'importe quel nom qui n'est pas un mot-clé mais doit être utilisé au moins une fois dans une fonction d'attente d'événement.

SIGNAL enverra l'événement au premier programme qui l'attend alors que Diffuse l'enverra à tous ceux qui l'attendent.

↳ L'instruction WAIT EVENT est utilisée pour attendre cet événement.

La syntaxe de l'instruction WAIT EVENT est :

WAIT EVENT <NomEvenement>

Après l'instruction WAIT EVENT, l'exécution du programme est arrêtée et continuera dès que l'événement sera reçu.

```
Exemple  avec SIGNAL et WAIT EVENT:
'Tâche maître                          'Tâche esclave
PROG                                   PROG
...
RUN TacheEsclave                       Debut:
...                                    ...
```

```
WAIT Inp(CycleStart)=On                    ...
...                                        ...
SIGNAL Start                               WAIT EVENT Start
...                                        GOTO Debut
...                                        END PROG
WAIT Inp(CycleStop)=On
HALT TacheEsclave
...
END PROG
```

Dans cet exemple, il y a une tâche maître qui contrôle l'exécution d'une tâche esclave. La tâche maître attend le bouton start. Quand la condition est vraie, la tâche maître signale le démarrage à la tâche esclave en envoyant l'événement start. Si le bouton Stop est pressé, la tâche maître arrête la tâche esclave. La tâche esclave attend l'événement start. Quand cet événement est reçu, la tâche esclave exécute son cycle et se remet de nouveau en attente.

```
Exemple  avec DIFFUSE et WAIT EVENT:
'Tâche maître                              'Tâche esclave
PROG                                       PROG
...
RUN TacheEsclave                           Debut:
...                                        ...
WAIT Inp(CycleStart)=On                    ...
...                                        ...
DIFFUSE Start                              WAIT EVENT Start
...                                        GOTO Debut
...                                        END PROG
WAIT Inp(CycleStop)=On
HALT TacheEsclave
...
END PROG
```

Cette exemple est le même que précédemment mais utilise l'instruction DIFFUSE.

5-1-7- Wait

Le deuxième mécanisme permet d'attendre un événement provenant d'une variable globale ou d'une entrée. L'instruction Wait <Expression> n'autorise pas l'exécution de la tâche tant que l'expression n'est pas valide. Le déblocage de la tâche ne se fera que par une affectation de <Expression> dans une autre tâche. Cette affectation correspond à l'envoi de l'événement.

L'exemple ci-dessous est le même que pour les instructions Signal – Wait Event en utilisant le mécanisme Wait :

```
'Tâche maître                              'Tâche esclave
WAIT Inp(CycleStart)=On                    ...
...                                        ...
VariableSignal=1                           WAIT VariableSignal=1
...                                        VariableSignal=0
```

L'utilisation de ce mécanisme nécessite un temps d'exécution au système supérieur au mécanisme précédent. En effet, il est nécessaire de dévalider la variable globale d'événement. Cette affectation implique un temps supplémentaire d'exécution.

5-1-8- Compteurs

A) Compteurs

Le SUPERVISOR possède 2 compteurs 16 bits. Chaque entrée de la carte SIO peut être affectée à un compteur.

' Attention :

- Lorsque le compteur atteint sa valeur maxi, il repasse à 0 au prochain front (valeur maxi : 65535).

5-1-9- Configuration

L'instruction SETUPCOUNTER permet de configurer le compteur.

Syntaxe : SETUPCOUNTER(<Compteur>,<Entrée>,<Inversion>,<Filtre désactivé>)

<Compteur> : Numéro du compteur (1 ou 2)

<Entrée> : Numéro de l'entrée correspondante sur la carte SIO

<Inversion> : Choix du front : 0 pour front montant, 1 pour front descendant

<Filtre désactivé> 0 pour filtre de 2 ms, 1 pour sans filtrage,.

5-1-10- Remise à zéro

L'instruction CLEARCOUNTER permet d'initialiser à 0 le compteur.

Syntaxe : CLEARCOUNTER(<Compteur>)

<Compteur> : Numéro du compteur (1 ou 2)

5-1-11- Lecture

L'instruction COUNTER_S permet de lire le compteur.

Syntaxe : <Variable>=COUNTER_S(<Compteur>)

<Variable> : entier compris entre 0 et 65535

<Compteur> : Numéro de compteur (1 ou 2)

5-1-12- Fonctions automate étendues

A) Fonctions automate étendues

Présentation

Les fonctions PLC (automate étendues) permettent d'intégrer le fonctionnement d'un PLC dans un programme basic multitâches. Ainsi, on garantit alors que les entrées sorties gérées par cette tâche sont traitées comme sur un PLC. Les entrées sont mémorisées dans des bits images avant d'être traitées, les sorties à modifiées sont mémorisées avant d'être affectées réellement.

Utilisation du PLC

Le PLC utilise des tableaux pour mémoriser les états des entrées/sorties. Deux tableaux d'entier long pour les entrées, deux tableaux d'entier pour les sorties.

La fonction PlcReadInputs lit l'état des entrées, après avoir mémorisé l'état précédent, pour permettre une détection des fronts.

Les fonctions PlcInp, PlcInpb, PlcInpw, PlcInpPe et PlcInpNe permettent de lire l'état des entrées et de détecter les fronts.

Les fonctions PlcOut, PlcOutB et PlcOutW modifie les bits images des sorties.

La fonction PlcWriteOutputs écrit l'état des bits images sur les sorties physiques.

Exemple

Dans cet exemple, les blocs de sorties sont utilisés pour représenter le comptage des fronts montant et descendant d'une entrée

```
PROG
‘ on utilise toutes les sorties
  Masque[1]=0FFFFh
  Masque[2]=0FFFFh
‘ on initialise le PLC
  PlcInit(Entrees,EntreesOld,Sorties,Masque)
  Repeat
   ‘ lecture des entrées
    PlcReadInputs
   ‘ détection des fronts montants
    If PlcInpPe(I1) Then
      PlcOutB(JL)=PlcOutB(JL)+1
    End If
  ‘ détection des fronts descendants
    If PlcInpNe(I1) Then
      PlcOutB(JH)=PlcOutB(JH)+1
    End If
‘ écriture des sorties
PlcWriteOutputs
  Until False
END PROG
```

5-2- Tâche ladder

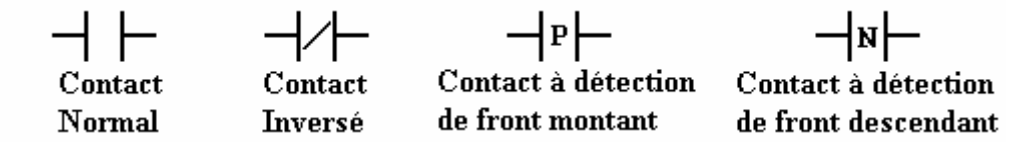
5-2-1- Présentation

Chaque tâche ladder se présente sous forme d'une série de réseaux dont le nombre est limité à 50 (par tâche). Un réseau est la combinaison de plusieurs bobines avec une seule expression. Ce qui implique que toutes les bobines d'un réseau ont la même expression. Un réseau peut recevoir un maximum de 5 bobines ou contacts en parallèle et de 10 contacts en série.

Attention : Une tâche ladder est automatiquement traduite en basic de façon interne. Il est déconseillé d'écrire une tâche ladder longue ou complexe afin d'éviter les fortes dégradations du temps de cycle et les limitations de la traduction basic.

5-2-2- Contacts, Bobine, Blocs

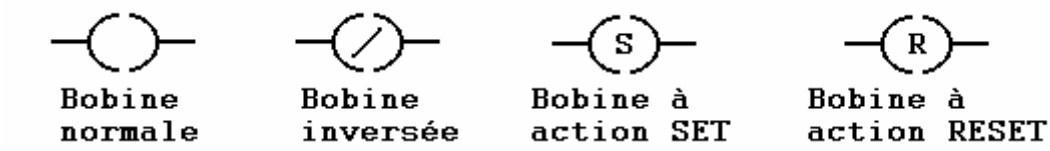
5-2-3- Contacts



On peut attribuer à chacun de ces contacts une entrée, une sortie, une variable globale ou une variable locale de type bit. L'affectation d'une variable système ne peut se faire que pour un contact normal ou inversé.

- Contact Normal : L'état du contact dépend directement de l'état de la variable qui lui est associée.
- Contact Inversé : L'état du contact est l'opposé de l'état de la variable qui lui est associée.
- Contact à détection de front montant : L'état du contact est vrai, lorsque la variable qui lui est associé passe de l'état faux à l'état vrai.
- Contact à détection de front descendant : L'état du contact et vrai, lorsque la variable qui lui est associé passe de l'état vrai à l'état faux.

5-2-4- Bobines

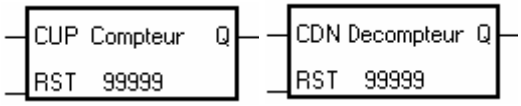


On peut attribuer à chacune de ces bobines une sortie, une variable globale ou une variable locale de type bit. L'affectation d'une entrée ou d'une variable système est impossible.

- Bobine normal : L'état de la bobine est directement celui de l'expression qui lui est associée.
- Bobine inversée : L'état de la bobine est l'opposé de l'expression qui lui est associée.
- Bobine à action SET : L'état de la bobine passe et reste à vrai dès que l'expression associée est vraie. La variable garde cette valeur jusqu'à un ordre inverse donné par une bobine de type "RESET".

↳ Bobine à action RESET : L'état de la bobine passe et reste à faux dès que l'expression associée est vraie. La variable garde cette valeur jusqu'à un ordre inverse donné par une bobine de type "SET".

5-2-5- Compteurs / Décompteurs



Les compteurs ou décompteurs sont munis de deux entrées et d'une sortie. Chaque compteur et décompteur est caractérisé par un nom et une valeur de présélection. Cette valeur de présélection peut-être une valeur fixe ou une variable globale. L'intérêt de la variable globale est de permettre une modification des caractéristiques à tous moments. Il est obligatoire d'associer la sortie d'un compteur ou décompteur avec une bobine, même si cette dernière n'est pas utilisée.

↳ Compteur : CUP est l'entrée de comptage. Sur une détection d'un front montant sur son entrée, elle incrémente d'une unité la variable compteur associée au compteur. Lorsque la valeur de la variable compteur est supérieure ou égale à la valeur de présélection, la sortie Q du compteur passe à un état vrai. L'entrée RST est prioritaire sur l'entrée CUP. Lorsqu'elle est vraie, elle permet d'initialiser la variable compteur en la forçant à 0. A l'état initial, la variable compteur vaut 0.

↳ Décompteur : CDN est l'entrée de décomptage. Sur une détection d'un front montant sur son entrée, elle décrémente d'une unité la variable compteur associée au décompteur. Lorsque la valeur de la variable compteur est inférieure ou égale à 0, la sortie Q du compteur passe à un état vrai. L'entrée RST est prioritaire sur l'entrée CDN. Lorsqu'elle est vraie, elle permet d'initialiser la variable compteur en la forçant à la valeur de présélection. A l'état initial, la variable compteur est égale à la valeur de présélection.

Les variables de comptage des compteurs ou décompteurs sont accessibles dans la tâche sous la forme : <Nom du bloc> + <&>.

```
Exemple  : Nom du bloc  : Compteur1
           Variable locale associée au compteur  : Compteur1&
```

5-2-6- Temporisateurs

Les temporisateurs sont du type retard à l'enclenchement (TON). Le retard à l'enclenchement est directement programmable et peut-être de deux formes différentes : valeur fixe ou variable globale. Les temporisateurs sont réalisés à partir de l'instruction TIMER, qui permet de prendre en compte les applications en route plus de 24 jours. Il est obligatoire d'associer la sortie d'une temporisation avec une bobine, même si cette dernière n'est pas utilisée.

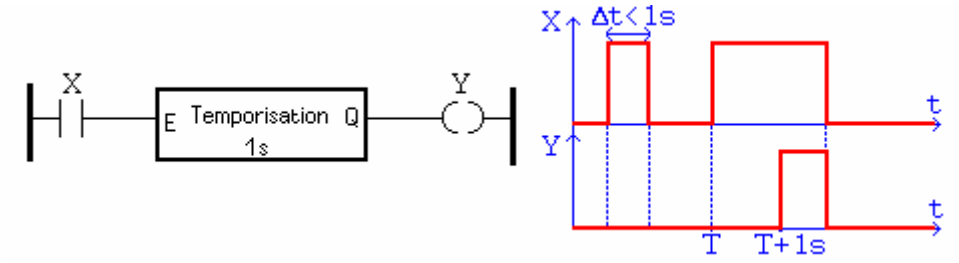
La variable associée à la temporisation est accessible dans la tâche sous la forme : <Nom du bloc> + <TVAL!>. Cette variable indique la durée écoulée depuis la mise en fonction du bloc.

```
Exemple  : Nom du bloc  : Tempo1
           Variable locale associée à la temporisation  : Tempo1Val!
```

↳ Réalisation d'une temporisation de type retard à l'enclenchement :

C'est la plus simple à réaliser. Il suffit de mettre l'expression de déclenchement sur l'entrée E. La sortie Q de la temporisation donne le résultat.

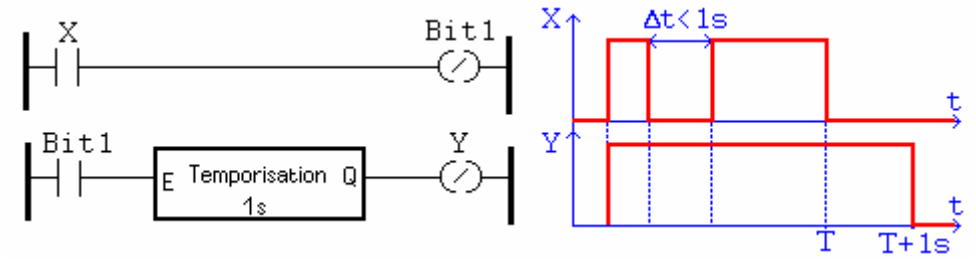
Exemple :



↳ Réalisation d'une temporisation de type retard au déclenchement :

Pour réaliser ce type de temporisation, l'expression de déclenchement et la bobine de sortie doivent être complémentées comme dans l'exemple ci-dessous.

Exemple :



5-2-7- Contact libre et Bobine libre

Les blocs libres ont été ajoutés pour palier aux limitations du ladder. Deux types de blocs existent : le contact libre et la bobine libre.

↳ Le contact libre : Ce type de contact est dédié aux tests de variables de type supérieur au bit (Ex :octet, Entier, Entier long, réel ou chaîne de caractère). Elle trouve son intérêt avec les instructions de mouvement (Ex : MOVE_S(X),…). L'édition d'un tel contact ne nécessite d'entrer que la condition avec les parenthèses nécessaires. La condition peut-être composée de plusieurs tests. (Ex : (MOVE_S(X)=1) And (POS(X)>2000))

↳ La bobine libre : Ce type de bobine est dédié à l'exécution d'instructions complexes et en particulier les instructions de mouvements. Elle sert aussi à l'affectation de variables dont le type est supérieur au bit (Ex :octet, Entier, Entier long, réel ou chaîne de caractère). L'édition d'une telle bobine ne nécessite d'entrer que l'action à exécuter.(Ex : STTA(X=100,Y=150))

' Attention : N'utilisez pas d'instructions bloquantes (Ex : MOVA, WAIT,…) qui affecteraient l'évolution de la tâche ladder.

5-2-8- Bit systèmes

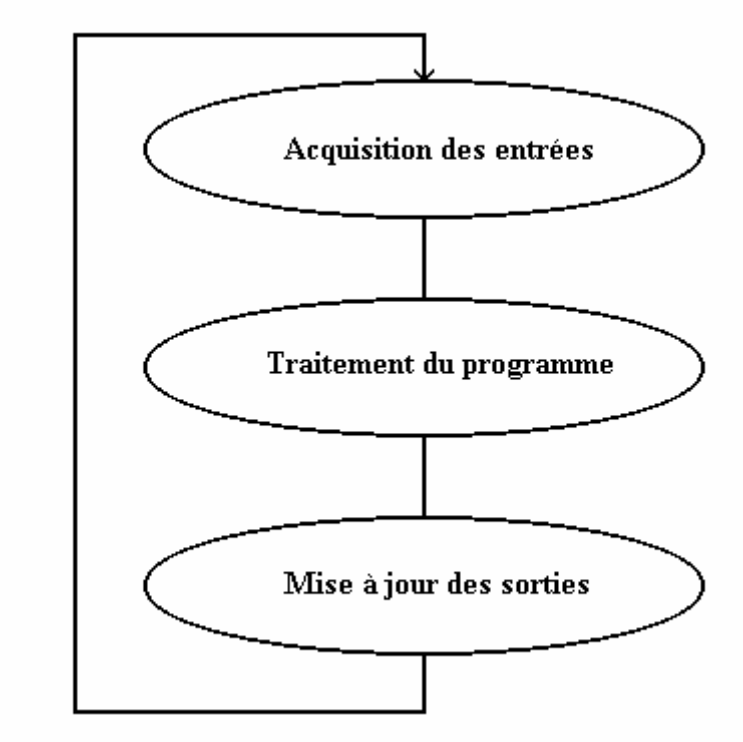
↳ Bit d'initialisation (initialisation) : Ce bit est 1 lors de la première scrutation de la tâche ladder.

↳ Bit clignotant 0.5s : bit dont le changement d'état est cadencé par une horloge de 0.5s.

↳ Bit clignotant 1s : bit dont le changement d'état est cadencé par une horloge de 1s.

5-2-9- Architecture de la tâche

L'exécution de la tâche ladder se fait suivant l'architecture suivante :



Ce système assure qu'une variable d'entrée n'évolue pas pendant un cycle et que les sorties ne sont mises à jour qu'une fois dans le cycle.

Le fonctionnement du multitâche autorise les tâches ladder à être interrompues à tout moment du cycle d'exécution.

6- PROGRAMMATION DES PORTS DE COMMUNICATION SERIAL1 / SERIAL2

6-1- Introduction

Le SUPERVISOR est équipé en standard d'un port de communication RS232 en Serial1. Il sera utilisé pour effectuer le téléchargement de la configuration, des variables, des tâches... entre le PC et le SUPERVISOR.

Un deuxième port série optionnel RS 232 ou RS 485 s'installe en Serial2.

Ces 2 ports peuvent être gérer à partir des tâches basic. On peut les ouvrir, les lire, y écrire des données, les fermer.

Les fonctions de conversion MKI$, CVI, MKL$, CVL... pourront être utilisées pour optimiser le temps de codage ou de décodage des messages.

6-2- Ouverture d'un port

L'instruction OPEN permet l'ouverture du port.

Syntaxe : OPEN <Port de communication> AS # <N°Comm>

<Port de communication> est une chaîne de caractères qui identifie un nom de port physique de communication et ses paramètres. <N°Comm> est un numéro de canal servant à identifier le port de communication ouvert et sera utilisé par les fonctions de lecture, d'écriture et de fermeture.

La chaîne de caractères <Port de communication> peut être décomposée en cinq parties :

"SERIAL2:[Vitesse [, Données [, Parité [, Stop]]]]"

↳SERIAL : port physique (1 ou 2)

↳Vitesse : Vitesse de communication (150, 300, 600, 1200, 2400, 4800 ou 9600 bauds)

↳Données : Nombre de bits de données (7 ou 8)

↳Parité : E paire, O impaire, M masquée, S espace, N pas de parité

↳Stop : Nombre de bits de stop (1 ou 2)

Les paramètres de vitesse, données, parité, stop sont optionnelles. Lors de la compilation des tâches, s'ils ne sont pas spécifiés, le système prend par défaut ceux présents dans l'écran de configuration du SPL (accessible avec un double click sur le SUBD de la Serial).

Exemple :

```
OPEN «SERIAL2  :9600,8,N,1" AS #1  ' ouverture du port SERIAL2
```

Quand un port de communication a été ouvert par une tâche, il ne peut pas l'être à nouveau par une autre tant qu'il n'a pas été fermé. Cependant un port de communication ouvert peut être lu ou écrit par toutes les tâches.

Un port doit être ouvert avant de pouvoir lire ou écrire des données.

Il est conseillé de réserver Serial1 uniquement au téléchargement car sinon à chaque transfert, on est obligé de débrancher le câble du périphérique « basic » et de mettre celui du PC....

Si Serial1 est utilisé dans le basic, avant chaque nouveau transfert, effectuez la commande « Arrêter les touches » à partir du menu de communication.

6-3- Lecture de données

⇨ Buffer de réception

Chaque port de communication série possède un buffer ou tampon de réception de 500 octets.

Si le buffer est plein (500 caractères reçus et non lus), les nouveaux caractères viendront effacer les premiers reçus.

L'instruction CLEARIN vide le buffer.

L'instruction CARIN retourne le nombre de caractères présents dans le buffer.

Pour lire les données, on dispose de deux instructions INPUT et INPUT$.

L'instruction INPUT attend des données et affecte les données reçues à des variables.

La syntaxe est la suivante : INPUT #<N°Comm>, <Variable> [{, <Variable> }]

<N°Comm> est le n° de canal spécifié dans l'instruction OPEN. Les données venant du port de communication doivent arriver dans l'ordre de la liste des variables et avec le même type. Par exemple :

```
OPEN "SERIAL1:"  AS #1       ' Ouverture du  port serial 1
                             ' affecté au canal 1
INPUT #1, B$, C%             ' Lecture d'une chaîne de caractère
                             ' et d'une valeur entière
CLOSE #1                     ' Fermeture du  port de communication
```

Pour les variables numériques de la liste, le premier caractère n'étant pas un espace est considéré comme le début d'un nombre. La fin du nombre est notifiée par un espace, une virgule ou une fin de ligne. Une ligne blanche est considérée comme un zéro. Si la donnée numérique n'est pas valide, la variable prend zéro pour valeur.

Pour les variables de type chaîne de caractères, le premier caractère n'étant pas un espace est considéré comme le début de la chaîne. La fin de la chaîne est notifiée par une virgule, un espace ou une fin de ligne. Une ligne blanche est considérée comme une chaîne de caractères de longueur nulle.

La seconde instruction, INPUT$ lit un nombre spécifié de caractères sur un port de communication et les stocke dans une chaîne de caractères. La syntaxe est la suivante :

<Variable de type chaîne de caractères> = INPUT$ (#<N°Comm>, <Nombre de caractères>)

Ces deux instructions de lecture sont bloquantes pour la tâche tant qu'elles n'ont pas reçues le nombre de caractères souhaité.

6-4- Ecriture de données

⇨ Buffer de transmission

Chaque port de communication série possède un buffer ou tampon de transmission de 500 octets.

Les caractères envoyés par une instruction print à partir d'une tâche basic passent dans ce buffer et sont transmis un à un sur le lien série via le format de la liaison.

Si le buffer de transmission est plein (500 caractères stockés et non encore transmis sur le lien série), la tâche basic d'émission est bloquée jusqu'à une libération de place dans le buffer.

L'instruction CLEAROUT vide le buffer.

L'instruction CAROUT retourne le nombre de caractères présents dans le buffer.

L'instruction OUTEMPTY indique que le buffer est vide et que le dernier caractère a été envoyé.

L'instruction PRINT convertit et envoie les données. Sa syntaxe est la suivante :

PRINT #<N°Comm>, <Expression> [{ [; | ,] <Expression> }] [; | ,]

<N°Comm> est le numéro de canal spécifié dans l'instruction OPEN.

Par exemple :

```
OPEN "SERIAL1:" AS #1 ' Ouverture d'un  port de communication
...
PRINT #1, A$, B%;            ' Ecriture d'une chaîne de caractères
                             ' et d'une valeur entière
PRINT #1, C$,
PRINT #1,CHR$(10)  ;MESSAGE1$  ;    ' Pas de caractère ASCII 13D
                                    ' envoyé après MESSAGE1$
PRINT #1,CHR$(10)  ;MESSAGE2$       ' Caractère ASCII 13D
                                    ' envoyé après MESSAGE2$
...
```

Un point-virgule entre deux expressions signifie que le caractère suivant est envoyé immédiatement après le dernier caractère. Un point-virgule en fin de ligne évite l'envoi d'un caractère ASCII 13D supplémentaire.

Une virgule signifie que le caractère suivant est envoyé au début de la prochaine ligne. S'il n'y a pas de liste d'expressions après l'instruction PRINT, celle-ci envoie un caractère ASCII 13D.

Si le #1 ou #2 n'est pas spécifié, par défaut le système envoie les informations sur le #1.

6-5- Fermeture d'un port

Pour fermer un port de communication utiliser l'instruction suivante :

CLOSE #<CommNumber>

6-6- Spécificités du traitement RS 485

Avec un port RS 232, le SUPERVISOR va dialoguer avec un seul périphérique alors qu'avec un RS 485, elle pourra communiquer avec plusieurs.

Pour émettre un message en RS 485, le SUPERVISOR doit « prendre » la ligne de communication .

L'instruction TX485 permet de prendre la ligne pendant un nombre donné de caractères. A chaque caractère envoyé, la valeur contenue dans TX485 est décrémentée. Dès qu'elle atteint 0, la ligne est « rendue » automatiquement afin qu'un autre périphérique puisse la prendre.

' Attention : chaque caractère envoyé est également reçu par le SUPERVISOR tant que TX485 est non nul (fonction écho).

Exemple :

```
.....
Message$=  «  Motion Control System  »
TX485(#1)=Len(Message$)
PRINT #1,Message$  ;         ' prise de la ligne RS485 pendant tout
                             ' l'envoi de Message$
CLEARIN #1                   ' vide les caractères écho
.....
```

6-7- Exemple : Driver Modbus RTU Esclave RS 232

```
Tâche SLAVE232
      Prog
       ' ***
       ' *** DRIVER MODBUS ESCLAVE RS232 ***
       ' ***
       '-------------------------------------------------------------
```

```
       ' *
       ' * INITIALISATION *
       ' *
       '
       ' ATTENTION!!! =>Déclarer en variable globale sauvegardée:
       ' TableModbus   type:entier    nombre:255
       '
       NumeroMcs#=1 'numéro de le SUPERVISOR
       TimeOut&=10  '10ms délai maxi entre 2 caractères recus
       '
       AdresseModBus%=600  'Adresse de début de la table
       NombreModbus%=300   'Nombre de mots dans la table
       ' init compteurs de maintenance
       CmtMessage&=0
       ErrLiaison&=0
       ErrAdresse&=0
       ErrData&=0
       'ouverture serial2
       Open "Serial2:9600,8,N,1" As #2
       Clearin #2 'vide buffer rxd
       TempoRxd&=Time
       '-------------------------------------------------------------
       ' *
       ' * RECEPTION *
       ' *
      InitRxd:
       PtrRxd#=0
       Rxd$=""
       '
      WaitRxd:
       If Carin(#2)<>0 Then Jump ReadRxd
       If PtrRxd#=0 Then Goto WaitRxd
       If Time>TempoRxd& Then Goto InitRxd
       Goto WaitRxd
       '
      ReadRxd:
       TempoRxd&=Time+TimeOut&
       If PtrRxd#>=2 Then Jump MessageRxd
       If PtrRxd#=1 Then Jump Car2Rxd
      Car1Rxd:
       CarRxd$=Input$ #2,1
      Car1tRxd:
       NumMcs#=Asc(CarRxd$)
       If (NumMcs#<>NumeroMcs#) And (NumMcs#<>0) Then Jump InitRxd
       PtrRxd#=1
       Rxd$=CarRxd$
       Jump WaitRxd
      Car2Rxd:
       CarRxd$=Input$ #2,1
       NumFonction#=Asc(CarRxd$)
       If (NumFonction#<>3) And (NumFonction#<>4) And (NumFonction#<>16) Then Jump
Car1tRxd
       PtrRxd#=2
       Rxd$=Rxd$+CarRxd$
       Jump WaitRxd
      MessageRxd:
       CarRxd$=Input$ #2,Carin(#2)
       PtrRxd#=PtrRxd#+len(CarRxd$)
       If PtrRxd#>240 Then Jump InitRxd
       Rxd$=Rxd$+CarRxd$
       If NumFonction#=16 Then
          If PtrRxd#<7 Then Jump WaitRxd
          If PtrRxd#<(Asc(Rxd$,7)+9) Then Jump WaitRxd
          Rxd$=Left$(Rxd$,Asc(Rxd$,7)+9)
         Else
          If PtrRxd#<8 Then Jump WaitRxd
          Rxd$=Left$(Rxd$,8)
       End If
```

```
 '
TraitementMessage:
 Sum$=Left$(Rxd$,Len(Rxd$)-2)
 Sum%=Crc(Sum$)
 Sum$=Mki$(Sum%)
 If Sum$<>Right$(Rxd$,2) Then Jump ErreurLiaison
 AdrBus%=Cvir(Mid$(Rxd$,3,2))
 NbrBus#=Asc(Rxd$,6)
 If (NbrBus#=0) Or (NbrBus#>100) Then Jump ErreurAdresse
 If AdrBus%<AdresseModbus% Then Jump ErreurAdresse
 A1%=AdrBus%+NbBus#
 A2%=AdresseModbus%+NombreModbus%
 If A1%>A2% Then Jump ErreurAdresse
 If NumFonction#=16 Then Jump WriteWord
 '-------------------------------------------------------------
 '
 ' LECTURE DES MOTS
 '
ReadWord:
 If NumMcs#<>NumeroMcs# Then Jump ErreurLiaison
 Txd$=""
 I#=1
 A%=(AdrBus%-AdresseModbus%)+1
ReadWordBcl:
 Txd$=Txd$+Mkir$(TableModbus[A%])
 A%=A%+1
 I#=I#+1
 If I#<=NbrBus# Then Jump ReadWordBcl
 Txd$=Chr$(Len(Txd$))+Txd$
 CmtMessage&=CmtMessage&+1
 Jump MessageTxd
 '-------------------------------------------------------------
 '
 ' ECRITURE DES MOTS
 '
WriteWord:
 I#=1
 J#=0
 A%=(AdrBus%-AdresseModbus%)+1
WriteWordBcl:
 TableModbus[A%]=Cvir(Mid$(Rxd$,8+J#,2))
 A%=A%+1
 I#=I#+1
 J#=J#+2
 If I#<=NbrBus# Then Jump WriteWordBcl
 Txd$=Mid$(Rxd$,3,4)
 CmtMessage&=CmtMessage&+1
 Jump MessageTxd
 '-------------------------------------------------------------
 ' *
 ' * TRANSMISSION *
 ' *
 ' Erreurs
ErreurLiaison:
 ErrLiaison&=ErrLiaison&+1
 Jump InitRxd
ErreurAdresse:
 NumFonction#=NumFonction#+128
 Txd$=Chr$(2)
 ErrAdresse&=ErrAdresse&+1
 Jump MessageTxd
ErreurData:
 NumFonction#=NumFonction#+128
 Txd$=Chr$(3)
 ErrData&=ErrData&+1
 ' Envoi message
MessageTxd:
 Clearin #2 'vide buffer rxd
```

```
 If NumMcs#=0 Then Jump InitRxd
 Txd$=Chr$(NumMcs#)+Chr$(NumFonction#)+Txd$
 Sum%=Crc(Txd$)
 Print #2,Txd$+Mki$(Sum%);
 Jump InitRxd
 '
End Prog
```

7- PROGRAMMATION DE L'ECRAN / CLAVIER

7-1- Présentation

7-1-1- Présentation du SUPERVISOR 640 :

Ecran

- Afficheur LCD monochrome avec rétroéclairage fluorescent
- Fenêtre d'affichage 122×66 mm
- Affichage normal, inversé, clignotant
- Jeux de caractères ASCII
- Résolution 240×128 pixels en mode graphique
- 4 tailles de caractères en mode texte pouvant être utilisées simultanément :
  - ⇨ 3×4 mm 16 lignes de 40 caractères
  - ⇨ 4×7 mm 9 lignes de 30 caractères
  - ⇨ 5×8 mm 8 lignes de 26 caractères
  - ⇨ 7×10 mm 6 lignes de 17 caractères

Clavier

- 33 touches à effet tactile
- 6 touches de fonctions dynamiques
- 6 touches de fonctions relégendables
- Touches de contrôle et de défilement
- Touche d'aide et d'affichage des alarmes
- Pavé numérique et alphanumérique
- 8 leds de visualisation d'état
- Buzzer

7-1-2- Présentation du SUPERVISOR 80 :

Ecran

- Afficheur LCD 4 lignes de 20 caractères avec rétroéclairage par leds
- Fenêtre d'affichage 74×23 mm
- Affichage normal, clignotant
- Jeux de caractères ASCII

Clavier

- 28 touches à effet tactile
- 4 touches de fonctions dynamiques
- 6 touches de fonctions relégendables
- Touches de contrôle et de défilement

- Touche d'aide et d'affichage des alarmes
- Pavé numérique et alphanumérique
- 8 leds de visualisation d'état
- Buzzer

7-2- Fonctions pupitre

7-2-1- Affichage

Quatre fonctions donnent accès à l'écran des terminaux opérateurs.

- La fonction CLS permet d'effacer.

La syntaxe de cette fonction est la suivante : CLS.

Cette instruction possède d'autres extensions :

- ⇨ CLS B : efface l'écran avec un fond noir
- ⇨ CLS W : efface l'écran avec un fond blanc

- Pour afficher ou non le curseur à l'écran, utilisez la fonction CURSOR (on/off). Cette fonction permet d'indiquer à l'utilisateur le début d'une saisie. CURSOR=<ON/OFF>

- Le curseur peut être placé à un endroit précis de l'écran avec la commande LOCATE. L'origine de l'écran est situé en haut et à gauche et a pour coordonnées 1,1. La syntaxe est la suivante : LOCATE <Ligne>,<Colonne>.

- La fonction PRINT permet d'afficher un texte ou le contenu d'une variable sur l'écran. La syntaxe est la suivante:

PRINT <Expression>[;/,]<Expression>[;/,]

Si on utilise une virgule pour séparer deux expressions, un saut de ligne est ajouté. Un point virgule après <Expression> indique au système de ne pas rajouter un saut de ligne (caractère ASCII 13(D))

Exemple :

```
CLS         'effacement d'écran
CURSOR=ON   'affichage du curseur
LOCATE 2,4  'placement du curseur en ligne n°2 et colonne n°4
```

- La fonction FONT permet de définir le type de police à utiliser. La syntaxe est la suivante : FONT=<Valeur>. <Valeur> représente le type de police et varie de 1à 8.

- La fonction PIXEL autorise l'affichage d'un point sur l'écran. La syntaxe est la suivante : PIXEL(X,Y,Couleur). La couleur peut-être le blanc (Couleur=1) ou noir (Couleur=0).

- La fonction BOX réalise l'affichage d'un rectangle. La syntaxe est la suivante :

BOX(X1,Y1,X2,Y2,<Couleur cadre>,<Couleur remplissage>). Les paramètres X1, Y1 représente le coin haut gauche du rectangle et X2, Y2 le coin bas droit. <Couleur cadre> définit la couleur du contour et <Couleur remplissage> la couleur de l'intérieur du rectangle.

- La fonction HLINE réalise l'affichage d'une ligne horizontale. La syntaxe est la suivante : HLINE(X1,Y1,X2,<Couleur>). Les paramètres X1,Y1 représente le point de départ du trait et X2,Y1 le point d'arrivé. <Couleur> définit la couleur du trait.

- La fonction VLINE réalise l'affichage d'une ligne verticale. La syntaxe est la suivante : VLINE(X1,Y1,Y2,<Couleur>). Les paramètres X1,Y1 représente le point de départ du trait et X1,Y2 le point d'arrivé. <Couleur> définit la couleur du trait.

7-2-2- Clavier

On dispose de deux fonctions et d'une variable locale système pour l'utilisation du clavier.

↳ La fonction INKEY permet de lire une touche clavier et de stocker son code dans une variable de type octet. Si aucune touche n'a été appuyée avant l'appel de la fonction celle-ci retourne 0. Cette fonction est non bloquante pour la tâche.

La syntaxe est la suivante : <Variable>=INKEY

Exemple :

```
Attente:
IF Inp(BoutonDepart)=On Then Goto Depart
K#=INKEY
IF K#=0 Then Goto Attente          'Scrutation du clavier
IF K#=@F1 Then Goto MenuF1

Goto Attente
```

↳ La fonction WAIT KEY permet d'attendre l'appui sur une touche et de stocker ensuite le code de cette touche dans la variable locale système KEY. Contrairement à la fonction précédente, cette fonction est bloquante tant qu'aucune touche n'a été appuyée. La syntaxe est la suivante : WAIT KEY

↳ Enfin, la variable système KEY contient le code de la dernière touche appuyée dans les fonctions WAIT KEY ou EDIT. Cette variable est locale à la tâche et ne peut qu'être lue.

Exemple :

```
WAIT  KEY                  'attente d'une touche
IF KEY=@F1 THEN GOTO ...
IF KEY=@F2 THEN GOTO ...
...
```

7-2-3- Editeur

Le SUPERVISOR autorise, via la commande EDIT, de saisir un réel avec ou sans signe et point, en l'affichant à un endroit précis de l'écran. Dans la ligne d'instructions, on choisit le nom de la variable réelle (<Variable>), les numéros de ligne (<Ligne>) et de colonne (<Colonne>) du premier chiffre de la saisie. On peut également préciser si oui ou non (0 ou 1) on utilise le signe (<Signe>) et/ou le point (<Point>).

La syntaxe est la suivante : <Variable> = EDIT(<Ligne>,<Colonne>,<Longueur>,<Signe>,

<Point>).

Pour saisir la valeur sur les pupitres opérateurs, on utilise les touches numériques, les touches DEL pour effacer, ENTER pour valider et ESC pour abandonner la saisie.

Exemple :

```
Saisie!=EDIT(1,5,4,0,0)      'édition d'un réel de quatre chiffres
                             'sans point ni signe en ligne 1 et colonne 5
If Key=@ESC Then Goto MenuPrincipal
If (Saisie!<10) Or (Saisie!>50) Then
          Beep
          Goto MenuPrincipal
End If
Longueur=Saisie!
Goto MenuPrincipal
```

La fonction EDIT possède une deuxième syntaxe. Cette deuxième forme permet d'autoriser des saisies de code d'accès en affichant directement une étoile (*) à chaque touche enfoncée. Ce mode est autorisé par le bit <Code>. La syntaxe est la suivante : <Variable> = EDIT(<Ligne>,<Colonne>,<Longueur>,<Signe>,<Point>,<Code>).

```
SaisieCode!=EDIT(1,5,4,0,0,1)      'édition d'un réel de quatre chiffres
                                   'sans point ni signe en ligne 1 et
```

```
                                    'colonne 5 en mode saisie de
                                    'code d'accès
If Key=@ESC Then Goto MenuPrincipal
If (SaisieCode!=CodeReglage) Then
   Goto MenuReglage
Else
   Beep
   Goto MenuPrincipal
End If
```

Pour saisir une chaîne de caractères à un endroit précis de l'écran, Le SUPERVISOR est muni de l'instruction EDIT$. Dans l'instruction, on spécifie le nom de la variable chaîne de caractères (<Variable>), les numéros de ligne (<Ligne>) et colonne (<Colonne>) du premier caractère de la saisie et le nombre de caractère maximum de la saisie (<Longueur>).

La syntaxe est la suivante : <Variable>=EDIT$(<Ligne>,<Colonne>,<Longueur>). Pour saisir les caractères sur les pupitres opérateurs, on utilise les touches numériques et alphanumériques, les touches DEL pour effacer, ENTER pour valider et ESC pour abandonner la saisie. La saisie d'un caractère alphanumérique s'obtient en appuyant plusieurs fois sur la touche numérique associée.

```
A$=Edit$(2,9,5)       'Saisie en ligne 2, colonne 9 de 5 caractères maxi
```

7-2-4- Buzzer

Deux possibilités sont offertes pour utiliser le buzzer :

↳ Produire et arrêter un son continu - instruction BUZZER

Syntaxe : BUZZER= <ON/OFF>

↳ Emettre un son bref - instruction BEEP - Syntaxe : BEEP

Exemple :

```
IF KEY<>@ENTER THEN BEEP     'émission d'un beep sur appui de «  enter  »
...
Alarme:
BUZZER=ON                    ' émission d'un son continu pendant
DELAY 1000                   ' une durée de 1s
BUZZER=OFF                   ' arrêt du buzzer
DELAY 1000
GOTO Alarme
```

7-2-5- Backlight

Le S640 est muni d'une instruction permettant de gérer la mise en veille du rétro-éclairage : BACKLIGHT. L'inactivité du rétro-éclairage est obtenue lorsqu'aucune touche n'a été appuyée pendant un intervalle de temps prédéfini. Le pupitre redevient actif sur l'appui d'une touche. La syntaxe est la suivante : BACKLIGHT= <durée>. <Durée> définit le temps d'activité d'un pupitre après l'appui de la dernière touche. Cette valeur est une valeur entière qui représente des minutes. La valeurs spécifiques 0 permet d'obtenir un rétro-éclairage tout le temps éteint et 1 un rétro-éclairage toujours allumé. Par défaut, <Durée> est égale à 15mn.

' Attention : le Backlight a une durée de vie de 10000h.

7-2-6- Leds

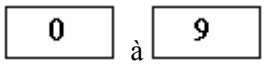
Le pilotage des leds peut s'obtenir à l'aide de l'instruction LED (numéro) =Etat. Le paramètre numéro correspond au nom de la touche où se situe la led (@F1 … @F6) ou pour les leds spécifiques par son nom direct (@ALARM ou @HELP). Le paramètre Etat permet de définir l'état de la led : éteinte (0), allumée (1) ou clignotante (2).

' Attention : Les leds des touches F7…F12 sur le S640 sont pilotées par l'instruction LED(@F1)…LED(@F6).

7-3- Correspondance des touches

7-3-1- Correspondance des touches

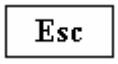
à F12 @F1 à @F12



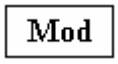
à @0 à @9

@HELP

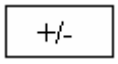
@ALARM



@ESC



@MOD



@SIGN

@POINT

@UP

@DOWN

@RIGHT

@LEFT

@RETURN

7-4- Menus internes

7-4-1- Généralités

Ces menus permettent de :

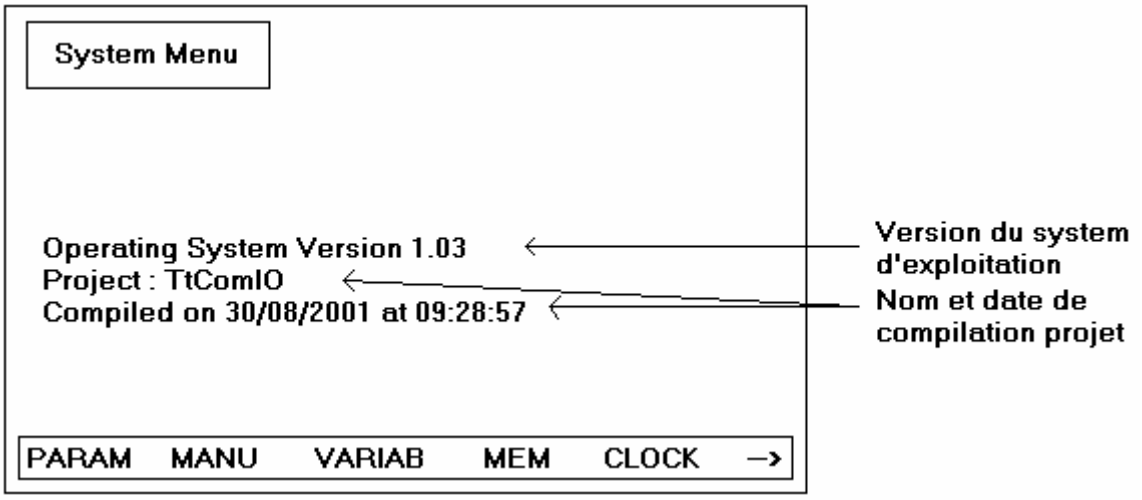
- Tester les entrées / sorties en manuel
- Lire et écrire les variables globales sauvegardées
- Gérer la sauvegarde et la restauration des données en flash
- Régler l'heure et la date
- Changer l'état du chien de garde

Ces menus internes sont exécutés grâce à l'instruction CALL du langage. Les menus sont utilisés comme des sous-programmes. Le nom des menus commence par le caractère '_'. La syntaxe est : CALL <Nom du menu>.

7-4-2- Menu général

Menu général : MENUMCS

Syntaxe : CALL _MENUMCS



Fonction : Donne accès à tous les sous-menus.

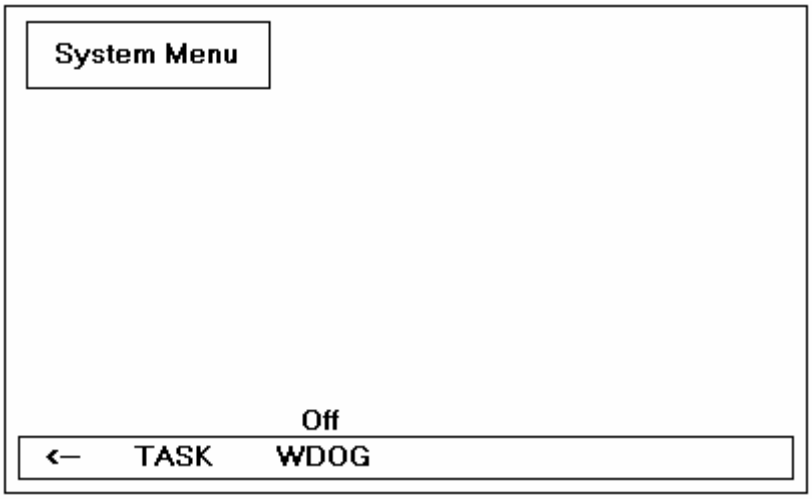
Touches :

F1 : sous-menu paramètres F2 : sous-menu manuel

F3 : sous-menu variables F4 : sous-menu mémoire

F5 : sous-menu horloge F6 : page suivante

ESC : sortie du menu



Fonction : Donne accès à tous les sous-menus.

Touches :

F1 : page precédente F2 : sous-menu tâche

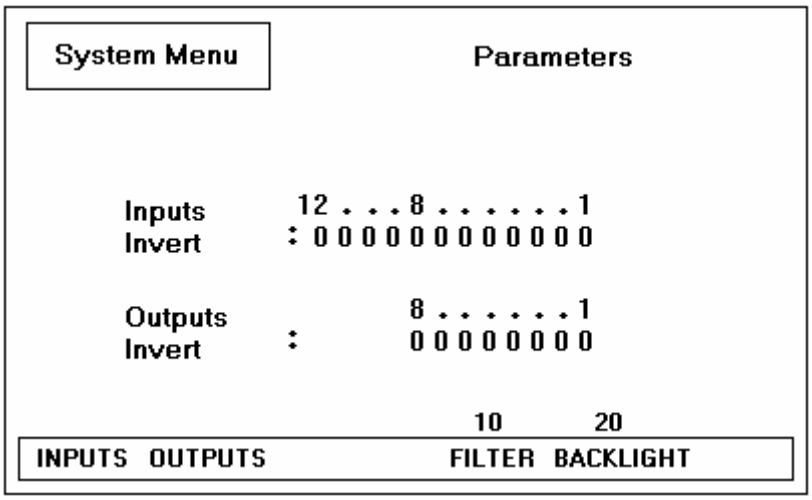
F3 : modifie l'état du chien de garde

ESC : sortie du menu

7-4-3- Sous-menu paramètre

Menu général : PARAMMCS

Syntaxe : CALL _PARAMMCS



Touches :

F1 : réglage des entrées F2 : réglage des sorties

⇒ Utiliser les flèches directionnels pour sélectionner une entrée ou une sortie

⇒ Utiliser les touches 0 ou 1 pour inversé ou non votre selection

F4 : réglage du filtre F4 : réglage du backlight

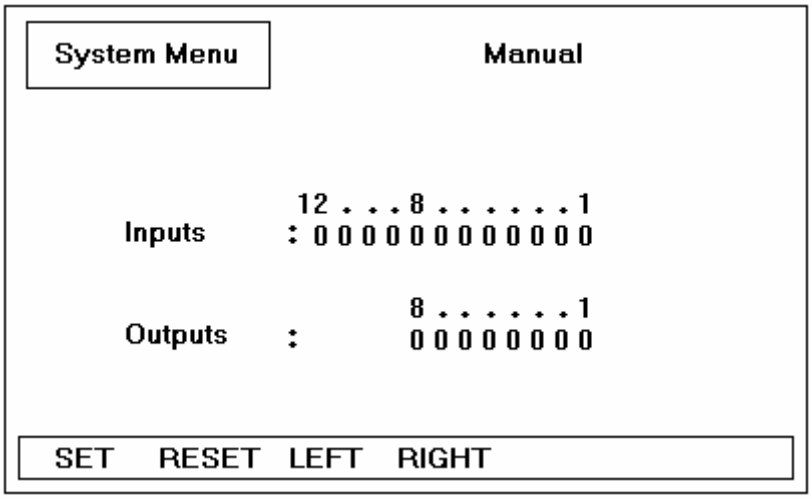
⇒ Entrer une valeur numérique comprise entre 000 et 999 puis valider

ESC : sortie du menu

7-4-4- Sous-menu manuel

Menu général : MANUMCS

Syntaxe : CALL _MANUMCS



Touches :

F1 : SET F2 : RESET

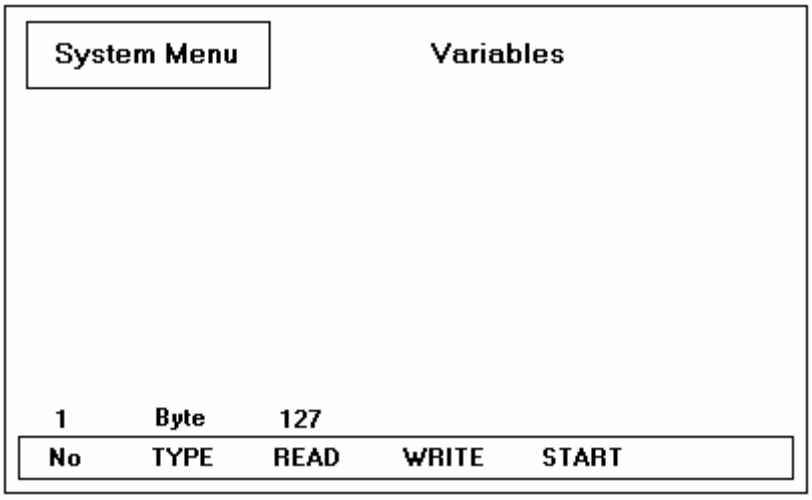
F3 : LEFT F4 : RIGHT

ESC : sortie du menu ou retour au menu général

7-4-5- Sous-menu variables

Menu général : VARIABMCS

Syntaxe : CALL _VARIABMCS



Touches :

F1 : Numéro de la variable F2 : Type de la variable

F3 : lecture F4 : écriture

F5 : lecture en temps réel F6 : Arrêt de la lecture

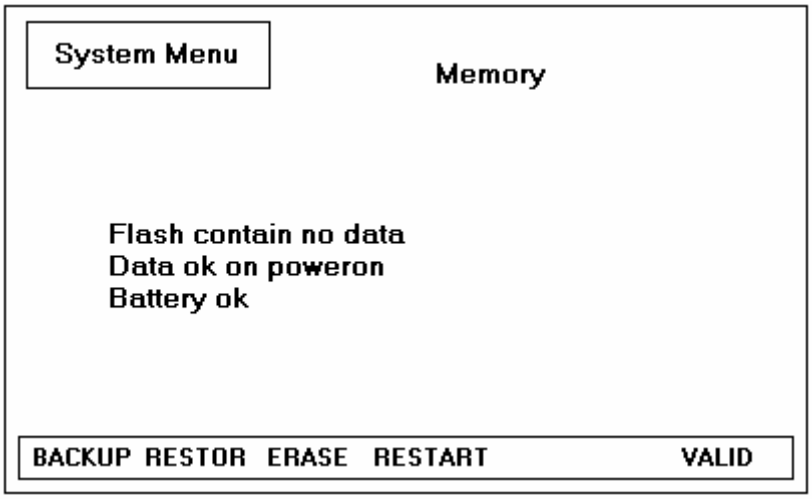
↑ : variable suivante ↓ : variable précédente

ESC : sortie du menu ou retour au menu général

7-4-6- Sous-menu mémoire

Menu général : MEMMCS

Syntaxe : CALL _MEMMCS



Touches :

F1 : sauvegarde des données ram->flash F2 : restauration des données ram->flash

F3 : efface les données en flash F4 : redémarre le supervisor

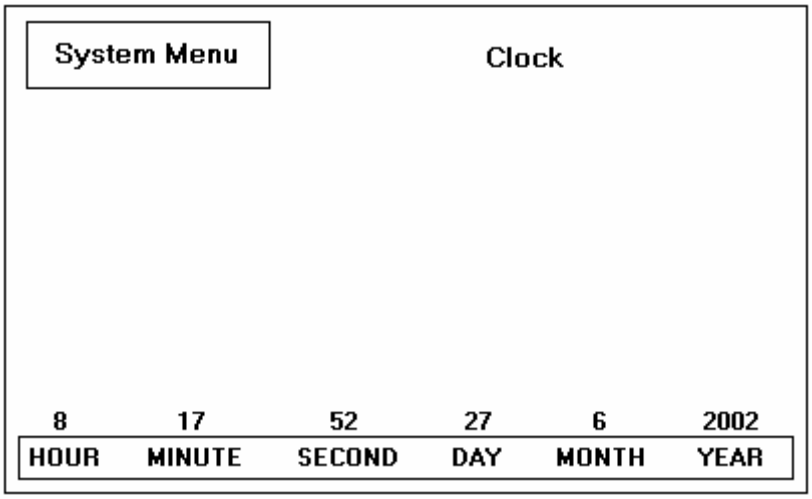
F6 : valide les données en memoire (même si elles sont erronées)

ESC : sortie du menu ou retour au menu général

7-4-7- Sous-menu horloge

Menu général : CLOCKMCS

Syntaxe : CALL _CLOCKMCS



Touches :

F1 : réglage de l'heure F2 : réglage des minutes F3 : réglage des secondes

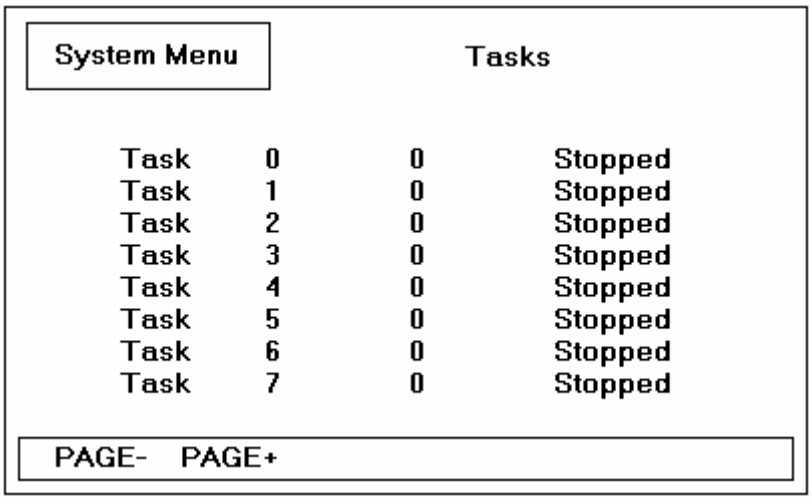
F4 : réglage du jour F5 : réglage du mois F6 : réglage de l'année

ESC : sortie du menu ou retour au menu général

7-4-8- Sous-menu tâches

Menu général : TASKMCS

Syntaxe : CALL _TASKMCS



Touches :

F1 : Page précédente F2 : Page suivante

ESC : sortie du menu ou retour au menu général

8- LISTE DES OPERATEURS ET INSTRUCTIONS

8-1- Programme

CALL	Appel de Sous-programme
ICALL	Appel de Sous-programme
END	Fin de bloc
EXIT SUB	Sortie d'un sous-programme
GOTO	Saut à une étiquette
JUMP	Saut à une étiquette
PROG ... END PROG	Début d'un programme
SUB ... END SUB	Sous-programme

8-2- Arithmétique

+	Addition
-	Soustraction
*	Multiplication
/	Division

8-3- Mathématique

ABS	Valeur absolue
ARCCOS	Cosinus inverse
ARCSIN	Sinus inverse
ARCTAN	Tangente inverse
COS	Cosinus
DIV	Division entière
EXP	Exponentiel
FRAC	Partie fractionnelle
INT	Partie entière
LOG	Logarithme
MOD	Modulo
SGN	Signe
SIN	Sinus
SQR	Racine carrée
TAN	Tangente
^	Puissance

8-4- Logique

<<	Décalage à gauche
>>	Décalage à droite
AND	Opérateur ET
NOT	Opérateur complément

OR	Opérateur OU
XOR	Opérateur OU exclusif

8-5- Test

<	Inférieur
<=	Inférieur ou égal
<>	Différent
=	Egal / affectation
>	Supérieur
>=	Supérieur ou égal
CASE ...	Test multiples
IF ... THEN ... ELSE ... END IF	Structure de test

8-6- Boucles

FOR ... TO ... STEP ... NEXT
REPEAT ... UNTIL
WHILE ... DO ... END WHILE

8-7- Communication

CARIN	Etat du buffer d'entrée
CAROUT	Etat du buffer de sortie
CLEARIN	Vide le buffer d'entrée
CLEAROUT	Vide le buffer de sortie
CLOSE	Fermer le port de communication
INPUT	Lecture de données
INPUT$	Lecture de chaînes de caractères
OPEN ... AS ...	Ouvre un port de communication
OUTEMPTY	Etat du buffer de sortie
PRINT	Ecrit sur le port de communication
TX485	Modifie l'état de la sortie RS485

8-8- Chaîne de caractères

ASC	Code ASCII d'un caractère
CHR$	Caractère à partir de son code ASCII
FORMAT$	Créer une chaîne formatée
INSTR	Cherche une sous-chaîne
LCASE$	Minuscule
LEFT$	Partie gauche d'une chaîne
LEN	Longueur d'une chaîne
LTRIM$	Enlève les espaces à gauche
MID$	Partie d'une chaîne
RIGHT$	Partie droite d'une chaîne

RTRIM$	Supprime les espaces à droite
SPACE$	Chaîne d'espace
STR$	Conversion en chaîne de caractères
STRING$	Création de chaîne
UCASE$	Majuscule
VAL	Convertir une chaîne en numérique

8-9- Automate

8-9-1- Entrées / sorties TOR

INP	Lecture d'une entrée TOR
INPB	Lecture d'un bloc de 8 entrées
INPW	Lecture d'un bloc de 16 entrées
OUT	Ecriture d'une sortie
OUTB	Ecriture d'un bloc de 8 sorties
OUTW	Ecriture d'un bloc de 16 sorties
PLCINIT	Initialisation des fonctions PLC
PLCINP	Lecture d’une entrée TOR
PLCINPB	Lecture d’un bloc 8 entrées
PLCINPPE	Lecture d’un front montant d’une entrée TOR PLC
PLCINPNE	Lecture d’un front montant d’une entrée TOR PLC
PLCINPW	Lecture d’un bloc 16 entrées
PLCOUT	Ecriture d’une sortie
PLCOUTB	Ecriture d’un bloc 8 sorties
PLCOUTW	Ecriture d’un bloc 16 sorties
PLCREADINPUTS	Lecture des entrées PLC
PLCWRITEOUTPUTS	Ecriture des sorties PLC
SETINP	Filtrage et inversion des entrées
SETOUT	Inversion des sorties
WAIT	Attente d'une condition

8-9-2- Temporisations

DATE$	Date courante
DELAY	Attente passive
GETDATE	Date courante
GETTIME	Heure courante
SETDATE	Fixe la date
SETTIME	Fixe l'heure
TIME	Base de temps
TIMER	Base de temps étendue
TIME$	Heure courante

8-9-3- Manipulation d'événements

DIFFUSE	Génération d'événement
GETEVENT	Lecture d'événements
MODIFYEVENT	Configuration des événements
SIGNAL	Génération d'événement
WAIT EVENT	Attente d'un événement

8-9-4- Compteurs

CLEARCOUNTER	Remise à zéro du compteur
COUNTER_S	Lecture du compteur
SETUPCOUNTER	Configuration du compteur

8-10- Gestion des tâches

CONTINUE	Continue l'exécution d'une tâche
HALT	Arrête une tâche
RUN	Lance une tâche
SUSPEND	Suspend une tâche
STATUS	Etat d'une tâche

8-11- Ecran / Clavier

8-12- Supervisor 80 et 640

BEEP	Emet un son bref
BUZZER	Emet un son continu
CLS	Efface l'écran
CURSOR	Efface ou affiche le curseur
EDIT	Saisie d'une valeur
EDIT$	Saisie d'une valeur alphanumérique
INKEY	Lit une touche
KEY	Dernière touche
KEYDELAY	Délai avant répétition touche
KEYREPEAT	Période de répétition touche
LED	Pilotage des leds
LOCATE	Positionne le curseur
PRINT	Affiche un texte
READKEY	Touche appuyée
WAIT KEY	Attente d'une touche

8-13- Supervisor 640

BACKLIGHT	Mise en veille
BOX	Dessin d'un rectangle
FONT	Sélection de la police
HLINE	Dessin d'une ligne horizontale
PIXEL	Affichage d'un point
VLINE	Dessin d'une ligne verticale

8-14- Conversion

CVL	Conversion chaîne - Entier long
CVLR	Conversion chaîne - Entier long inverse
CVI	Conversion chaîne - Entier
CVIR	Conversion chaîne - Entier inverse
LONGTOINTEGER	Conversion Entier long - Entier
MKL$	Conversion Entier long - chaîne
MKLR$	Conversion Entier long inverse - chaîne
MKI$	Conversion Entier - chaîne
MKIR$	Conversion Entier inverse - chaîne
REALTOLONG	Conversion réel - Entier long
REALTOINTEGER	Conversion réel - Entier
REALTOBYTE	Conversion réel - Octet

8-15- Flash, Sécurité, Divers

CRC	Calcule le CRC 16 au format modbus RTU
CLEARFLASH	Efface la mémoire Flash
FLASHOK	Test les données en Flash
FLASHTORAM	Transfert de la Flash vers la ram
POWERFAIL	Gestion des microcoupures
RAMTOFLASH	Transfert de la ram vers la Flash
RAMOK	Etat de la ram au démarrage
RESTART	Redémarrage du système
VERSION	Version de l'operating system
WATCHDOG	Chien de garde

8-16- Liste alphabétique

8-16-1- Addition (+)

Syntaxe : <Expression1> + <Expression2>

Types acceptés : Octet, Entier, Entier long, réel ou chaîne de caractère

Description : Cet opérateur additionne deux expressions numériques ou réalise la concaténation de deux chaînes de caractères et retourne une valeur du même type que ces opérandes.

Remarques : <Expression1> et <Expression2> doivent être des expressions valides. <Expression1> et <Expression2> doivent être de même type.

Exemple :
```
a%=10
b%=5
c%=a%+b%    'Résultat : c%=15
```

Voir aussi : `-`, `*' et `/'.

8-16-2- Soustraction (-)

Syntaxe : <Expression1> **-** <Expression2>

Types acceptés : Octet, Entier, Entier long ou réel

Description : Cet opérateur soustrait l'<Expression2> de l'<Expression1> et retourne une valeur du même type que ces opérandes.

Remarques : <Expression1> et <Expression2> doivent être des expressions numériques valides. <Expression1> et <Expression2> doivent être de même type.

Exemple :
```
a%=10
b%=5
c%=a%-b%    'Résultat : c%=5
```

Voir aussi : '+', `*' et `/'.

8-16-3- Multiplication (*)

Syntaxe : <Expression1> ***** <Expression2>

Types acceptés : Octet, Entier, Entier long ou réel

Description : Cet opérateur multiplie l'<Expression1> par l'<Expression2> et retourne une valeur du même type que ces opérandes.

Remarques : <Expression1> et <Expression2> doivent être des expressions numériques valides. <Expression1> et <Expression2> doivent être de même type.

Exemple :
```
a%=10
b%=5
c%=a%*b%    'Résultat : c%=50
```

Voir aussi : '+', '-' et `/'.

8-16-4- Division (/)

Syntaxe : <Expression1> **/** <Expression2>

Types acceptés : Octet, Entier, Entier long ou réel

Description : Cet opérateur divise l'<Expression1> par l'<Expression2>

Remarques : <Expression1> et <Expression2> doivent être des expressions numériques valides. <Expression1> et <Expression2> doivent être de même type. <Expression2> doit être différente de zéro. Cet opérateur retourne toujours une valeur réelle.

Exemple :
```
a%=10
b%=5
c!=a%/b%    'Résultat : c!=2.0
```

Voir aussi : '+', '-', `*' et DIV.

8-16-5- Inférieur (<)

Syntaxe : <Expression1> **<** <Expression2>

Types acceptés : Octet, Entier, Entier long, réel ou chaîne de caractères

Description : Cet opérateur teste si <Expression1> est inférieure à <Expression2>.

Remarques : <Expression1> et <Expression2> doivent être des expressions valides. <Expression1> et <Expression2> doivent être de même type.

Exemple :
```
a%=10
IF b%<a% THEN ...
```

Voir aussi : '=', '>', '>=', '<=', '<>'.

8-16-6- Inférieur ou égal (<=)

Syntaxe : <Expression1> **<=** <Expression2>

Types acceptés : Octet, Entier, Entier long, réel ou chaîne de caractères

Description : Cet opérateur teste si <Expression1>est inférieure ou égale à <Expression2>.

Remarques : <Expression1> et <Expression2> doivent être des expressions valides. <Expression1> et <Expression2> doivent être de même type.

Exemple :
```
a%=10
IF b%<=a% THEN ...
```

Voir aussi : '=', '>', '>=', '<', '<>'.

8-16-7- Décalage à gauche (<<)

Syntaxe : <Expression1> **<<** <Expression2>

Types acceptés : Octet ou Entier

Description : Cet opérateur déplace <Expression2> bits de <Expression1> de droite à gauche.

Remarques : <Expression2> représente le nombre de bits à déplacer. Le décalage n'est pas circulaire.

Exemple :
```
a%=100b
b% =a%<<2   'Résultat b%=10000b
```

Voir aussi : '>>'.

8-16-8- Différent (<>)

Syntaxe : <Expression1> **<>** <Expression2>

Types acceptés : Octet, Entier, Entier long, réel ou chaîne de caractères

Description : Cet opérateur teste si <Expression1> et <Expression2> sont différentes.

Remarques : <Expression1>et <Expression2> doivent être des expressions valides. <Expression1> et <Expression2> doivent être de même type.

Exemple :
```
a%=10
IF b%<>a% THEN ...
```

Voir aussi : '=', '>', '>=', '<', '<='

8-16-9- Affectation/Egalité (=)

Syntaxe : <Expression1> = <Expression2> Ou <Variable>=<Expression2>

Types acceptés : Bit, Octet, Entier, Entier long, réel ou chaîne de caractères

Description : Cet opérateur affecte <Variable> à <Expression2> ou teste si <Expression1> est égale à <Expression2>.

Remarques : <Expression1> et <Expression2> doivent être des expressions valides. <Expression1>, <Expression2> et <Variable> doivent être de même type.

Exemple :
```
a%=10
```

```
IF b%=5 THEN ...
```

Voir aussi : '>', '>=', '<', '<=', '<>'

8-16-10- Supérieur (>)

Syntaxe : <Expression1> **>** <Expression2>

Types acceptés : Octet, Entier, Entier long, réel ou chaîne de caractères

Description : Cet opérateur teste si <Expression1> est supérieure à <Expression2>.

Remarques : <Expression1> et <Expression2> doivent être de même type.

Exemple :
```
IF b%>a% THEN ...
```

Voir aussi : '=', '>=', '<', '<=', '<>'

8-16-11- Supérieur ou égal (>=)Diff_rent

Syntaxe : <Expression1> **>=** <Expression2>

Types acceptés : Octet, Entier, Entier long, réel ou chaîne de caractères

Description : Cet opérateur teste si <Expression1> est supérieure ou égale à <Expression2>.

Remarques : <Expression1> et <Expression2> doivent être de même type.

Exemple :
```
IF b%>=a% THEN ...
```

Voir aussi : '=', '>', '<', '<=', '<>'.

8-16-12- Décalage à droite (>>)

Syntaxe : <Expression1> **>>** <Expression2>

Types acceptés : Octet ou Entier

Description : Cet opérateur déplace <Expression2> bits de <Expression1> de gauche à droite.

Remarques : <Expression2> représente le nombre de bits à déplacer. Le décalage n'est pas circulaire.

Exemple :
```
a%=11010b
b% =a%>>2  'Résultat b%=110b
```

Voir aussi : '<<'.

8-16-13- Puissance (^)

Syntaxe : <Expression1> ^ <Expression2>

Types acceptés : Octet, Entier, Entier long ou réel

Description : Cet opérateur élève <Expression1> à la puissance <Expression2>.

Exemple :
```
a!=b!^2 ' a=b²
```

8-16-14- ABS - Valeur absolue

Syntaxe : **ABS** (<Expression>)

Types acceptés : Octet ou Entier

Description : Cette fonction fournit la valeur absolue de <Expression>. Un nombre négatif est donc converti en un nombre positif.

Remarques : <Expression> doit être une expression numérique valide. La valeur absolue d'un nombre est sa valeur non-signée.

Exemple :
```
a%=ABS(-100)      'Résultat : a%=100
a%=ABS(25) 'Résultat : a%=25
```

8-16-15- AND – Opérateur ET

Syntaxe : <Expression1> **AND** <Expression2>

Types acceptés : Bit, Octet ou entier

Description : Cette fonction effectue un ET binaire entre deux expressions et retourne une valeur du type de l'opérande.

Remarques : <Expression1> et <Expression2> doivent être du même type.

Exemple :
```
IF (A% AND 0FF00h)<>0 THEN ...
```

Voir aussi : OR, NOT, XOR et IF

8-16-16- ARCCOS – Cosinus inverse

Syntaxe : **ARCCOS** (<Expression>)

Limite : de –1 à +1

Types acceptés : Octet, Entier, Entier long, réel

Description : Cette fonction restitue l' arccosinus de <Expression>.

Remarques : <Expression> doit être une expression numérique valide. Cette fonction restitue un angle exprimé en radian.

Exemple :
```
pi!=2*ARCCOS(0)
```

Voir aussi : SIN, COS et TAN

8-16-17- ARCSIN – Sinus inverse

Syntaxe : **ARCSIN** (<Expression>)

Limite : de –1 à +1

Types acceptés : Octet, Entier, Entier long, réel

Description : Cette fonction restitue l' arcsinus de <Expression>.

Remarques : <Expression> doit être une expression numérique valide. Cette fonction restitue un angle exprimé en radian.

Exemple :
```
pi!=2*ARCSIN(1)
```

Voir aussi : SIN, COS et TAN

8-16-18- ARCTAN – Tangente inverse

Syntaxe : **ARCTAN** (<Expression>)

Types acceptés : Octet, Entier, Entier long, réel

Description : Cette fonction restitue l'arctangente de <Expression>.

Remarques : <Expression> doit être une expression numérique valide. La fonction ARCTAN prend le rapport de deux côtés d'un triangle rectangle et restitue l'angle correspondant. Le rapport est la longueur du côté opposé à l'angle par la longueur du côté adjacent à l'angle.

Exemple :
```
a!=ARCTAN(3)
pi!=4*ARCTAN(1)
```

Voir aussi : SIN, COS et TAN

8-16-19- ASC – Code ASCII d'un caractère

Syntaxe : **ASC**(<Chaîne>)

Types acceptés : Chaîne de caractères

Description : Cette fonction restitue une valeur numérique qui correspond au code ASCII du premier caractère d'une chaîne.

Remarques : Si la longueur de l'argument <chaîne> est nulle la valeur zéro est retournée.

Exemple :
```
a$="A"
b#=ASC(a$) 'Résultat : b#=65
```

Voir aussi : CHR$.

8-16-20- BACKLIGHT – Mise en veille du Supervisor 640

Syntaxe : **BACKLIGHT**=<durée>

Unité : durée : minutes

Types acceptés : durée : Entier

Description : Cette fonction permet de définir la durée en minute pendant laquelle le rétro-éclairage du S640 restera allumé si l'utilisateur n'appuie pas sur les touches du pupitre. Par défaut la durée est de 15 minutes.

Remarques : Lorsque le S640 est en veille, si l'utilisateur appuie sur une touche du S640 le rétro-éclairage du pupitre redevient actif. Durée doit être une variable de type entier.

Durée : 0→rétro-éclairage éteint

1→rétro-éclairage toujours allumé.

(Durée>1) →durée en minute.

Attention : la durée de vie du rétro-éclairage est de 10 000 heures.

Exemple :
```
BACKLIGHT=120     'Le S640 se mettra en veille au bout de
                  '2 heures si l'utilisateur n'appuie pas sur
                  ' les touches du pupitre.
```

8-16-21- BEEP – Emet un son bref

Syntaxe : **BEEP**

Description : Cette instruction émet un son bref.

Exemple :
```
IF KEY<>@ENTER THEN BEEP
```

Voir aussi : BUZZER

8-16-22- BOX – Affichage rectangle

Syntaxe : **BOX**(X1,Y1,X2,Y2,Couleur cadre, Couleur remplissage)

Unité : X1, Y1, X2, Y2 : pixel

Limites : X1, X2 : de 1 à 240

Y1, Y2 : de 1 à 128

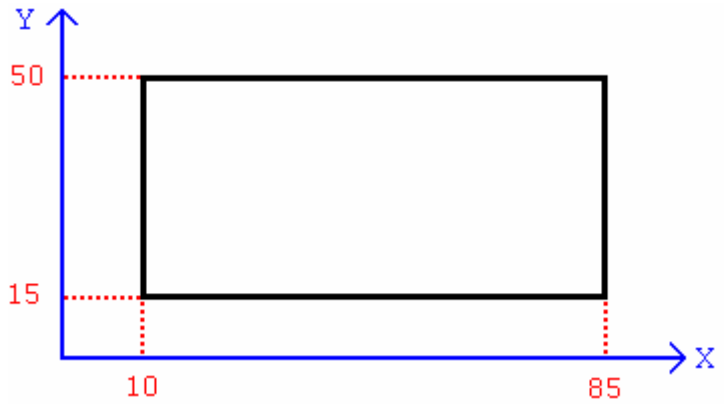
Types acceptés : X1, Y1, X2, Y2, Couleur remplissage : Octet

Couleur cadre : Bit

Description : Trace un rectangle de coordonées X1,Y1 (coin haut, gauche) et X2,Y2 (coin bas, droit).

Remarques : Le paramètre Couleur cadre permet de modifier la couleur du contour en noir (0) ou blanc (1). Couleur remplissage permet de modifier la couleur intérieur du cadre en noir (0), en blanc (1) ou transparent (2).

Exemple :
```
BOX(10,50,85,15,0,1)     'cadre noir avec une couleur de
                         'remplissage blanche
```



8-16-23- BUZZER – Emet un son continu

Syntaxe : **BUZZER** = <ON|OFF>

Description : Cette fonction active ou désactive le buzzer.

Exemple :

```
Alarme:
    BUZZER=ON
    DELAY 1000
    BUZZER=OFF
    DELAY 1000
    GOTO Alarme
```

Voir aussi : BEEP

8-16-24- CALL – Appel d'un sous-programme

Syntaxe : **CALL** <Nom>

Description : Cette instruction est utilisée pour appeler un sous-programme défini par un bloc SUB. <Nom> est le nom du bloc du sous-programme.

Remarques : Un sous-programme ne peut pas s'appeler. Le système contient des sous-programmes pré-définis : _MENUMCS, _PARAMMCS, _MANUMCS, _VARIABMCS, _MEMORYMCS et _CLOCKMCS. L'exécution de cette instruction provoque un basculement de tâche.

Exemple :

```
CALL Move
```

Voir aussi : SUB, ICALL

8-16-25- CASE – Test multiples

Syntaxe 1 : **CASE** <Expression> CALL <Label 1> [{ , <Label2> }]

Syntaxe 2 : **CASE** <Expression> GOTO <Label 1> [{ , <Label2> }]

Types acceptés : Expression : Entier

Description : Cette fonction permet de faire des sauts à des étiquettes ou des appels de sous-programmes en fonction des valeurs de <Expression>.

Remarques : <Expression> doit être une valeur valide entière. Si la valeur de Expression est égale à 0 ou supérieure au nombre d'étiquettes, la tâche continue à la ligne suivante. Cette instruction provoque le basculement à une tâche suivante.

Exemple :

```
Case  a% GOTO Move1, Move2
Goto Fin    'a%=0 ou a%>2
...
```

```
Move1:      'a% = 1
...
Move2:      'a% = 2
...
Fin:
```

8-16-26- CARIN – Etat du buffer d'entrée de communication

Syntaxe : **CARIN** (<Numéro>)

Description : Cette fonction restitue le nombre de caractères présents dans le buffer d'entrée du port de communication.

Remarques : <Numéro> est le numéro utilisé pour ouvrir le port de communication avec l'instruction OPEN. Cette fonction retourne un entier.

Exemple :
```
WAIT CARIN(#1)>=3 ' Attend au moins 3 caractères reçus
A$=Input$ #1,3    ' lit 3 caractères
```

Voir aussi : CAROUT, CLEARIN

8-16-27- CAROUT – Etat du buffer de sortie de communication

Syntaxe : <Expression>=**CAROUT** (<Numéro>)

Types acceptés : <Expression> : entier

Description : Cette fonction retourne le nombre de caractères présents dans le buffer de sortie du port de communication.

Remarques : <Numéro> est le numéro utilisé pour ouvrir le port de communication avec l'instruction OPEN.

Exemple :
```
WAIT CAROUT(#1)<10      'Attend de la place dans le buffer
Print A$;               'écriture des caratères
```

Voir aussi : CARIN, CLEAROUT

8-16-28- CHR$ - Caractère à partir de son code ASCII

Syntaxe : **CHR$**(<Code>)

Types acceptés : Code : Octet

Description : Cette fonction restitue une chaîne d'un caractère dont le code ASCII est l'argument.

Exemple :
```
a#=97
b$=CHR$(a#)       'Résultat : b$="a"
```

Voir aussi : ASASC C

8-16-29- CLEARCOUNTER – remise à zéro du compteur

Syntaxe : **CLEARCOUNTER**(<Compteur>)

Types acceptés : <Compteur> : 1 ou 2

Description : Cette instruction permet d'initialiser à 0 le compteur

Remarques : <Compteur> : Numéro du compteur (1 ou 2)

Voir aussi : COUNTER_S, SETUPCOUNTER

8-16-30- CLEARFLASH – Efface la mémoire flash

Syntaxe : **CLEARFLASH**

Description : Cette fonction efface les paramètres et les 10000 premières variables sauvegardées de la mémoire flash.

Voir aussi : RAMOK, FLASHOK, FLASHTORAM

8-16-31- CLEARIN – Vide le buffer d'entrée de communication

Syntaxe : **CLEARIN** <Numéro>

Description : Cette instruction vide le buffer d'entrée du port de communication.

Remarques : <Numéro> est le numéro utilisé pour ouvrir le port de communication avec l'instruction OPEN.

Exemple :
```
CLEARIN #1
Wait CARIN (#1)>=3       'Attend au moins 3 caractères
A$=Input$ #1,3           'Lit 3 caractères
```

Voir aussi : CARIN

8-16-32- CLEAROUT – Vide le buffer de sortie de communication

Syntaxe : **CLEAROUT** <Numéro>

Description : Cette instruction vide le buffer de sortie du port de communication.

Remarques : **<**Numéro> est le numéro utilisé pour ouvrir le port de communication avec l'instruction OPEN.

Exemple :
```
CLEAROUT #1
Print A$;          'Ecrit les caractères
```

Voir aussi : CAROUT

8-16-33- CLOSE – Ferme le port de communication

Syntaxe : **CLOSE** #Numéro

Description : L'argument Numéro est le numéro utilisé dans l'instruction OPEN pour ouvrir le port de communication**.**

Remarques : Si vous voulez changer le mode de communication (format de la liaison…), vous devez fermer et réouvrir le port de communication.

Exemple :
```
CLOSE #1
```

Voir aussi : OPEN, INPUT et PRINT.

8-16-34- CLS – Efface l'écran du terminal

Syntaxe : **CLS**

CLS 1, CLS 2, CLS 3 ou CLS 4 (uniquement avec le Supervisor 80)

CLS B, CLS W (uniquement avec le Supervisor 640)

Description : CLS efface l'écran du pupitre opérateur. Cls 1, Cls 2, Cls 3, Cls 4 efface respectivement la première, seconde, troisième ou quatrième ligne des Supervisor 80. La fonction CLS B efface l'écran du Supervisor 640 avec un fond noir. La fonction CLS W efface l'écran du Supervisor 640 avec un fond blanc.

8-16-35- CONTINUE – Continue l'exécution d'une tâche

Syntaxe : **CONTINUE** <Nom>

Description : Cette instruction est utilisée pour continuer l'exécution d'une tâche supendue.

Remarques : <Nom> doit être le nom d'une tâche suspendue. Cette fonction n'a pas d'effet sur une tâche stoppée ou en cours d'exécution.

Exemple :
```
Wait Inp(Start)
RUN Coupe
Begin:
  Wait Inp(Stop)
  SUSPEND Coupe
  Wait Inp(Start)
  CONTINUE Coupe
Goto Begin
```
Voir aussi : RUN, HALT, SUSPEND

8-16-36- COS - Cosinus

Syntaxe : **COS**(<Expression>)

Types acceptés : Expression : réel

Description : Cette instruction retourne le cosinus de <Expression>.

Remarques : L'argument <Expression> doit être une expression numérique valide. La fonction COS prend un angle et restitue le rapport de deux côtés d'un triangle rectangle. Le rapport est la longueur du côté adjacent à l'angle divisé par la longueur de l'hypoténuse. <Expression>est exprimée en radians.

Exemple : a!=COS(3.14159)

Voir aussi : SIN, ARCTAN et TAN

8-16-37- COUNTER_S – lecture du compteur

Syntaxe : <Variable>=**COUNTER_S(**<Compteur>)

Types acceptés : <Variable> : Entier

<Compteur> : 1 ou 2

Description : Cette instruction permet de lire le compteur

Remarques : <Compteur> : Numéro du compteur (1 ou 2)

Voir aussi : SETUPCOUNTER, CLEARCOUNTER

8-16-38- CRC – CRC16

Syntaxe : Valeur CRC%=**CRC**(<Expression >)

Types acceptés : Expression : chaîne de caractères

Description : Retourne la valeur du CRC au format Modbus RTU (CRC 16) d'une chaîne de caractères.

Exemple :
```
A%=CRC(message$)
```

8-16-39- CURSOR – Affiche ou efface le curseur

Syntaxe : **CURSOR** = <ON | OFF>

Description : Cette fonction affiche ou non le curseur sur le pupitre opérateur.

Remarques : Cette fonction utilise le port de communication #1. Par défaut, le port de communication SERIAL1 sera utilisé. Si un pupitre opérateur est connecté au port SERIAL2, veuillez vous référer à la fonction OPEN pour affecter #1 au port SERIAL2.

8-16-40- CVL – Conversion Chaîne / Long

Syntaxe : <Variable>=**CVL**(<Expression chaîne de 4 octets>)

Types acceptés : Variable : Entier Long

Expression : chaîne de 4 octets

Description : La fonction CVL sert à convertir une chaîne de 4 octets créée par MKL$ en une valeur de type Long. Poids faible puis poids fort

Exemple :
```
A&=CVL(A$) 'Si  A$=chr$(2)+chr$(3)+chr$(1)+chr$(0)
           'alors  A&=2+(3*256)+(1*65536)+(0*16777216)=66306
```

Voir aussi : CVLR, MKL$, MKLR$

8-16-41- CVLR – Conversion Chaîne / Long reverse

Syntaxe : <Variable>=**CVLR**(<Expression>)

Types acceptés : Variable : Entier Long

Expression : chaîne de 4 octets

Description : La fonction CVLR sert à convertir une chaîne de 4 octets créée par MKLR$ en une valeur de type Long. Poids fort puis poids faible

Exemple :
```
A&=CVLR(A$)'Si  A$=chr$(0)+chr$(1)+chr$(3)+chr$(2) alors
           'A&=(0*16777216)+(1*65536)+(3*256)+(2*1)=66306
```

Voir aussi : CVL, MKL$, MKLR$

8-16-42- CVI – Conversion Chaîne / Integer

Syntaxe : <Variable>=**CVI**(<Expression>)

Types acceptés : Variable : Entier

Expression : chaîne de 2 octets

Description : La fonction CVIR sert à convertir une chaîne de 2 octets créée par MKI$ en une valeur de type Integer. Poids faible puis poids fort

Exemple :
```
A&=CVI(A$) 'Si  A$=chr$(0)+chr$(1) alors A&=0+(1*256)=256
```

Voir aussi : CVIR, MKI$, MKIR$

8-16-43- CVIR – Conversion Chaîne / Integer reverse

Syntaxe : <Variable>=**CVIR**(<Expression>)

Types acceptés : Variable : Entier

Expression : chaîne de 2 octets

Description : La fonction CVIR sert à convertir une chaîne de 2 octets créée par MKIR$ en une valeur de type Integer. Poids forts puis poids faible.

Exemple :
```
A%=CVIR(A$) 'Si  A$=chr$(3)+chr$(2) alors A&=(3*256)+(2*1)=770
```

Voir aussi : CVI, MKI$, MKIR$

8-16-44- DATE$ - Date Courante

Syntaxe : **DATE$**

Description : Cette instruction restitue une chaîne de 10 caractères de la forme jj/mm/aaaa, où jj est le jour(01-31), mm est le mois (01-12) et aaaa est l'année.

Exemple :
```
a$=DATE$ 'Résultat : a$="01/01/1996"
```

Voir aussi : TIME$, TIME, TIMER

8-16-45- DELAY – Attente passive

Syntaxe : **DELAY** <Durée>

Unité : Durée : millisecondes

Types acceptés : Durée : Entier

Description : Cette fonction réalise une attente suivant la durée spécifiée. Elle endort la tâche et provoque le basculement vers la tâche suivante.

Exemple :
```
DELAY 500  'Délai de 0.5 s.
DELAY Tempo1
```

8-16-46- DIFFUSE – génération d'événements

Syntaxe : **SIGNAL** <Nom>

Description : Cette instruction génère un événement.

Remarques : <Nom> doit être le même que dans l'instruction WAIT EVENT. Toutes les tâches contenant le WAIT EVENT associé prennent en compte l'événement et peuvent poursuivre leur exécution.

Exemple :
```
Program1                Program2
...                     ...
WAIT EVENT Ready        DIFFUSE Ready
...                     ...
```

Voir aussi : WAIT EVENT, SIGNAL

8-16-47- DIV – Division entière

Syntaxe : <Expression1> **DIV** <Expression2>

Types acceptés : Expression1, Expression2 : Entier

Description : Cet opérateur restitue le résultat d'une division entière.

Remarques : Cette opérateur restitue un entier.

Exemple :
```
a%=7
a%=a% DIV 2       'Résultat : a%=3
```

Voir aussi : MOD

8-16-48- EDIT – Saisie sur terminal

Syntaxe 1: <Variable>=**EDIT**(<Ligne>,<Colonne>,<Longueur>,<Signe>,<Point>)

Syntaxe 2: <Variable>=**EDIT**(<Ligne>,<Colonne>,<Longueur>,<Signe>,<Point>, <Code>)

Limites : Ligne : de 1 à 4 pour le Supervisor 80 ou de 1 à 16 pour le Supervisor 640.

Colonne : de 1 à 20 pour le Supervisor 80 ou de 1 à 40 pour la Supervisor 640.

Types acceptés : Variable : réel

Ligne, Colonne, Longueur : Entier

Signe, Point, Code : bit

Description : Cette fonction permet d'éditer un nombre réel avec le pupitre opérateur en utilisant les touches numériques, la touche DEL pour supprimer, la touche ENTER pour valider et ESC pour abandonner. La deuxième syntaxe permet de définir si la saisie est du type code d'accès (Code=1). Dans ce dernier cas, chaque appui sur une touche provoque l'affichage d'une étoile (*). Cette fonction provoque le basculement à une tâche suivante.

Remarques : <Ligne> et <Colonne> sont la position du premier caractère. <Longueur> est le nombre de caractères maximum. <Signe> indique si le signe du nombre peut être changé. <Point> indique si le point est permis. <Code> indique si la saisie est en mode code d'accès. La variable système KEY contient la dernière touche enfoncée. Si l'édition est abandonnée alors KEY=@ESC et sinon KEY=@RETURN.

Exemple :
```
A!=EDIT(1,10,4,0,0)      'Saisie en ligne 1 colonne 10 sur 4
                         'caractères, le signe et le point
                         'n'étant pas autorisés.
A!=EDIT(1,10,4,0,0,1)    'Même saisie avec utilisation
                         'du mode code d'accès
```

8-16-49- EDIT$

Syntaxe : <Variable>=**EDIT$**(<Ligne>,<Colonne>,<Longueur>)

Limites : Ligne : de 1 à 4 pour le Supervisor 80 ou de 1 à 16 pour le Supervisor 640.

Colonne : de 1 à 20 pour le Supervisor 80 ou de 1 à 40 pour la Supervisor 640.

Types acceptés : Variable : chaîne de caractères

Ligne, Colonne, Longueur : Entier

Description : Cette fonction permet d'éditer une chaîne de caractères en utilisant le pavé numérique, la touche DEL pour supprimer, la touche ENTER pour valider et ESC pour abandonner. Pour saisir un caractère alphanumérique, appuyer plusieurs fois sur la touche numérique associée, afin de changer le caractère afficher. L'enregistrement du caractère s'effectue automatiquement lorsque l'on n'appuie plus sur la touche numérique associée ou que l'on appuie sur une autre touche.

Remarques : <Ligne> et <Colonne> indique la position du premier caractère. <Longueur> est le nombre de caractères maximum. La variable système KEY contient la dernière touche enfoncée, si l'édition est abandonnée alors KEY=@ESC et sinon KEY=@RETURN.

Exemple :
```
A$=EDIT$(2,9,5)   'saisie en ligne 2, colonne 9
                  'de 5 caractères maxi.
```

8-16-50- END – Fin de bloc

Syntaxe : **END** {PROG | SUB |IF | WHILE}

Description : Fin de bloc**.**

Remarques : Vous devez spécifier un mot-clé après END

Exemples :
```
SUB   Manuel
...
END  SUB
```

Voir aussi : PROG, SUB, IF, WHILE

8-16-51- EXIT SUB – Sortie d'un sous-programme

Syntaxe : **EXIT SUB**

Description : Cette instruction permet de sortir d'un sous-programme

Voir aussi : SUB

8-16-52- EXP - Exponentiel

Syntaxe : **EXP** (<Expression>)

Types acceptés : Expression : réel

Description : Cette fonction restitue *e* (la base des logarithmes naturels) élevée à la puissance <Expression>.

Remarques : L'argument <Expression> doit être une expression numérique valide. La valeur retournée est de type réel.

Exemple :
```
a!=EXP(2)
```

Voir aussi : LOG

8-16-53- FLASHOK – Test la mémoire flash

Syntaxe : FLASHOK

Description : Cette fonction indique si les paramètres et les 10000 premières variables sont sauvegardés dans la mémoire flash**.**

Voir aussi : RAMOK, RAMTOFLASH, FLASHTORAM

8-16-54- FLASHTORAM – Transfert de la flash vers la ram

Syntaxe : FLASHTORAM

Description : Cette fonction transfert les paramètres et les 10000 premières variables de la mémoire flash dans la ram.

Voir aussi : RAMOK, RAMTOFLASH, FLASHOK

8-16-55- FONT – Sélection police

Syntaxe : **FONT=<**Valeur**>**

Types acceptés : <Valeur> : octet.

Description : Cette fonction permet de définir une police.

Remarques : Suivant <Valeur> :

Police 1 : 16 l x 40 c affichage texte blanc, fond noir, 3x4mm

Police 2 : 9 l x 30 c affichage texte blanc, fond noir, 4x7mm

Police 3 : 6 l x 20 c affichage texte blanc, fond noir, 12x20mm

Police 4 : 4 l x 15 c affichage texte blanc, fond noir, 16x22mm

Police 5 : 16 l x 40 c affichage texte noir, fond blanc, 3x4mm

Police 6 : 9 l x 30 c affichage texte noir, fond blanc, 4x7mm

Police 7 : 6 l x 20 c affichage texte noir, fond blanc, 12x20mm

Police 8 : 4 l x 15 c affichage texte noir, fond blanc, 16x22mm

Exemple :
```
FONT=1
Locate 2,15
Print "MODE AUTO"
```

8-16-56- FOR – Boucle FOR … NEXT

Syntaxe : FOR <Compteur>=<Début> TO <Fin> [STEP <Incrément>]

...

NEXT <Compteur>

Types acceptés : Compteur : Octet, Entier, Entier long

Description : Répète un groupe d'instructions un nombre fini de fois.

Remarques : FOR débute la structure de boucle FOR ... NEXT. FOR doit apparaître avant toutes les autres parties de la structure. <Compteur> est égal à <Fin>+1 à la fin de la

boucle. L'exécution de l'instruction Next provoque l'incrémentation de compteur et le basculement vers la tâche suivante. <Incrément> doit être une valeur positive.

Exemple :
```
FOR i%=1 TO 10
...
NEXT i%
```

Voir aussi : WHILE

8-16-57- FORMAT$

Syntaxe : **FORMAT$**(<Expression>,<Nombre>,<Précision>,'<Car>',<Signe>,<Align>)

Types acceptés : Expression : réel

Nombre, Précision : Octet

Car : chaîne de caractère

Signe, Align : Bit

Description : Cette fonction créé une chaîne formatée.

Remarques : L'argument <Expression> doit être une expression numérique valide. <Nombre> est le nombre minimal de caractères de la chaîne. <Précision> est le nombre de caractères après le point décimal. <Car> est le caractère de remplacement utilisé pour atteindre ce nombre minimal si nécéssaire. <Signe> indique si le caractère "+" ou "-" doit être ajouté en début de chaîne. <Align> indique si la chaîne est alignée à gauche.

Exemple :
```
a!=1.2562
b$=FORMAT$(a!,5,2," ",0,1)      ' a$="1.25 "
```

8-16-58- FRAC – Partie fractionnelle

Syntaxe : **FRAC**(<Expression>)

Types acceptés : <Expression> : réel

Description : Cette fonction restitue la partie fractionnelle de <Expression>.

Remarques : Cette fonction restitue une valeur réelle.

Exemple :
```
b!=3.0214
a!=FRAC(b!)        'Résultat a!=0.0214
```

Voir aussi : INT

8-16-59- GETDATE – Date courante

Syntaxe : **GETDATE**(<Année>,<Mois>,<Jour>,<JourDansLaSemaine>)

Types acceptés : <Année>, <Mois>, <Jour>, <JourDansLaSemaine> : Entier.

Description : Cette instruction lit la date courante.

Voir aussi : GETTIME

8-16-60- GETEVENT – Lecture des événements

Syntaxe : <Variable> = **GETEVENT**

Types acceptés : <Variable> : Entier

Description : Elle permet de consommer et de lire les événements qui ont été détectés.

Remarques : Chaque bit associé à un événement est à l'état 1 si l'événement a été détecté. Si un nouvel événement apparaît pendant l'exécution de la tâche événementielle, il est mémorisé et traité dès que possible.

Voir aussi : MODIFYEVENT

8-16-61- GETTIME – Heure courante

Syntaxe : **GETTIME**(<Heure>,<Minute>,<Seconde>)

Types acceptés : <Heure>, <Minute>, <Seconde> : Entier.

Description : Cette instruction lit l'heure courante.

Voir aussi : GETDATE

8-16-62- GOTO – Saut à une étiquette

Syntaxe : **GOTO** <Label>

Description : Réalise un branchement à une étiquette

Remarques : Les programmes avec beaucoup d'instructions GOTO peuvent devenir difficiles à lire et à mettre au point. Utilisez les structures de contrôle (FOR...NEXT, REPEAT...UNTIL, WHILE...END WHILE, IF...THEN...ELSE...END IF) à chaque fois que cela est possible. Une étiquette est un nom suivi du caractère ":". l'exécution de cette instruction provoque le basculement vers la tâche suivante.

Exemple :
```
GOTO Begin
...
Begin :
```

Voir aussi : JUMP, FOR, REPEAT, WHILE, IF, END

8-16-63- HALT – Arrêter une tâche

Syntaxe : **HALT** <Nom>

Description : Cette instruction est utilisée pour stopper une tâche en cours d'exécution ou suspendue.

Remarques : Cette fonction n'a pas d'effet sur une tâche déjà arrêtée. Elle n'affecte pas les mouvements en cours ni les buffers de mouvements**.**

Exemple :
```
Begin :
  Wait Inp(Puissance)=On
  RUN  Coupe
  Wait Inp(Puissance)=Off
  HALT Coupe
Goto Begin
```

Voir aussi : RUN, SUSPEND, CONTINUE

8-16-64- HLINE – Affichage d'une ligne horizontale

Syntaxe : **HLINE**(X1,Y1,X2,couleur)

Unité : X1, Y1, X2 : pixel

Limites : X1, X2 : de 1 à 240

Y1 : de 1 à 128

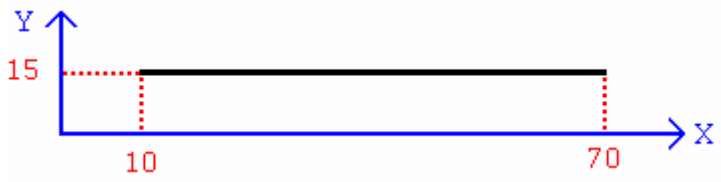
Types acceptés : X1, Y1, X2 : octet.

Couleur : Bit

Description : Trace une ligne horizontale avec le point de départ en X1, Y1 et d'arrivée en X2, Y1.

Remarques : Couleur permet de modifier la couleur du trait : noir (0) ou blanc (1)

Exemple :
```
HLINE(10,15,70,0)
```



8-16-65- ICALL – Appel d'un sous programme

Syntaxe : **ICALL** <Nom>

Description : Cette instruction est utilisée pour appeler un sous-programme défini par un bloc SUB. <Nom> est le nom du bloc du sous-programme.

Remarques : Un sous-programme ne peut pas s'appeler. Le système contient des sous-programmes pré-définis : _MENUMCS, _PARAMMCS, _MANUMCS, _VARIABMCS, _MEMORYMCS et _CLOCKMCS. L'exécution de cette instruction ne provoque pas un basculement de tâche.

Exemple : ICALL Move

Voir aussi : SUB, CALL

8-16-66- IF - IF…Then…Else

Syntaxe 1 : **IF** <Condition> **THEN**
{<Instruction1>}
...
ELSE
{<Instruction2>}
...
END IF

Syntaxe 2 : **IF** <Condition> **THEN** <Instruction1> **ELSE** <Instruction2>

Description : Permet l'exécution conditionnelle, basée sur l'évaluation d'une expression.

Remarques : Le mot-clé IF commence une structure de contrôle IF...THEN...ELSE...END IF . Il doit apparaître avant toute autre partie de la structure. <Condition> doit être une expression booléenne.

Si <Condition> est vraie alors <Instructions1> sont exécutées.

Si <Condition> est fausse alors <Instructions2> sont exécutées.

Exemple :

```
IF (a%>1) AND (a%<10) THEN
  Locate 1,1
  Print "Longueur 1"
Else
  Locate 2,1
  Print "Largeur 1"
END IF
```

Voir aussi : END

8-16-67- INKEY– Lit une touche sur le terminal

Syntaxe : <Variable>=**INKEY**

Types acceptés : Variable : Octet

Description : Cette fonction lit une touche à partir du clavier du terminal et retourne son code.

Remarques : Cette fonction est non bloquante pour la tâche. Si le buffer de réception est vide (aucune touche n'a été appuyée) cette fonction retourne 0.Cette fonction utilise le port de communication #1. Par défaut, le port de communication SERIAL1 sera utilisé. Si un pupitre opérateur est connecté au port SERIAL2, veuillez vous référer à la fonction OPEN pour affecter #1 au port SERIAL2.

Exemple :
```
REPEAT
A#=INKEY
UNTIL A#<>0
```

8-16-68- INP – Lecture d'une entrée TOR

Syntaxe : **INP** (<Entrée>)

Types acceptés : Entrées : Bit

Description : Cette fonction donne l'état d'une entrée TOR.

Remarques : <Entrée> doit représenter un nom d'entrée TOR. Le type de donnée retourné est Bit.

Exemple :
```
C~=INP(CouteauEnHaut)
```

Voir aussi : INPB, INPW, OUT, OUTB, OUTW

8-16-69- INPB – Lecture d'un bloc 8 entrées

Syntaxe : **INPB** (<Entrées>)

Types acceptés : Entrées : Octet

Description : Cette fonction retourne l'état d'un bloc de 8 entrées TOR.

Remarques : <Entrées> doit représenter le nom d'un bloc de 8 entrées. Le type de données retourné est octet.

Exemple :
```
B#=INPB(Data)
```

Voir aussi : INP, INPW, OUT, OUTB, OUTW

8-16-70- INPUT – Lecture de données

Syntaxe : **INPUT** <Numéro>, <Variable> [{,<Variable>}]

Types acceptés : Variable : Bit, Octet, Entier, Entier long, réel et chaîne de caractères

Numéro : #1 ou #2

Description : Lire des données à partir d'un port de communication et assigne les données à des variables. L'exécution de cette instruction provoque le basculement vers la tâche suivante.

Remarques : <Numéro> est le numéro utilisé pour ouvrir le port de communication avec l'instruction OPEN. Les données lues doivent apparaître dans le même ordre que la liste des variables.

Exemple :
```
OPEN "SERIAL1:"  AS #1
INPUT #1,A$,B%
CLOSE #1
```

Voir aussi : OPEN, PRINT, CLOSE

8-16-71- INPUT$ - Lecture de chaînes de caractères

Syntaxe : <Variable> =**INPUT$** <Numéro>, <Nombre>

Types acceptés : Variable : chaîne de caractères

Numéro : #1, #2 ou #3

Nombre : Octet

Description : Lit <Nombre> caractères à partir du port de communication (#1 ou #2) ou du fichier sauvegardé (#3) et les stocke dans une chaîne de caractères. L'exécution de cette instruction provoque le basculement vers la tâche suivante.

Remarques : <Numéro> est le numéro utilisé pour ouvrir le port de communication (#1 ou #2) ou le fichier sauvegardé (#3) avec l'instruction OPEN. <Variable> doit être une variable de type chaîne de caractères. La tâche se bloque sur cette instruction tant qu'elle ne reçoit pas le nombre effectif de caractère.

Exemple :

```
OPEN "SERIAL1:" AS #1
A$=INPUT$ #1,5    'Lit 5 caractères à partir du port de
                  'communication
CLOSE #1
OPEN «  File  :  » AS #3
A$=Input$ #3,5    'Lecture de 5 caractères dans le fichier
CLOSE #3
```

Voir aussi : OPEN, PRINT, CLOSE, SEEK

8-16-72- INPW – Lecture d'un bloc de 16 entrées

Syntaxe : **INPW** (<Entrées>)

Types acceptés : Entrées : Entier

Description : Cette fonction donne l'état d'un bloc de 16 entrées TOR.

Remarques : <Entrées> doit représenter le nom d'un bloc de 16 entrées TOR. Le type de données retourné est entier.

Exemple :

```
A%=INP(Bloc)
```

Voir aussi : INP, INPB, OUT, OUTB, OUTW

8-16-73- INSTR – cherche une sous-chaîne

Syntaxe : **INSTR**(<Chaîne1>,<Chaîne2>)

Types acceptés : chaîne1, chaîne2 : chaîne de caractères

Description : Cette fonction recherche une sous-chaîne dans une chaîne de caractères et retourne la position de la première occurrence de cette sous-chaîne.

Remarques : <Chaîne1> est la chaîne à rechercher dans <Chaîne2>

Exemple :

```
a$="Press ENTER to start"
EnterPos%=INSTR("ENTER",a$) 'Résultat : EnterPos%=7
```

Voir aussi : LEN

8-16-74- INT – Partie entière

Syntaxe : **INT** (<Expression>)

Types acceptés : Expression : réel

Description : Cette fonction restitue la partie entière d'<Expression>.

Remarques : L'argument <Expression> doit être une expression numérique valide.

Exemple :
```
b!=25.36
a!=INT(b!)        'Résultat : a!=25
```

Voir aussi : FRAC

8-16-75- JUMP

Syntaxe : **JUMP** <Label>

Description : Réalise un branchement à une étiquette

Remarques : Les programmes avec beaucoup d'instructions JUMP peuvent devenir difficiles à lire et à mettre au point. Utilisez les structures de contrôle (FOR...NEXT, REPEAT...UNTIL, WHILE...END WHILE, IF...THEN...ELSE...END IF) à chaque fois que cela est possible. Une étiquette est un nom suivi du caractère ":". Cette instruction ne provoque pas de basculement vers la tâche suivante.

Exemple :
```
JUMP Begin
...
Begin :
```

Voir aussi : GOTO, FOR, REPEAT, WHILE, IF, END

8-16-76- KEY – Dernière touche

Syntaxe : **KEY**

Description : Cette variable système contient la dernière touche pressée.

Remarques : Cette variable doit être utilisée avec les instructions EDIT et WAIT KEY. La variable key est locale à une tâche.

Exemple :
```
WAIT KEY
IF KEY=@F1 THEN CALL ...
IF KEY=@F2 THEN CALL ...
IF KEY=@F3 THEN CALL ...
```

Voir aussi : EDIT, WAIT KEY

8-16-77- KEYDELAY – Délai avant répétition d'une touche

Syntaxe : **KEYDELAY** = <Expression>

Unité : Expression : en trente deuxième de seconde.

Types acceptés : Expression : Octet

Description : Cette instruction définit le délai avant la répétition automatique d'une touche du pupitre opérateur lorsque celle-ci reste appuyée.

Remarques : La valeur par défaut du délai est 1 seconde (32).

Exemple :
```
KEYDELAY = 10
```

Voir aussi : KEYREPEAT

8-16-78- KEYREPEAT – Période de répétition d'une touche

Syntaxe : **KEYREPEAT**=<Expression>

Unité : Expression : en 32ème de seconde.

Types acceptés : Expression : Octet

Description : Cette instruction définit le délai séparant chaque répétition automatique d'une touche du pupitre opérateur lorsque celle-ci reste appuyée.

Remarques : La valeur par défaut de la période de répétition est 0.3 seconde (10).

Exemple : `KEYREPEAT = 5`

Voir aussi : KEYDELAY

8-16-79- LCASE$ - Minuscules

Syntaxe : <Expression> = **LCASE$**(<Chaîne>)

Types acceptés : Chaîne : chaîne de caractères

Description : Cette fonction retourne une chaîne dans laquelle toutes les lettres de l'argument ont été converties en minuscules.

Remarques : L'argument <Expression> doit être une chaîne de caractères. Seules les lettres majuscules sont converties en minuscules; les autres lettres ne sont pas modifiées**.**

Exemple :
```
a$="Capteur1"
b$=LCASE$(a$) 'Résultat : b$="capteur1"
```

Voir aussi : UCASE$

8-16-80- LED – Pilotage des leds

Syntaxe : **LED**(numéro)=Etat

Types acceptés : Etat : octet.

Description : Cette fonction permet de piloter les leds.

Remarques : Numéro représente la led : de @F1 à @F6 ou @ALARM ou @HELP

Etat représente les différents états de la led : éteint (0), allumé (1), clignotant (2).

Exemple :
```
LED(@ALARM)=2   'led alarme clig
```

8-16-81- LEFT$ - Partie gauche d'une chaîne

Syntaxe : **LEFT$**(<Chaîne>,<Nombre>)

Types acceptés : Chaîne : chaîne de caractères

Nombre : Entier

Description : Cette fonction retourne les <Nombre> caractères de gauche d'une chaîne.

Remarques : Pour trouver le nombre de caractères dans <Chaîne>, utilisez LEN(<Chaîne>).

Exemple :
```
a$="Capteur1"
b$=LEFT$(a$,6) 'Résultat : b$="Capteur"
```

Voir aussi : RIGHT$, LEN

8-16-82- LEN– Longueur d'une chaîne

Syntaxe : **LEN**(<Chaîne>)

Description : Cette fonction retourne le nombre de caractères d'une chaîne.

Exemple :
```
a$="Capteur1"
b%=LEN(a$) 'Résultat : b%=8
```

Voir aussi : INSTR

8-16-83- LOCATE – Positionne le curseur

Syntaxe : **LOCATE** <Ligne>,<Colonne>

Limites : Ligne : de 1 à 4 pour le Supervisor 80 ou de 1 à 16 pour le Supervisor 640.

Colonne : de 1 à 20 pour le Supervisor 80 ou de 1 à 40 pour la Supervisor 640.

Types acceptés : Ligne, Colonne : Octet

Description : Cette fonction est utilisée pour sélectionner la position du curseur sur le pupitre opérateur.

Remarques : Cette fonction utilise le port de communication #1. Par défaut, le port de communication SERIAL1 sera utilisé. Si un pupitre opérateur est connecté au port SERIAL2 veuillez vous référer à la fonction OPEN pour affecter #1 au port SERIAL2.

Exemple :
```
LOCATE 1,1
PRINT "<MENU PRINCIPAL>"
```

8-16-84- LOG - Logarithme

Syntaxe : **LOG** (<Expression>)

Types acceptés : Expression : réel

Description : Retourne le logarithme naturel de <Expression>

Remarques : <Expression> doit être une expression numérique.

Exemple :
```
a!=LOG(1.2)
```

Voir aussi : EXP

8-16-85- LONGTOINTEGER – Conversion Entier long / Entier

Syntaxe : **LONGTOINTEGER**(<Expression>)

Types acceptés : Expression : Entier long

Description : Cette fonction convertit une donnée de type entier long en une donnée de type entier.

Exemple :
```
A&=Time
B%=LongToInteger(A&)
```

8-16-86- LTRIM$ - Enlève les espaces à gauche

Syntaxe : **LTRIM$**(<Expression>)

Description : Retourne une copie d'une chaîne sans les espaces se trouvant à gauche.

Remarques : <Expression> doit être une chaîne de caractères.

Exemples :
```
a$="     Menu    "
b$=LTRIM$(a$)            ' Résultat b$="Menu    "
```

Voir aussi : RTRIM$

8-16-87- MID$ - Partie d'une chaîne

Syntaxe : **MID$**(<Chaîne>, <Début>, <Longueur>)

Types acceptés : Chaîne : chaîne de caractères

Début, Longueur : Octet

Description : Cette fonction retourne une partie d'une chaîne.

Remarques : <Début> définit le début de la sous-chaîne extraite et <Longueur> le nombre de caractères à extraire.

Exemple :
```
a$="Menu principal     "
b$=MID$(a$,6,9) ' Résultat : b$="principal"
```

Voir aussi : LEFT$, RIGHT$

8-16-88- MKI$ - Conversion Integer / Chaîne

Syntaxe : <Chaîne >=**MKI$**(<Variable>)

Types acceptés : Chaîne : chaîne de caractères de 2 octets
Variable : Entier

Description : La fonction MKI$ sert à convertir une valeur entière de type Integer en une chaîne de 2 octets. Poids faible puis poids fort.

Exemple :
```
A$=MKI$(A%)        'Si A%=256 alors    A$=01
```

Voir aussi : MKIR$, CVI, CVIR

8-16-89- MKIR$ - Conversion Integer reverse / Chaîne

Syntaxe : <Chaîne >=**MKIR$**(<Variable>)

Types acceptés : Chaîne : chaîne de caractères de 2 octets
Variable : Entier

Description : La fonction MKIR$ sert à convertir une valeur entière de type Integer en une chaîne de 2 octets. Poids fort puis poids faible.

Exemple :
```
A$=MKI$(A%)        'Si A%=770 alors    A$=32
```

Voir aussi : MKI$, CVI, CVIR

8-16-90- MKL$ - Conversion Long / Chaîne

Syntaxe : <Chaîne >=**MKL$**(<Expression>)

Types acceptés : Chaîne : chaîne de caractères de 4 octets
Expression : Entier long

Description : La fonction MKL$ sert à convertir une valeur de type Long en une chaîne de 4 octets. Poids faible puis poids fort.

Exemple :
```
A$=MKL$(A&)        'Si A&=66306 alors  A$=2310
```

Voir aussi : MKLR$, CVL, CVLR

8-16-91- MKLR$ - Conversion Long reverse / Chaîne

Syntaxe : <Chaîne >=**MKLR$**(<Expression>)

Types acceptés : Chaîne : chaîne de caractères de 4 octets
Expression : Entier long

Description : La fonction MKLR$ sert à convertir une valeur de type Long en une chaîne de 4 octets. Poids fort puis poids faible.

Exemple :
```
A$=MKL$(A&)        'Si A&=66305 alors  A$=0132
```

Voir aussi : MKL$, CVL, CVLR

8-16-92- MOD - Modulo

Syntaxe : <Expression1> **MOD** <Expression2>

Types acceptés : Expression1, Expression2 : Octet, Entier, Entier long

Description : Cet opérateur restitue le reste d'une division entière.

Exemple :
```
a%=5
a%=a% MOD 2 'Résultat : a%=1
```

Voir aussi : DIV

8-16-93- MODIFYEVENT– Configuration des événements

Syntaxe : MODIFYEVENT (<Condition>, <Seuil compteur1>, <Seuil compteur2>, <Masque>, <Durée>)

Limites : <Durée> : de 10ms à 30000ms

Types acceptés : <Condition> : Entier

<Seuil Compteur1> : Entier

<Seuil Compteur2> : Entier

<Durée> : Entier

Description : Elle permet de configurer les événements souhaités.

Remarques : <Condition> :

↳ Bits 0...7 : activation des entrées n° 1 à n° 8 de la carte entrée. Un front montant générera l'événement. L'entrée tient compte du filtre et de l'inversion paramétrés dans l'écran de configuration de la carte.

↳ Bit 8 : Seuil du compteur 1 atteint

↳ Bit 9 : Seuil du compteur 2 atteint

↳ Bit 10 : SDOEvent

↳ Bit 11 : PDOEvent

↳ Bit 12 : Base de temps

<Durée> : Durée de la base de temps entre 10 ms et 30000 ms. Si la base de temps n'est pas utilisée, la valeur de durée rentrée ne sera pas traitée.

Après l'affectation de la configuration, la tâche événementielle est exécutée dès qu'au moins un événement est détecté. Le temps maxi entre l'apparition de l'événement et son traitement est égal au « temps de vieillissement » d'une tâche.

Si l'on désire par la suite modifier la configuration des événements, l'instruction MODIFYEVENT doit être traitée dans une tâche basic normale ou dans la tâche événementielle à condition qu'elle soit placée après GETEVENT.

Voir aussi : GETEVENT

8-16-94- NOT – Opérateur complément

Syntaxe : **NOT**(<Expression>)

Types acceptés : Expression : Bit, Octet, Entier

Description : La fonction NOT retourne le complément.

Remarques : <Expression> doit être une expression entière valide.

Exemple :
```
a%=0FF00h
b%=NOT a%  'Résultat b%=00FFh
```

Voir aussi : AND, OR, XOR

8-16-95- OPEN – Ouvre un port de communication ou un fichier

Syntaxe 1 : **OPEN** <PortCommunication> **AS #**<Numéro>

Syntaxe 2 : **OPEN** « File : » **AS** #3

Description : La première syntaxe autorise les opérations de lecture / écriture sur un port de communication. La seconde syntaxe autorise l'ouverture du fichier sauvegardé de 128 Koctets du SUPERVISOR.

Remarques : Vous devez ouvrir un port de communication avant toute opération d'entrée / sortie. <PortCommunication> est une chaîne de caractères qui définit les paramètres de la façon suivante :

"SERIALn:[vitesse [, donnée [, parité [, stop]]]]"

n: Numéro physique 1 ou 2

Vitesse : 150, 300, 600, 1200, 2400, 4800, 9600 bauds.

Donnée : 7 ou 8 bits

Parité : E (paire), O (impaire), M (marque), S (espace) or N (sans).

Stop : 1 ou 2 bits

<Numéro> définit le numéro du canal de communication utilisé par les fonctions. Lorsque l'on écrit "SERIAL2:" As #1, les paramètres de vitesse, donnée, parité, stop sont ceux de la fenêtre Serial2 de la page configuration et sont pris en compte lors de la compilation des tâches.

Pour ouvrir le fichier du SUPERVISOR, il est nécessaire d'utiliser intégralement la syntaxe n°2.

Exemple 1 :
```
Dialog 160 relié au SERIAL2 : SERIAL2 affecté au canal 1
OPEN "SERIAL2:9600,8,N,1" As #1
PRINT "<MENU PRINCIPAL>";
```

Exemple 1 :
```
Open «  File  :  » As #3 'Ouvre le fichier du SUPERVISOR
```

Voir aussi : INPUT, PRINT, CLOSE

8-16-96- OR - Opérateur ou

Syntaxe : <Expression1> **OR** <Expression2>

Types acceptés : Expression1, Expression2 : Bit, Octet, Entier

Description : Cette fonction effectue un OU binaire entre deux expressions.

Remarques : <Expression1> et <Expression2> doivent être du même type. Cette fonction restitue le même type de donnée que ses arguments

Exemple :
```
A%=A% OR 000FFh
```

Voir aussi : AND, NOT, XOR et IF

8-16-97- OUT – Ecriture d'une sortie

Syntaxe : **OUT** (<Sortie>) = <Expression>

Types acceptés : Expression : Bit

Description : Cette fonction envoie un état logique à une sortie TOR.

Remarques : <Sortie> doit représenter le nom d'une sortie

Exemple : OUT(Couteau)=ON

Voir aussi : INP, INPB, INPW, OUTB, OUTW

8-16-98- OUTEMPTY – Etat du buffer de sortie

Syntaxe : <Expression>=**OUTEMPTY** (<Numéro>)

Types acceptés : <Expression> : bit

Description : Cette fonction indique si le buffer de sortie est vide et que le dernier caractère a été envoyé.

Remarques : <Numéro> est le numéro utilisé pour ouvrir le port de communication avec l'instruction OPEN

Exemple : `WAIT OUTEMPTY(#1)`

Voir aussi : CARIN, CAROUT

8-16-99- OUTB – Ecriture d'un bloc de 8 sorties

Syntaxe : **OUTB** (<Sorties>) = <Expression>

Types acceptés : Expression : Octet

Description : Cette fonction envoie des états logiques à un bloc de 8 sorties TOR.

Remarques : <Sorties> doit représenter le nom d'un bloc de 8 sorties.

Exemple : `OUTB(Bloc1)=0Fh`

Voir aussi : INP, INPB, INPW, OUT, OUTW

8-16-100- PIXEL – Affiche un point

Syntaxe : **PIXEL**(X,Y,Couleur)

Unité : X, Y : pixel

Limites : X : de 1 à 240

Y : de 1 à 128

Types acceptés : X,Y : octet.

Couleur : Bit

Description : Cette fonction permet d'afficher un pixel de coordonné X et Y.

Remarques : Couleur permet de définir la couleur : noir (0) ou blanc (1)

Exemple : `PIXEL(23,15,0) 'affichage d'un point noir de coordonnée 23,15`

8-16-101- PLCINIT – Initialisation des fonctions PLC

Syntaxe : PLCINIT(<Tableau entrées>,< Tableau entrées précédentes>, <Tableau sorties>, <Tableau masque sorties>)

Description : Cette fonction indique au système les tableaux de variable à utiliser

Remarques : <Tableau entrées>, <Tableau entrées précédentes> : tableau d'entier long contenant autant d'éléments que le système contient de cartes d'entrées.

<Tableau sorties>, <Tableau masques> : tableau d'entier contenant autant d'éléments que le système contient de cartes de sorties

<Tableau masques> contient les masques des sorties utilisées par le PLC (bit à 1 => sortie utilisée par le PLC)

Exemple : Masque[1]=0FFFFh

Masque[2]=0FFFFh

PlcInit(Entrees,EntreesOld,Sorties,Masque)

Voir aussi : PLCINP, PLCINPB, PLCINPW, PLCINPPE, PLCINPNE, PLCOUT, PLCOUTB, PLCOUTW

8-16-102- PLCINP – Lecture d'une entrée TOR

Syntaxe : PLCINP (<Entrée>) ou PLCINP (<Numéro Carte>, <Numéro Entrée>)

Types acceptés : <Entrées> : Bit

<Numéro Carte>, <Numéro Entrée> : Octet

Description : Cette fonction donne l'état d'une entrée TOR PLC.

Remarques : <Entrée> doit représenter un nom d'entrée TOR. Le type de donnée retourné est Bit.

Exemple : C~=PLCINP(CouteauEnHaut)

Voir aussi : PLCINPB, PLCINPW, PLCOUT, PLCOUTB, PLCOUTW

8-16-103- PLCINPB – Lecture d'un bloc 8 entrées

Syntaxe : PLCINPB (<Entrées>)

Types acceptés : Entrées : Octet

Description : Cette fonction retourne l'état d'un bloc de 8 entrées TOR.

Remarques : <Entrées> doit représenter le nom d'un bloc de 8 entrées. Le type de données retourné est octet.

Exemple : B#=PLCINPB(Data)

Voir aussi : PLCINP, PLCINPW PLCOUT, PLCOUTB, PLCOUTW

8-16-104- PLCINPW – Lecture d'un bloc de 16 entrées

Syntaxe : PLCINPW (<Entrées>)

Types acceptés : Entrées : Entier

Description : Cette fonction donne l'état d'un bloc de 16 entrées TOR.

Remarques : <Entrées> doit représenter le nom d'un bloc de 16 entrées TOR. Le type de données retourné est entier.

Exemple : A%=INP(Bloc)

Voir aussi : PLCINP, PLCINPB, PLCOUT, PLCOUTB, PLCOUTW

8-16-105- PLCINPPE – Lecture d'un front montant d'une entrée TOR PLC

Syntaxe : PLCINPPE (<Entrée>) ou PLCINPPE (<Numéro Carte>, <Numéro Entrée>)

Types acceptés : <Entrées> : Bit

<Numéro Carte>, <Numéro Entrée> : Octet

Description : Cette fonction indique si un front montant s'est produit sur une entrée TOR PLC.

Remarques : <Entrée> doit représenter un nom d'entrée TOR. Le type de donnée retourné est Bit.

Exemple : If PLCINPPE(CouteauEnHaut) Then goto FrontDetecte

Voir aussi : PLCINPB, PLCINPW, PLCOUT, PLCOUTB, PLCOUTW

8-16-106- PLCINPNE – Lecture d'un front descendant d'une entrée TOR

Syntaxe : PLCINPNE (<Entrée>) ou PLCINPNE (<Numéro Carte>, <Numéro Entrée>)

Types acceptés : <Entrées> : Bit

<Numéro Carte>, <Numéro Entrée> : Octet

Description : Cette fonction indique si un front descendant s'est produit sur une entrée TOR PLC.

Remarques : <Entrée> doit représenter un nom d'entrée TOR. Le type de donnée retourné est Bit.

Exemple : C~=PLCINPNE(CouteauEnHaut)

Voir aussi : PLCINPB, PLCINPW PLCOUT, PLCOUTB, PLCOUTW

8-16-107- PLCOUT – Ecriture d'une sortie

Syntaxe : PLCOUT (<Sortie>) = <Expression> ou

PLCOUT (<Numéro carte>, <Numéro sortie>) = <Expression>

Types acceptés : Expression : Bit

<Numéro Carte>, <Numéro Sortie> : Octet

Description : Cette fonction affecte l'état logique du bit image.

Remarques : <Sortie> doit représenter le nom d'une sortie

Exemple : PLCOUT(Couteau)=ON

…

If PLCOUT(Voyant) Then goto Alarm

Voir aussi : PLCINPB, PLCINPW PLCOUT, PLCOUTB, PLCOUTW

8-16-108- PLCOUTB – Ecriture d'un bloc de 8 sorties

Syntaxe : PLCOUTB (<Sorties>) = <Expression>

Types acceptés : Expression : Octet

Description : Cette fonction affecte l'état logique des 8 bits images associés.

Remarques : <Sorties> doit représenter le nom d'un bloc de 8 sorties.

Exemple : PLCOUTB(Bloc1)=0Fh

Voir aussi : PLCINPB, PLCINPW PLCOUT, PLCOUTB, PLCOUTW

8-16-109- PLCOUTW - Ecriture d'un bloc de 16 sorties

Syntaxe : PLCOUTW (<Sorties>) = <Expression>

Types acceptés : Expression : Entier

Description : Cette fonction affecte l'état logique des 16 bits images associés.

Remarques : <Sorties> doit représenter le nom d'un bloc de 16 sorties.

Exemple : PLCOUTW(Bloc1)=0FFFFh

Voir aussi : PLCINPB, PLCINPW PLCOUT, PLCOUTB, PLCOUTW

8-16-110- PLCREADINPUTS – Lecture des entrées PLC

Syntaxe : PLCREADINPUTS

Description : Cette fonction lit les entrées PLC et les mémorise dans les tableaux bits images.

Voir aussi : PLCINP, PLCINPB, PLCINPW, PLCINPPE, PLCINPNE, PLCOUT, PLCOUTB, PLCOUTW

8-16-111- PLCWRITEOUPUTS – Ecriture des sorties PLC

Syntaxe : PLCREADINPUTS

Description : Cette fonction lit les entrées PLC et les mémorise dans les tableaux bits images.

Voir aussi : PLCINP, PLCINPB, PLCINPW, PLCINPPE, PLCINPNE, PLCOUT, PLCOUTB, PLCOUTW

8-16-112- POWERFAIL – Gestion des microcoupures

Syntaxe : **POWERFAIL**= <ON|OFF>

Description : Si POWERFAIL=ON, activation de la gestion des microcoupures.

Remarques : A chaque redémarrage du SUPERVISOR, cette instruction est activée automatiquement.

8-16-113- PRINT – Ecrit sur le port de communication

Syntaxe : **PRINT** [#<Numéro>], <Expression> [{ [; | ,] <Expression>}] [; | ,]

Description : Ecrit des données sur un port de communication ou dans le fichier sauvegardé de 128 Koctets du SUPERVISOR.

Remarques : <Numéro> est le numéro utilisé pour ouvrir le port de communication avec l'instruction OPEN. Si celui-ci est 3, il permet d'accéder au fichier sauvegardé du SUPERVISOR. Si <Numéro> n'est pas indiqué, il sera pris par défaut pour le port de communication 1. Un point-virgule à la fin de cette instruction signifie que le prochain caractère est imprimé immédiatement après le dernier caractère. Une virgule signifie que le prochain caractère est imprimé à la prochaine ligne (en ajoutant un retour chariot). Print équivaut à Print #1. L' instruction Print sur une variable réelle n'affiche que sa partie entière, utiliser l'instruction Format$ pour accéder à la partie décimale. Si le buffer de transmission est plein, la tâche se bloque jusqu'à ce qu'une place se libère dans le buffer.

Exemple :
```
PRINT #1,A$,B%    ' Ecriture vers le port de communication 1
PRINT "LONGUEUR"  ' Ecriture vers le port de communication 1
PRINT #3, Chr$(1);«  123456  »;Chr$(2);'Ecriture de 8 caractères
                                        'dans le fichier sauvegardé
```

Voir aussi : OPEN, PRINT, CLOSE, SEEK

8-16-114- PROG – Début d'un programme

Syntaxe : **PROG**

Description : Ce mot-clé commence un bloc de programme principal. Il est également utilisé pour idendifier la fin d'un bloc de programme principal lorsqu'il est précédé de END.

Remarques : Un et seulement un bloc PROG - END PROG doit être défini dans un programme

Exemple :
```
PROG
...
END PROG
```

Voir aussi : END

8-16-115- RAMOK – Test la mémoire ram

Syntaxe : **RAMOK**

Description : Cette fonction indique si les paramètres et les variables en flash on été utilisés au démarrage pour palier au défaut de la mémoire ram.

Remarques : Cette fonction indique si au dernier redémarrage du SUPERVISOR le checksum de contrôle des données en RAM a été correct.

Si RAMOK=1, démarrage normal

Si RAMOK=0 et zone de copie des données en flash non vierge, le SUPERVISOR transfert la zone de data flash dans la zone ram et lance les tâches. Si RAMOK=0 et zone de copie des données en flash vierge, le SUPERVISOR ne lance pas les tâches et indique une erreur 20 sur son afficheur de status.

Voir aussi : FLASHOK, RAMTOFLASH, FLASHTORAM

8-16-116- RAMTOFLASH – Transfert de la ram vers la flash

Syntaxe : **RAMTOFLASH**

Description : Cette fonction transfert les paramètres et les 1000 premières variables dans la mémoire flash.

Voir aussi : RAMOK, FLASHTORAM, FLASHOK

8-16-117- READKEY – Retourne l'état du clavier du terminal

Syntaxe : <Variable>=**READKEY**

Types acceptés : Variable : Octet

Description : Cette fonction lit l'état du clavier du terminal et retourne le code de la touche restée appuyée .

Remarques : Cette fonction est non bloquante pour la tâche. Si aucune touche n'est appuyé, cette fonction retourne 0.Cette fonction utilise le port de communication #1. Par défaut, le port de communication SERIAL1 sera utilisé. Si un pupitre opérateur est connecté au port SERIAL2, veuillez vous référer à la fonction OPEN pour affecter #1 au port SERIAL2. Fontion utilisée pour jouer avec la durée d'appuie d'une touche (ex: mouvement JOG+ et JOG- d'un axe).

Exemple :

```
REPEAT
A#=READKEY
Until A#<>0
```

8-16-118- REALTOLONG – Conversion réel / entier long

Syntaxe : **REALTOLONG**(<Expression>)

Types acceptés : Expression : réel

Description : Cette fonction convertit une donnée de type réel en une donnée de type entier long.

Exemple :

```
A!=Edit(1,1,4,0,0)
B&=RealToLong(A!)
```

8-16-119- REALTOINTEGER – Conversion réel / entier

Syntaxe : **REALTOINTEGER**(<Expression>)

Types acceptés : Expression : réel

Description : Cette fonction convertit une donnée de type réel en une donnée de type entier.

Exemple :

```
A!=Edit(1,1,4,0,0)
B%=RealToInteger(A!)
```

8-16-120- REALTOBYTE - Conversion réel / octet

Syntaxe : **REALTOBYTE**(<Expression>)

Types acceptés : Expression : réel

Description : Cette fonction convertit une donnée de type réel en une donnée de type octet.

Exemple :

```
A!=Edit(1,1,4,0,0)
B#=RealToInteger(A!)
```

8-16-121- REPEAT – Repeat…Until

Syntaxe : **REPEAT**

{<Instructions>}

UNTIL <Condition>

Description : Cette structure permet au système d'exécuter une série d' instructions dans une boucle aussi longtemps que la condition donnée est fausse.

Remarques : Dans la structure REPEAT ... UNTIL les <Instructions> sont exécutées au moins une fois même si la condition est vraie. L'exécution de l'instruction « Until » provoque le basculement vers la tâche suivante.

Exemple :
```
a%=0
REPEAT
    PRINT #1,a%
    a%=a%*2
UNTIL a%>100
```

Voir aussi : WHILE

8-16-122- RESTART – Redémarrage du système

Syntaxe : **RESTART**

Description : Redémarre le système de la même manière qu'une mise sous tension.

Remarques : En lecture, si RESTART=VRAI ⇒ Reset à chaud.
si RESTART=FAUX ⇒ Reset à froid.

8-16-123- RIGHT$ - Partie droite d'une chaîne

Syntaxe : **RIGHT$**(<Chaîne>,<Nombre>)

Types acceptés : chaîne : chaîne de caractères
Nombre : Entier

Description : Cette fonction retourne les <Nombre> caractères de droite d'une chaîne.

Remarques : Pour trouver le nombre de caractères dans <Chaîne>, utilisez LEN(<Chaîne>).

Exemple :
```
a$="Capteur1"
b$=RIGHT$(a$,1) 'Résultat : b$="1"
```

Voir aussi : LEFT$

8-16-124- RTRIM$ - Enlève les espaces à droite

Syntaxe : **RTRIM$** (<Expression>)

Types acceptés : Expression : chaîne de caractères

Description : Retourne une copie d'une chaîne sans les espaces se trouvant à droite.

Exemple :
```
a$="    Menu   "
b$=LTRIM$(a$)          ' Résultat b$="   Menu"
```

Voir aussi : LTRIM$

8-16-125- RUN – Lance une tâche

Syntaxe : **RUN** <Nom>

Description : Cette instruction est utilisée pour lancer une tâche stoppée (ex : tâche déclarée en démarrage manuel).

Remarques : Cette fonction n'a pas d'effets sur une tâche suspendue ou déjà lancée.

Exemple :
```
Debut:
    Wait Inp(Puissance)=On
    RUN  Couteau
    Wait Inp(Puissance)=Off
    HALT Couteau
```

```
Goto Debut
```

Voir aussi : CONTINUE, HALT, SUSPEND

8-16-126- SEEK – Déplacement dans un fichier sauvegardé

Syntaxe 1 : **SEEK** #3,<Déplacement Long>

Syntaxe 2 : <Variable> = **SEEK** #3

Types acceptés : <Déplacement long>, <Variable> : entier long

Description : La syntaxe 1 permet de se déplacer dans le fichier sauvegardé du nombre de caractère spécifié par <Déplacement long>. Le déplacement se fait à partir de la position courante. La syntaxe 2 permet de connaître la position courante dans le fichier sauvegardé.

Remarques : Le premier caractère est situé à la position zéro.

Exemple :

```
P&=Seek(#3)          'Rapport nominal  : ratio 0.5
Seek #3, P&+100      'Déplacement sur le 100ème caractère à partir
                     'de la position courante
```

Voir aussi : OPEN, INPUT$, PRINT

8-16-127- SETDATE – Fixe la date

Syntaxe : **SETDATE**(<Année>,<Mois>,<Jour>,<JourDanslaSemaine>)

Types acceptés : Année, Mois, Jour, JourDanslaSemaine : Entier

Description : Cette instruction fixe la date courante.

Voir aussi : GETDATE, SETTIME

8-16-128- SETINP – Filtrage et inversion des entrées

Syntaxe : **SETINP** (<Nom>,<Inversion>,<Filtre>)

Unité : Filtre : millisecondes

Types acceptés : Inversion : Entier long

Filtre : Octet

Description : Cette fonction définit le masque d'inversion des entrées et la période de filtrage d'une carte.

Remarques : <Inversion > est un entier long dans lequel les bits représentent l'inversion de chacune des entrées. <Filtre> peut être défini lors de la configuration de la carte entrée.

Exemple :

```
SETINP(INPUTS11,4,10)    ' Inversion de la deuxième entrée
                         ' de la carte INPUTS1
                         ' et filtrage à 10 ms
```

8-16-129- SETOUT – Inversion des sorties

Syntaxe : **SETOUT** (<Nom>,<Inversion>)

Types acceptés : Inversion : Entier long

Description : Cette fonction définit le masque d'inversion des sorties d'une carte.

Remarques : <Inversion > est un entier long dans lequel les bits représentent l'inversion de chacune des sorties. Ce paramètre peut être défini lors de la configuration de la carte sortie.

Exemple :
```
SETOUT(OUTPUTS1,3)        ' inversion des 2 premières sorties
                          ' de la carte OUTPUTS1
```

8-16-130- SETTIME – Fixe l'heure

Syntaxe : **SETTIME**(<Heures>,<Minutes>,<Secondes>)

Types acceptés : <Heures>, <Minutes> et <Secondes> : Entier.

Description : Cette instruction fixe l'heure courante.

Voir aussi : GETTIME, SETDATE

8-16-131- SETUPCOUNTER – Configure le compteur

Syntaxe : **SETUPCOUNTER**(<Compteur>,<Entrée>,<Inversion>,<Filtre Inversé>)

Types acceptés : <Compteur> : 1 ou 2

<Entrée> : Octet

<Inversion>, <Filtre> : bit

Description : Cette instruction permet de configurer le compteur

Remarques : <Compteur> : Numéro du compteur (1 ou 2)

<Entrée> : Numéro de l'entrée correspondante sur la carte d'entrée

<Inversion> : Choix du front : 0 pour front montant, 1 pour front descendant

<Filtre Inversé> : Validation du filtre : 1 pour sans filtrage, 0pour filtre de 2 ms.

Voir aussi : COUNTER_S , CLEARCOUNTER

8-16-132- SGN - Signe

Syntaxe : **SGN** (<Expression>)

Types acceptés : Expression : Entier long, réel

Description : Cette fonction retourne un réel égal à -1 pour les nombres négatifs, 1 pour les nombres positifs et 0 pour les nombres nuls.

Exemple :
```
a!=SGN(10) 'Résultat : a!=1
```

8-16-133- SIGNAL – Génération d'événement

Syntaxe : **SIGNAL** <Nom>

Description : Cette instruction génère un événement.

Remarques : <Nom> doit être le même que dans l'instruction WAIT EVENT. La première tâche contenant le WAIT EVENT associé prend en compte l'événement et peut poursuivre son exécution.

Exemple :
```
Program1                  Program2
...                       ...
WAIT EVENT Ready          SIGNAL Ready
...                       ...
```

Voir aussi : WAIT EVENT, DIFFUSE

8-16-134- SIN - Sinus

Syntaxe : **SIN** (<Expression>)

Types acceptés : Expression : réel

Description : Cette instruction retourne le sinus de <Expression>. <Expression>est exprimée en radians.

Exemple : <Expression> doit être une expression numérique.

Voir aussi : COS, ARCTAN, TAN

8-16-135- SPACE$ - Chaîne d'espaces

Syntaxe : **SPACE$**(<Nombre>)

Limites : Nombre : de 1 à 255

Types acceptés : Nombre : Octet

Description : Cette fonction retourne une chaîne constituée d'espaces.

Exemple :
```
a$=SPACE$(10) 'Résultat a$="          "
```

Voir aussi : STR$, VAL

8-16-136- SQR – Racine carrée

Syntaxe : **SQR** (<Expression>)

Types acceptés : Expression : réel

Description : Cette fonction retourne la racine carrée de <Expression>.

Exemple :
```
a!=SQR(2)
```

8-16-137- STATUS – Etat d'une tâche

Syntaxe : **STATUS** (<Nom>)

Description : Cette fonction retourne l'état d'une tâche

Remarques : Les valeurs possibles sont :

0 : La tâche est stoppée

1 : La tâche est suspendue

2 : La tâche est en cours d'exécution

Exemple :
```
Run Coupe
Wait Status(Coupe)=0
```

8-16-138- STRING$ - Création de chaîne

Syntaxe : **STRING$**(<Nombre>, <Code>)

Limites : Nombre, Octet : de 0 à 255

Types acceptés : Nombre, Code : Octet

Description : Cette fonction retourne une chaîne de caractères dont tous les caractères ont le même code ASCII.

Remarques : On utilise STRING$ pour créer une chaîne qui est constitué d'un caractère répété. <Nombre> indique la longueur de la chaîne restituée. <Code> est le code ASCII du caractère utilisé pour construire la chaîne.

Exemple :
```
a$=STRING$(10,"0") 'Résultat a$="0000000000"
```

Voir aussi : STR$, VAL

8-16-139- STR$ - Conversion en chaîne de caractères

Syntaxe : **STR$**(<Expression>)

Types acceptés : Expression : octet … réel

Description : Cette fonction retourne une chaîne représentant la valeur d'une expression numérique.

Remarques : Quand les nombres sont convertis en texte , un espace en tête est toujours réservé pour le signe de <Expression>. Si <Expression> est positive, la chaîne restituée par Str$ contient un espace en tête et le signe plus est sous-entendu.

Attention : Cette fonction peut renvoyer une valeur suivant une notation de type exposant.

Il est préférable d 'utiliser l'instruction Format$ avec nombre=1

Exemple :
```
a%=10
b$=STR$(a%) 'Résultat b$=" 10"
```

Voir aussi : VAL

8-16-140- SUB – Sous-programme

Syntaxe : **SUB** <Nom>

Description : Ce mot-clé commence un bloc de sous-programme et est aussi utilisé pour définir la fin d'un bloc de sous-programme quand il est précédé de END.

Remarques : Les blocs SUB - END SUB doivent être en dehors d'un bloc PROG – END PROG.

Exemple :
```
SUB Move
...
END SUB
```

Voir aussi : END

8-16-141- SUSPEND – Suspend une tâche

Syntaxe : **SUSPEND** <Nom>

Description : Cette instruction suspend une tâche en cours d'exécution.

Remarques : Cette instruction n'a pas d'effets sur les tâches stoppées. Les mouvements présents dans le buffer de mouvement continuent à s'exécuter.

Exemple :
```
  Wait Inp(Start)
  RUN  Coupe
Begin:
  Wait Inp(Stop)
  SUSPEND Coupe
  Wait Inp(Start)
  CONTINUE Coupe
Goto Begin
```

Voir aussi : RUN, CONTINUE, HALT

8-16-142- TAN - Tangente

Syntaxe : **TAN** (<Expression>)

Types acceptés : Expression : réel

Description : Cette instruction retourne la tangente de <Expression>. <Expression> est un angle exprimé en radians.

Remarques : Cet argument <Expression> doit être une expression numérique valide. La fonction TAN prend un angle et restitue le rapport de deux côtés d'un triangle rectangle. Le rapport est la longueur du côté opposé d'un angle divisée par la longueur du côté adjacent à l'angle.

Exemple : `a!=TAN(PI)`

Voir aussi : SIN, ARCTAN,TAN

8-16-143- TIME – Base de temps

Syntaxe : **TIME**

Description : Cette instruction retourne un entier long qui représente le nombre de millisecondes écoulées depuis la dernière mise sous tension. Cette instruction permet d'effectuer des temporisations non bloquantes. A la mise sous tension Time est égal à 0 puis s'incrémente jusqu'à 2^{31} puis passe en -2^{31} (au bout de 24 jours), s'incrémente jusqu'à 0 et le cycle recommence.

Remarques : Si le SUPERVISOR est sous-tension pendant plus de 24 jours, il faut utiliser TIMER pour éviter l'aléa sur le basculement de la valeur de la tempo de 2^{31} à -2^{31}.

Exemple :
```
Tempo&=Time+5000  'Charge tempo de 5s
WAIT  (INP(Depart)=On) Or (Time>Time&)
'Si l'entrée Départ n'est pas activée au bout de 5 s,
'le programme se poursuit
```

Voir aussi : TIME$, TIMER

8-16-144- TIMER – Base de temps étendue

Syntaxe : **TIMER**

Description : Cette instruction retourne un réel qui représente le nombre de millisecondes écoulées depuis la dernière mise sous tension. Cette instruction permet d'effectuer des temporisations non bloquantes. A la mise sous tension Time est égal à 0 puis s'incrémente toutes les millisecondes de 0.001.

Exemple :
```
Timer!=Timer+5.25        'Charge tempo de 5.25s
WAIT  (INP(Depart)=On) Or (Timer>Time!)
'Si l'entrée Départ n'est pas activée au bout de 5.25s,
'le programme se poursuit
```

Voir aussi : TIME$, TIME, DATE$

8-16-145- TIME$ - Heure courante

Syntaxe : **TIME$**

Description : Cette instruction retourne une chaîne de 8 caractères de la forme hh:mm:ss, où hh sont les heures (00-23), mm sont les minutes (00-59) et ss sont les secondes (00-59).

Voir aussi : TIME, TIMER, DATE$

8-16-146- TX485 – Modifie l'état de la sortie RS485

Syntaxe : **TX485**(<Numéro>)=<Expression>

Types acceptés : Expression : Entier

Description : Cette instruction établie la sortie du port de communication RS485 pendant le nombre de caractères choisi. Si le nombre est 0, la sortie est inhibée.

Remarques : <Numéro> est le numéro utilisé pour ouvrir le port de communication avec l'instruction OPEN. En mode RS485 tous les caractères émis sont également reçus.

Exemple : `TX485(#1)=10`

8-16-147- UCASE$ - Majuscule

Syntaxe : **UCASE$**(<Expression>)

Types acceptés : Expression : chaîne de caractères

Description : Cette fonction retourne une chaîne dans laquelle toutes les lettres de l'argument ont été converties en majuscules.

Remarques : L'argument <Expression> doit être une chaîne de caractères. Seules les lettres minuscules sont converties en majuscules; les autres lettres ne sont pas modifiées.

Exemple :
```
a$="Capteur1"
b$=UCASE$(a$) 'Résultat : b$="CAPTEUR1"
```

Voir aussi : LCASE$

8-16-148- VAL – Conversion Chaîne de caractères / réel

Syntaxe : **VAL**(<Expression>)

Types acceptés : Expression : chaîne de caractères

Description : Cette fonction retourne la valeur numérique de la chaîne <Expression>.

Remarques : L'argument <Expression> est une chaîne de caractères qui peut être interprétée comme une valeur numérique. La fonction VAL arrête de lire la chaîne au premier caractère qui n'est pas reconnu. VAL ignore également les espaces, tabulation et sauts de lignes. La fonction VAL retourne toujours une donnée de type réel.

Exemple :
```
a$="10"
b!=VAL(a$) 'Résultat b!=10
```

Voir aussi : STR$

8-16-149- VERSION – Version de l'operating system

Syntaxe : <Variable>=VERSION

Types acceptés : Variable : chaîne de caractères

Description : Cette fonction retourne la version de l'Operating System sous forme de chaîne de caractères.

8-16-150- VLINE – Affichage d'une ligne verticale

Syntaxe : **VLINE**(X1,Y1,Y2,couleur)

Unité : X1, Y1, Y2 : pixel

Limites : X1 : de 1 à 240

Y1, Y2 : de 1 à 128

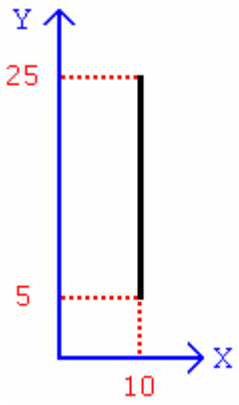
Types acceptés : X1, Y1, Y2 : octet.

Couleur : Bit

Description : Trace une ligne horizontale avec le point de départ en X1, Y1 et d'arrivée en X1, Y2.

Remarques : Couleur permet de modifier la couleur du trait : noir (0) ou blanc (1)

Exemple : VLINE(10,5,25,0)



8-16-151- WAIT EVENT – Attente d'un événement

Syntaxe : **WAIT EVENT** <Nom>

Description : Cette instruction permet au système d'attendre jusqu'à ce qu'un événement soit reçu. L'exécution de cette instruction provoque le basculement vers la tâche suivante.

Remarques : Dans l'instruction WAIT EVENT, les instructions suivantes ne sont pas exécutées tant que l'événement n'est pas reçu. Cette instruction fournit une attente passive d'événement.

Exemple :

```
WHILE Ready=False DO END WHILE        'Attente active
'Ce programme peut être remplacé par :
WAIT EVENT Ready                       'Attente passive
```

Voir aussi : SIGNAL, DIFFUSE, WAIT STATE, DELAY

8-16-152- WAIT KEY – Attente d'une touche

Syntaxe : **WAIT KEY**

Description : Cette fonction attend l'appui d'une touche sur le terminal et enregistre son code dans la variable KEY. L'exécution de cette instruction provoque le basculement vers la tâche suivante.

Remarques : Cette fonction utilise le port de communication #1.Par défaut, le port de communication SERIAL1 sera utilisé. Si un pupitre opérateur est connecté au port SERIAL2, veuillez vous référer à la fonction OPEN pour affecter #1 au port SERIAL2.

Exemple :

```
WAIT  KEY
IF KEY=@F1 THEN GOTO ...
IF KEY=@F2 THEN GOTO ...
...
```

8-16-153- WAIT – Attente d'une condition

Syntaxe : **WAIT** <Condition>

Description : Cette instruction permet au système d'attendre que la condition soit vraie. L'exécution de cette instruction provoque le basculement vers la tâche suivante.

Remarques : L'instruction WAIT, les instructions suivantes ne sont pas exécutées tant que <Condition> se révèle fausse. Cette instruction fournit une attente passive pour une condition.

Exemple :

```
WHILE INP(Capteur)=Off  DO END WHILE  'Attente active
Ce programme peut être remplacé par :
WAIT  INP(Capteur)=On                 'Attente passive
```

Voir aussi : WAIT EVENT, DELAY

8-16-154- WATCHDOG – Chien de garde

Syntaxe 1 : WATCHDOG = ON / OFF

Syntaxe 2 : WATCHDOG

Description : Cette fonction permet à l'utilisateur de lire ou d'écrire l'état du relais de chien de garde.

Remarques : L'état du chien de garde à la mise sous tension est OFF. Il doit donc être mis à ON en début de programme. Le relais est automatiquement décollé quand un axe passe en erreur de poursuite. Cette fonction doit être testée dans une tâche de sécurité. La fonction SECURITY peut modifier également son comportement.

Exemple :

```
WATCHDOG=ON.
WAIT WATCHDOG=OFF
```

8-16-155- WHILE – While…Do…End While

Syntaxe : **WHILE** <Condition> **DO**

{<Instructions>}

END WHILE

Description : Cette instruction permet au système d'exécuter une série d'instructions dans une boucle aussi longtemps que la condition donnée est vraie. L'exécution de l'instruction « END WHILE » provoque le basculement vers la tâche suivante.

Remarques : Dans la structure WHILE ... DO ... END WHILE <Instruction> ne sont pas exécutées si la condition est fausse.

Exemple :

```
a%=0
WHILE a%<=100
    PRINT #1,a%
    a%=a%*2
END WHILE
```

Voir aussi : REPEAT

8-16-156- XOR – Opérateur ou exclusif

Syntaxe : <Expression1> **XOR** <Expression2>

Types acceptés : Expression1, Expression2 : Bit, Octet, Entier

Description : Cette fonction fait un Ou Exclusif entre les expressions.

Remarques : Expression1> et <Expression2> doivent être du même type de donnée. Cette fonction restitue le type de donnée de <Expression1>.

Exemple :

```
IF A% XOR 0FF00h THEN ...
```

Voir aussi : AND, OR, NOT, IF

9- CANopen

9-1- Définition

9-1-1- Introduction

Le bus CAN (Controller Area Network) est apparu au milieu des années 80 pour répondre aux besoins de la transmission de données dans le secteur automobile. Ce type de bus permet d'obtenir des taux de transfert jusqu'à 1Mbit/s.

Les spécifications du CAN définissent 3 couches parmi le modèle OSI : la couche physique, la couche liaison des données et la couche application. La couche physique définit le mode de transmission des données en fonction du support de transmission. La couche liaisons des données représente le noyau du protocole CAN puisque cette couche est responsable de la trame à envoyer, de l'arbitrage, de la détection des erreurs, etc… . La dernière couche est la couche application appelée aussi CAL (CAN Application Layer). Celle-ci est donc une description générale du langage pour les réseaux CAN qui offre de nombreux services de communication.

CANopen est un type de réseau qui est basé sur le système du bus série et de la couche application CAL. CANopen ne propose qu'une partie des services de communication offerte par CAL. Ce sont les avantages nécessaires dont ont besoins les ordinateurs ayant des performances réduites et des capacités de stockage faible.

Le CANopen est, par conséquent, une couche application standardisée par les spécifications du CIA (CAN In Automation) : DS-201…DS-207.

Le gestionnaire du réseau permet une initialisation simplifiée du réseau. Le réseau peut être étendu avec tous les composants que l'utilisateur désire.

Le bus CAN est un bus multi-maître. Contrairement aux autre bus de terrain, ce sont les messages qui sont identifiés et non les modules connectés. Les éléments du réseau sont autorisés à envoyer leurs messages à chaque fois que le bus est libre. Les conflits sur le bus sont résolus par un niveau de priorité donné aux messages. Le bus CAN émet des messages qui sont divisés en 2032 niveaux de priorités. Tous les éléments du réseau ont les mêmes droits et donc cette communication n'est seulement possible que sans bus maître.

Chaque élément décide lui-même lorsqu'il veut envoyer des données. Il est cependant possible de faire envoyer des données par un autre élément. Cette demande est effectuée par la trame distante.

Les spécifications du CANopen (DS-201…DS-207) définissent les caractéristiques techniques et fonctionnelles que nécessitent un appareil individuel pour être associé sur le réseau. Le bus CANopen fait une distinction entre les appareils serveurs et les appareils clients.

9-1-2- La communication CANopen

Le profil de la communication du CANopen permet de spécifier les informations pour l'échange de données en temps réel et des paramètres. Le CANopen utilise des services optimisés suivant les différentes sortes de données.

↳ PDO (Process Data Object)

- ⇨ Echange de donnée en temps réel
- ⇨ Identifiant à haute priorité
- ⇨ Transmission synchrone ou asynchrone
- ⇨ Maximum de 8 octets (un message)
- ⇨ Format prédéfini

- SDO (Service Data Object)
  - ⇨ Accède au dictionnaire des objets d'un appareils
  - ⇨ Identifiant à basse priorité
  - ⇨ Transmission asynchrone
  - ⇨ Données distribuées dans plusieurs télégrammes
  - ⇨ Données adressées par un index

Les caractéristiques diffusées par le CAN sont reçues et évalués par tous les appareils connectés. Chaque service d'un appareil CAN est paramétré par un COBID (Communication OBject Identifier). Le COBID est un identifiant qui caractérise le message. C'est ce paramètre qui permet d'indiquer à un appareil si le message doit être traiter. Pour chaque service (PDO ou SDO), il est nécessaire de spécifier un COBID à l'émission (envoi d'un message) et un COBID à la réception (récupération de massage). Pour le premier SDO serveur, le COBID est fixe est ne peut pas être modifié à distance. De plus, il est calculé à partir du NODE-ID. Le NODE-ID est le paramètre qui caractérise l'appareil et qui permet d'accéder de façon unique à l'appareil.

PDO (Process Data Object)

C'est un échange de donnée arbitré entre deux modules. Les PDO peuvent transférer alternativement des synchronisations ou des événements contrôlés pour réaliser la demande d'envoi des messages. Avec le mode d'événements contrôlés, la charge du bus peut être réduite au minimum. Un appareil peut, donc, réaliser une communication à haute performance avec un faible taux de transfert.

L'échange de donnée avec le PDO utilise les avantages du CAN :

- L'envoi de message peut être déclenché par un événement asynchrone. (événements contrôlés)
- L'envoi de message peut être déclenché sur la réception d'un événement de synchronisation.
- Récupération par une trame à distance.

SDO (Service Data Object)

C'est un échange de donnée point à point. Un appareil vient faire une demande d'accès dans la liste d'objets d'un SDO. Le SDO renvoie une information correspondant au type de requête fait par le demandeur. Chaque SDO peut être client et/ou serveur. Un SDO serveur ne peut pas faire de demande envers un autre SDO par contre lui peut répondre à toute demande d'un SDO client. Contrairement aux PDO, les SDO doivent suivre un protocole de communication particulier. La trame envoyée est composée de 8 octets :

- Domain Protocol (Octet 0) : il définit la commande (Upload, Download,….)
- Index sur 16 bits (Octet 1 et 2) : il définit l'adresse du dictionnaire des objets
- Sub-index sur 8 bits (Octet 3) : il définit l'élément de l'objet sélectionné dans le dictionnaire
- Paramètre (Octet 4 à 7) : Il définit la valeur du paramètre lu ou écrit.

Le gestionnaire de réseau comporte un mode simplifié de démarrage du réseau. La configuration du réseau n'est pas nécessaire dans tous les cas. La configuration par défaut des paramètres est donc parfois suffisante. Si l'utilisateur désire optimiser le réseau CANopen ou augmenter ses fonctionnalités, il peut alors modifier lui-même ces paramètres. Dans les réseaux CANopen, tous les appareils ont les mêmes droits et l'échange des données est directement régulé entre chaques appareils participants.

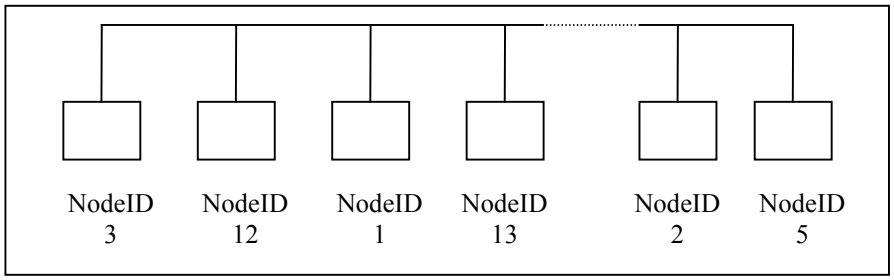
Le profil d'un appareil définit les paramètres nécessaires pour une communication. Le contenu de ce profil est spécifié par le constructeur. Les appareils ayant le même profil sont directement interchangeables. La plupart des paramètres sont décrits par le constructeur. Le profil possède aussi des emplacements vides qui correspondent aux futures extensions de fonctionnalités des constructeurs.

Dans la plupart des bus maître/esclave, l'efficacité du maître détermine le comportement de tout le réseau. De plus, les esclaves ne peuvent pas directement communiquer entre eux. Toutes ces caractéristiques augmentent, donc, le nombre d'erreurs de transmission. CANopen élimine tous ces désavantages. Le comportement temporel peut être spécifié individuellement pour chaque tâche respective des appareils participants. Ainsi, le système entier de communication n'a pas besoin de plus d'efficacité si seulement certains appareils participants nécessitent plus de performance. De plus, une tâche automatique peut être séparée pour chacun des appareils participants. Ainsi, les performances disponibles du contrôleur du réseau peuvent être utilisées de manière optimales et peuvent être augmentées à tout instant par adjonction de nouveaux appareils participants.

Le mapping des variables utilisées lors des échanges de type PDO permet d'utiliser de manière optimale la bande passante actuelle du bus. CANopen détermine les valeurs en défaut de tous les paramètres.

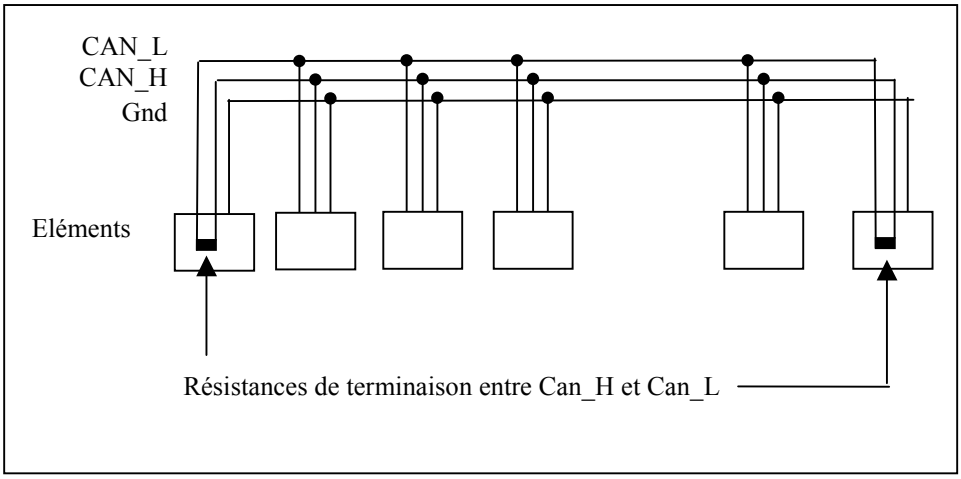
9-1-3- Configuration de réseau

Le réseau CAN OPEN est constitué de différents éléments, tous pouvant être maître et esclaves. Ils sont identifiés dans le réseau par un numéro arbitraire, que l'on appelle le Node-ID. Ce paramètre doit être unique : 2 éléments distincts du réseau ne peuvent pas avoir le même Node-ID. Ce Node-ID est très important, c'est la véritable carte d'identité du périphérique sur le réseau CAN Open.



Exemple de configuration de réseau Can Open

Le principe de câblage est le suivant :



Principe de câblage d'un réseau Can Open

Important : Pensez aux résistances de terminaison à chaque extrémité du réseau. Pour les produits SERAD (SCAN et pupitres), la résistance est validée si le jumper JP1 est mis. Dans le cas contraire, la résistance est dévalidée.

Sur les autres produits, se reporter à la notice du constructeur.

9-1-4- Type de messages envoyés

Il y a deux grandes familles de messages envoyés par la liaison Can Open :

- Les SDO (Service Data Object) transportent des données
- Les PDO (Process Data Object) transmettent des événements.

9-2- Bus CANopen pour Supervisor

9-2-1- Présentation du SERIAL 3

La carte coprocesseur SCAN est implantée dans le SUPERVISOR. Elle possède 3 tables locales de 254 données chacune, pour les trois formats de variables suivants : 8 bits, 16 bits, 32 bits.

Ces tables peuvent être lues et écrites par le SUPERVISOR sans transiter par le réseau CAN Open, par les instructions *CanLocal*.

Les divers paramètres sont inclus dans un élément à 2 dimensions, le **dictionnaire**.

Chaque donnée ou paramètre est défini par une adresse d'index et une adresse de SubIndex.

La bus SCAN peut communiquer avec un autre élément du réseau de plusieurs façons. Elle peut mettre à disposition des données en les écrivants dans sa table locale : n'importe quel élément du réseau peut alors lire et même écrire sur cette table locale.

La bus SCAN peut également lire ou écrire une table locale d'un autre élément. Cette opération se réalise alors par les instructions *CanRemote*.

9-2-2- Caractéristiques

↳ Un serveur SDO par défaut pour le paramétrage de la carte à distance par un supervisor.

↳ Un client SDO pour accéder aux variables et aux paramètres des périphériques CANopen tels que des pupitres, automates et cartes PC.

↳ 8 PDO en émission pour piloter les sorties des modules I/O ou signaler un événement à un SUPERVISOR.

↳ 8 PDO en réception pour recevoir les entrées des modules I/O ou recevoir les événements à un SUPERVISOR.

↳ Un tableau de 254 variables « 8 bits non signé » accessible en lecture et écriture par SDO.

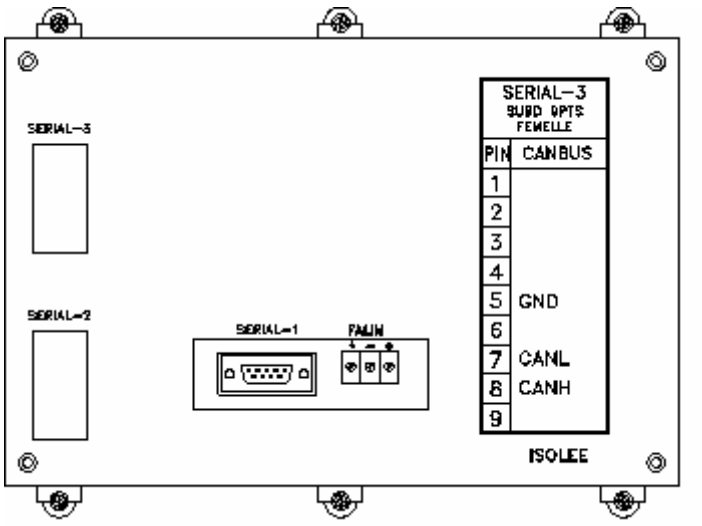
↳ Un tableau de 254 variables « 16 bits non signé » accessible en lecture et écriture par SDO.

↳ Un tableau de 254 variables « 32 bits non signé » accessible en lecture et écriture par SDO.

↳ Des fonctions d'accès direct au bus CAN pour envoyer et recevoir des messages spécifiques tels que les fonctions NMT et DBT.

9-2-3- Raccordement

↳ JP1 Cavalier de validation des résistances (120Ω) de terminaison (à coté du connecteur CANOpen).

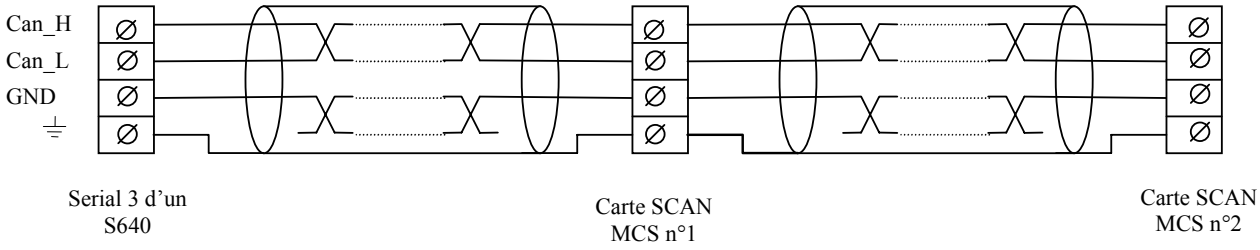


Utiliser un câble 2 paires torsadées à blindage individuel et général (type LiY.CY.CY ou équivalent) :

- une paire pour CAN_L et CAN_H
- une paire pour le GND

Relier les blindages aux bornes

Exemple avec 1 SUPERVISOR et 2 MCS 32 EX en réseau :



Attention

A chaque extrémité du bus prévoir une résistance de terminaison de 120 Ω entre CAN_H et CAN_L (dans le cas d'un Dialog 80, Dialog 640, SUPERVISOR ou carte SCAN, la mise en place du jumper JP1 permet de valider cette résistance.

Par exemple, dans la configuration précédente, on aura :

SUPERVISOR : Mettre le jumper JP1

Carte SCAN n°2 : Enlever le jumper JP1

Carte SCAN n°3 : Mettre le jumper JP1

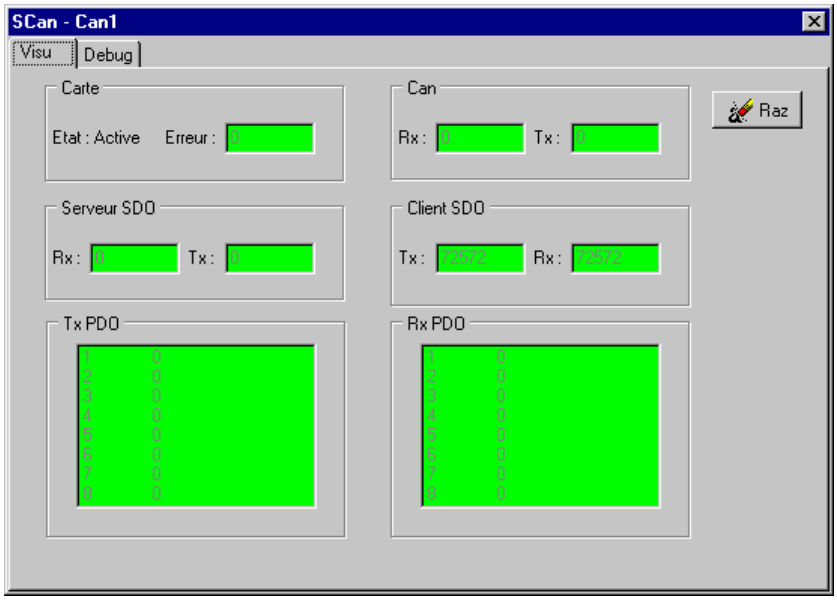
Vitesses maximales de transmission en fonction de la longueur du réseau CAN Open

Vitesse maximale de transmission	Longueur du bus
10K à 125 kBauds	500 m
250 kBauds	250 m
500 kBauds	100 m
800 kBauds	50 m
1 Mbauds	25 m

9-2-4- Test et diagnostic du bus

A partir du logiciel SPL, activez le mode debug et ensuite double-cliquez sur SERIAL3 dans l'onglet configuration.

A) Page VISU



- Carte : visualisation du nombre d'erreurs de communication dans la carte et son état

  3 états possibles :

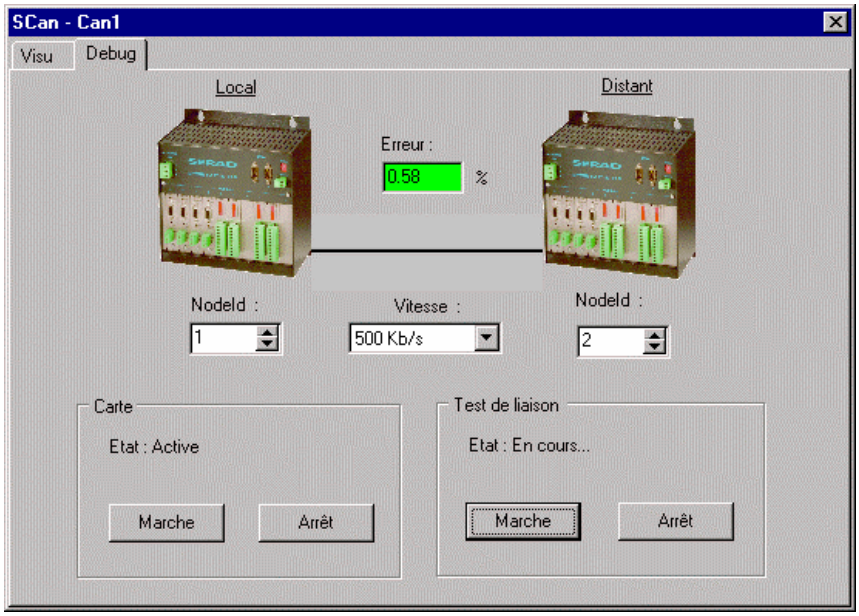
  1- mode arrêté : bus SCAN attend une instruction StartCan

  2- Mode démarré : bus CAN est prête à communiquer

  3- Mode actif : bus CAN est en cours de communication

- CAN : visualisation du nombre d'émissions et de réceptions en protocole libre
- Serveur SDO : affichage du nombre de requêtes reçues et le nombre de réponses correctes envoyées.
- Client SDO : affichage du nombre de requêtes envoyées et le nombre de réponses correctes reçues.
- Tx PDO : affichage du nombre de PDO envoyés (sous-total par numéro de PDO)
- Rx PDO : affichage du nombre de PDO reçus (sous-total par numéro de PDO)
- RAZ : cliquer ici pour remettre à zéro tous les compteurs de cette page

B) Page DEBUG :



Cette page permet de valider très facilement le communication entre 2 périfériques à l'intérieur d'un réseau CAN Open.

La procédure est la suivante :

- Sur la carte locale, arrêter les tâches
- Accéder au mode debug
- Saisir son Node-Id, la vitesse de transmission, et le Node-Id de la carte distante.
- Pour la carte distante, deux cas sont distincts.

1- Il n'y a pas de tâche dans le SUPERVISOR

Il convient d'en créer une, afin d'activer le SERIAL3, en démarrage automatique, appelée INIT par exemple :

```
Prog
Delay 2000
StartCan (NomCarte, Vitesse, Node-Id)
Halt INIT
End Prog
```

2- Il y a déjà des tâches dans le SUPERVISOR

Dans la tâche automatique, ajouter au début :

```
Prog
Delay 2000
StartCan (NomCarte, Vitesse, Node-Id)
Halt INIT
…
End Prog
```

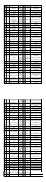
Attention : s'assurer que le projet ne contient qu'une seule tâche automatique (sinon passer les autres en manuel)

- Dans les deux cas, compiler et transférer le programme
- Valider le test de liaison en cliquant sur « marche » dans test de la liaision. Le pourcentage d'erreur vous dira très rapidement si le bus est opérationnel d'un point de vue hardware pour ces deux périphériques.

NB : le pourcentage est calculé sur les valeurs affichées dans la page VISU. Il est donc directement lié à la remise à zéro de cette page, qu'il est conseillé de faire pour réinitialiser cet indicateur.

9-2-5- Dictionnaire

Le dictionnaire contient les différents paramètres et variables de la carte. Ils sont directement accessibles par le SUPERVISOR à l'aide des fonctions CANSETUP. Les tableaux des variables sont accessibles par les fonctions CANLOCAL. Pour accéder aux paramètres des autres périphériques CANopen, il faut utiliser les fonctions CANREMOTE.



7280	from 1 to FEh	Read 16 bits variables	16 bits non signés	ro	aucune	
8180	from 1 to FEh	Write 32 bits variable	32 bits signé	wo	aucune	
8200	from 1 to FEh	Write 8 bits variable	8 bits non signé	wo	aucune	
8280	from 1 to FEh	Write 16 bits variable	16 bits non signés	wo	aucune	

9-3- Liste des instructions

9-3-1- Liste des instructions CANopen

A) Lecture et écriture du dictionnaire

CANSETUP#	Lecture ou écriture d'un paramètre (octet)
CANSETUP%	Lecture ou écriture d'un paramètre (word)
CANSETUP&	Lecture ou écriture d'un paramètre (entier long)

B) Modification de variables locales

CANLOCAL#	Lecture ou écriture d'une variable local (octet)
CANLOCAL%	Lecture ou écriture d'une variable local (word)
CANLOCAL&	Lecture ou écriture d'une variable local (entier long)

C) Modification de variables distantes

CANREMOTE#	Lecture ou écriture d'une variable distante (octet)
CANREMOTE%	Lecture ou écriture d'une variable distante (word)
CANREMOTE&	Lecture ou écriture d'une variable distante (entier long)

D) Instructions en mode PDO

CAN	Lecture ou écriture des données
CANEVENT	Test de l'arrivée d'un message
PDOEVENT	Test l'arrivée d'un PDO
PDO	Lecture ou écriture des données par un PDO
SETUPCAN	Paramétrage d'un message

E) Instructions de contrôle

CANERROR	Détection des erreurs
CANERRORCOUNTER	Contrôle et efface les erreurs de la communication
STARTCAN	Démarrage du module CANopen
STOPCAN	Arrêt du module CANopen

F) Instructions en mode SDO

SDOEVENT	Test si écriture par SDO
SDOINDEX	Lecture de l'index de l'objet du dictionnaire
SDOSUBINDEX	Lecture du sub-index de l'objet du dictionnaire

9-3-2- CAN – Lecture et écriture d'un message

Syntaxe 1 :	**CAN**(<Carte>, <Donnée>)
Syntaxe 2 :	<Variable> = **CAN**(<Carte>)
Types acceptés :	<Donnée>, <Variable> : chaîne de caractères
Description :	Cette fonction permet de lire ou d'envoyer un message.
Remarques :	<Carte> doit être une carte CANopen. Il est nécessaire de paramétrer le COBID de réception pour pouvoir recevoir un message.

9-3-3- CANERROR – Détection des erreurs

Syntaxe : <Variable> = **CANERROR**(<Carte>)

Types acceptés : <Variable> : booléen

Description : Cette fonction permet de détecter si une erreur s'est produite.

Remarques : <Carte> doit être une carte CANopen.

9-3-4- CANERRORCOUNTER - Contrôle et efface les erreurs de la communication

Syntaxe 1 : <Variable> = **CANERRORCOUNTER** (<Carte>)

Syntaxe 2 : **CANERRORCOUNTER**(<Carte>) = 0

Limites : <Variable> : de 0000h à FFFFh

Types acceptés : <Variable> : entier

Description : La syntaxe 1 permet de connaître le nombre d'erreur qui se sont produites depuis la dernière initialisation du compteur. La deuxième syntaxe permet d'initialiser le compteur d'erreur.

Remarques : <Carte> doit être une carte CANopen.

9-3-5- CANEVENT – Test l'arrivée d'un message

Syntaxe : <Variable> = **CANEVENT** (<Carte>)

Types acceptés : <Variable> : booléen

Description : Cette fonction permet de savoir si un message a été réceptionné.

Remarques : <Carte> doit être une carte CANopen. Il est nécessaire de paramétrer le COBID de réception pour pouvoir recevoir un message.

9-3-6- CANLOCAL - Lecture ou écriture d'une variable local

Syntaxe 1 : **CANLOCAL#** (<Carte>, <Index>, <Expression>)

Syntaxe 2 : <Variable> = **CANLOCAL#** (<Carte>, <Index>)

Syntaxe 3 : **CANLOCAL%** (<Carte>, <Index>, <Expression>)

Syntaxe 4 : <Variable> = **CANLOCAL%** (<Carte>, <Index>)

Syntaxe 5 : **CANLOCAL&** (<Carte>, <Index>, <Expression>)

Syntaxe 6 : <Variable> = **CANLOCAL&** (<Carte>, <Index>)

Limites : <Index> : de 0000h à FFFFh

Syntaxe 1 et 2 : <Variable>, <Expression> : de 00h à FFh

Syntaxe 3 et 4 : <Variable>, <Expression> : de 0000h à FFFFh

Syntaxe 5 et 6 : <Variable>, <Expression> : +/- 7FFFFFFFh

Types acceptés : Syntaxe 1 et 2 : <Expression>, <Variable> : Octet

Syntaxe 3 et 4 : <Expression>, <Variable> : Entier

Syntaxe 5 et 6 : <Expression>, <Variable> : Entier long

Description : Cette fonction permet de lire ou d'écrire une variable locale du dictionnaire de la carte CANopen de leSUPERVISOR. Les syntaxes 1 et 2 réalisent un accès sur le tableau des variables de type 8 bits non signé. Les syntaxes 3 et 4 réalisent un accès sur le tableau des variables de type 16 bits non signé. Les syntaxes 5 et 6 réalisent un accès sur le tableau des variables de type 32 bits signé.

Remarques : <Carte> doit être une carte CANopen. <Index> doit se référer à une variable locale du dictionnaire.

9-3-7- CANSETUP - Lecture ou écriture d'un paramètre

Syntaxe 1 : **CANSETUP#** (<Carte>, <Index>, <Sub-Index>, <Expression>)

Syntaxe 2 : <Variable> = **CANSETUP#** (<Carte>, <Index>, <Sub-Index>)

Syntaxe 3 : **CANSETUP%** (<Carte>, <Index>, <Sub-Index>, <Expression>)

Syntaxe 4 : <Variable> = **CANSETUP%** (<Carte>, <Index>, <Sub-Index>)

Syntaxe 5 : **CANSETUP&** (<Carte>, <Index>, <Sub-Index>, <Expression>)

Syntaxe 6 : <Variable> = **CANSETUP&** (<Carte>, <Index>, <Sub-Index>)

Limites : <Index> : de 0000h à FFFFh

<Sub-index> : de 00h à FFh

Syntaxe 1 et 2 : <Variable>, <Expression> : de 00h à FFh

Syntaxe 3 et 4 : <Variable>, <Expression> : de 0000h à FFFFh

Syntaxe 5 et 6 : <Variable>, <Expression> : +/- 7FFFFFFFh

Types acceptés : Syntaxe 1 et 2 : <Expression>, <Variable> : Octet

Syntaxe 3 et 4 : <Expression>, <Variable> : Entier

Syntaxe 5 et 6 : <Expression>, <Variable> : Entier long

Description : Cette fonction permet de lire ou d'écrire une donnée dans le dictionnaire du SUPERVISOR.

Remarques : <Carte> doit être une carte CANopen. <Index> et <Sub-Index> doivent se référer à un élément du dictionnaire.

9-3-8- CANREMOTE - Lecture ou écriture d'une variable distante

Syntaxe 1 : **CANREMOTE#** (<Carte>, <Index>, <Sub-Index>, <Expression>)

Syntaxe 2 : <Variable> = **CANREMOTE#** (<Carte>, <Index>, <Sub-Index>)

Syntaxe 3 : **CANREMOTE%** (<Carte>, <Index>, <Sub-Index>, <Expression>)

Syntaxe 4 : <Variable> = **CANREMOTE%** (<Carte>, <Index>, <Sub-Index>)

Syntaxe 5 : **CANREMOTE&** (<Carte>, <Index>, <Sub-Index>, <Expression>)

Syntaxe 6 : <Variable> = **CANREMOTE&** (<Carte>, <Index>, <Sub-Index>)

Limites : <Index> : de 0000h à FFFFh

<Sub-index> : de 00h à FFh

Syntaxe 1 et 2 : <Variable>, <Expression> : de 00h à FFh

Syntaxe 3 et 4 : <Variable>, <Expression> : de 0000h à FFFFh

Syntaxe 5 et 6 : <Variable>, <Expression> : +/- 7FFFFFFFh

Types acceptés : Syntaxe 1 et 2 : <Expression>, <Variable> : Octet

Syntaxe 3 et 4 : <Expression>, <Variable> : Entier

Syntaxe 5 et 6 : <Expression>, <Variable> : Entier long

Description : Cette fonction permet de lire ou d'écrire une variable à distance dans le dictionnaire du SUPERVISOR.

Remarques : <Carte> doit être une carte CANopen. <Index> et <Sub-Index> doivent se référer à un élément du dictionnaire distant. Il est nécessaire de préciser les paramètres client et serveur SDO de la carte avant de pouvoir envoyer une lecture ou une écriture de variable à distance.

9-3-9- PDOEVENT – Test l'arrivée d'un PDO

Syntaxe : <Variable> = **PDOEVENT** (<Carte>, <N°PDO>)

Limites : <N°PDO> : de 01h à 08h

Types acceptés : <Variable>, <N°PDO> : Octet

Description : Cette fonction permet de connaître si une demande d'un PDO est effective.

Remarques : <Carte> doit être une carte CANopen. Il est nécessaire de préciser les paramètres de transmission du PDO pour pouvoir recevoir un PDO.

9-3-10- PDO - Lecture ou écriture des données par un PDO

Syntaxe 1 : **PDO** (<Carte>, <N°PDO>, <Donnée>)

Syntaxe 2 : <Variable> = **PDO** (<Carte>, <N°PDO>)

Limites : <N°PDO> : de 01h à 08h

<Donnée>, <Variable> : chaîne de caractères

Types acceptés : <N°PDO> : Octet

<Donnée>, <Variable> : chaîne de caractères

Description : Cette fonction permet de lire ou d'envoyer un PDO.

Remarques : <Carte> doit être une carte CANopen. Il est nécessaire de préciser les paramètres de transmission du PDO pour pouvoir recevoir un PDO.

9-3-11- SDOEVENT – Evènement SDO

Syntaxe : <Variable bit> = SDOEvent(<Carte Can>)

Description : Cette instruction permet de savoir si une écriture par SDO a été effectuée dans la carte CAN. La lecture du bit provoque son effacement.

9-3-12- SDOINDEX – Index SDO

Syntaxe : <Variable entière> = SDOIndex(<Carte Can>)

Description : Cette instruction permet de connaître l'index de l'objet du dictionnaire qui a été écrit.

9-3-13- SDOSUBINDEX – Sous-index SDO

Syntaxe : <Variable octet> = SDOSubIndex(<Carte Can>)

Description : Cette instruction permet de connaître le sub-index de l'objet du dictionnaire qui a été écrit.

9-3-14- SETUPCAN - Paramétrage d'un message

Syntaxe : **SETUPCAN** (<Carte>, <TX COBID>, <RX COBID>)

Types acceptés : <TX COBID>, <RX COBID> : entier long

Description : Cette fonction permet de configurer les COBID de réception et de transmission avant l'envoi d'un message.

Remarques : <Carte> doit être une carte CANopen.

9-3-15- STARTCAN – Démarrage d'une carte CANopen

Syntaxe : **STARTCAN** (<Carte>, <Node ID>, <Fréq>)

Limites : <Node ID> : de 01h à FFh

<Fréq> : de 1 à 8

Types acceptés : <Node ID>, <Fréq> : octet

Description : Cette fonction permet de relier la carte CANopen au réseau.

Remarques : <Carte> doit être une carte CANopen.

9-3-16- STOPCAN – Arrête une carte CANopen

Syntaxe : **STOPCAN** (<Carte>)

Description : Cette fonction permet d'enlever la carte correspondante du réseau CANopen.

Remarques : <Carte> doit être une carte CANopen.

9-4- Exemples

9-4-1- Liaison CANopen entre deux SUPERVISOR

Chaque SUPERVISOR contient un SERIAL 3. Deux exemples vont être décrits:

- Les 2 SUPERVISOR mettent à disposition des variables dans leur table locale, et vont lire la table locale de l'autre carte : transmission de SDO uniquement
- Ajout de notion d'événement : communication de PDO

A) Transfert de variables

- Initialisation du bus CANOPEN

La première chose à faire, une fois les 2 secondes écoulées pour l'initialisation hardware de port SERIAL 3, est d'exécuter l'instruction STARTCAN.

```
Delay 2000
' Démarrage de la liaison CANOpen avec :
' Vitesse : 500Khz
' Node-Id : 1
 StartCan(Can1,5,1)
```

Nous utilisons une sub-routine pour l'initialisation des SDO. En effet, le seul paramètre pour cette sub-routine est le numéro du Node-ID avec lequel la carte va communiquer.

```
' Initialisation client SDO pour communiquer avec le SUPERVISOR sur le Node-ID 2

 ID#=2

  Call SetupSDO

Sub SetupSDO

' Fonction : Saisie des paramètres SDO

'

' Paramètres : ID# : Node-Id du périphérique déporté

'

  CanSetup&(Can1,1280h,1,600h+ID#)

  CanSetup&(Can1,1280h,2,580h+ID#)

End Sub
```

- Ecriture de données sur la table locale

  Cette opération permet de mettre à disposition des données pour n'importe quel périphérique du réseau Can Open.

  Elle se réalise grâce aux fonctions CanLocal.

```
' Ecriture d'une variable 8 bits

CanLocal#(Nom_SCAN, index, variable)

' Ecriture d'une variable 16 bits

CanLocal%(Nom_SCAN, index, variable)

' Ecriture d'une variable 32 bits

CanLocal&(Nom_SCAN, index, variable)
```

- Lecture de données sur la table locale

  Cette opération rapatrie les données de la table locale vers les variables de le SUPERVISOR elle-même.

```
' Lecture d'une variable 8 bits

Variable=CanLocal#(Nom_SCAN, index)
```

‘ Lecture d’une variable 16 bits

Variable=CanLocal%(Nom_SCAN, index)

‘ Lecture d’une variable 32 bits

Variable=CanLocal&(Nom_SCAN, index)

- Lecture de données déportées sur la table locale d’un autre périphérique CanOpen

  Cette opération se fait de manière transparente par les fonctions CanRemote. L’initialisation du périphérique « cible » doit être faite au préalable à partir des fonctions CanSetup&, comme indiqué précédemment.

  ‘ Lecture d’une variable 8 bits

  Variable=CanRemotel#(Nom_SCAN, index)

  ‘ Lecture d’une variable 16 bits

  Variable=CanRemote%(Nom_SCAN, index)

  ‘ Lecture d’une variable 32 bits

  Variable=CanRemote&(Nom_SCAN, index)

  Cette opération doit être accompagnée d’un test « CanError » pour s’assurer qu’elle a bien été faite.

- Ecriture de données déportées sur la table locale d’un autre périphérique CanOpen

  Cette opération se fait de manière transparente par les fonctions CanRemote. L’initialisation du périphérique « cible » doit être faite au préalable à partir des fonctions CanSetup&, comme indiqué précédemment.

  ‘ Ecriture d’une variable 8 bits

  CanRemote#(Nom_SCAN, index, variable)

  ‘ Ecriture d’une variable 16 bits

  CanRemote%(Nom_SCAN, index, variable)

  ‘ Ecriture d’une variable 32 bits

  CanRemote&(Nom_SCAN, index, variable)

  NB : Pour les instructions CanRemote, il est judicieux d’insérer des délais très courts (2 ms par exemple) entre les instructions si l’ont fait de la copie par blocs, afin de ne pas surcharger le bus Can Open

  Lecture par CanRemote : il est conseillé de lire dans une variable temporaire, de tester si il y a eu une erreur CAN et dans le cas où il n’y en a pas eu affecter la variable finale.

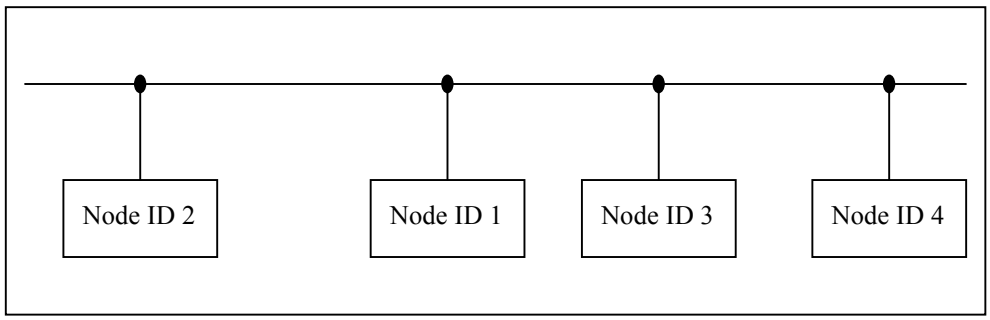
Un exemple de programme est présenté sur le CD Rom. Il est conseillé de s'en inspirer pour programmer de telles fonctions. Se reporter au fichier lisez_moi.txt du CD Rom pour trouver ces fichiers exemples.

B) Communication d'événements (PDO)

La communication par PDO est différente dans l'esprit par rapport aux SDO précédemment décrits.

Là où les SDO envoient des données à un périphérique bien défini, le PDO envoie un octet sur le réseau, et tous les périphériques ayant été paramétrés en adéquation avec cette émission reçoivent cette information.

Il existe différents PDO. Pour l'exemple présenté ici, nous n'utiliserons que le premier PDO. Les autres fonctionnent exactement de la même façon.



Prenons l'exemple du réseau ci-dessus, avec différents périphériques. Chaque élément de ce réseau possède 2 paramètres importants relatifs à un PDO, un pour l'émission et un pour la réception. Ces paramètres sont des COB-ID.

Ce sont ces paramètres qui définissent à qui sont destinés les messages PDO envoyés sur le réseau par n'importe quel périphérique.

Pour une carte SCAN, les paramètres de transmission par défaut du premier PDO sont :

200h + Node-ID en réception

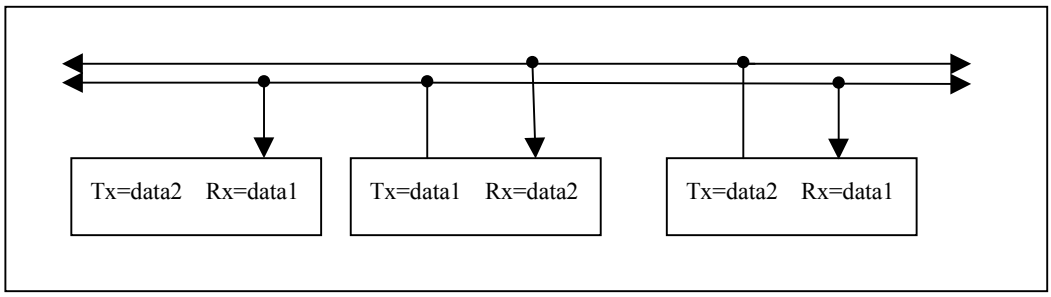
180h + Node-ID en émission

Prenons par exemple le cas du périphérique 2, valeur de Node-ID 2.

Ses paramètres par défaut seront 202h en réception et 182h en émission.

Cela signifie que, en gardant ces paramètres par défaut, lorsque ce périphérique émettra un PDO, celui-ci aura comme COB-ID 182h. Alors tous les périphériques dont le COB-ID de réception du PDO 1 seront accordés avec cette valeur 182h, recevront ce PDO.

De la même façon, le périphérique 2 recevra tous les PDO émis par les périphériques dont le COB-ID de PDO en émission sera égal à 202h.



On peut symboliser ce fonctionnement par le schéma suivant :

Donc, plusieurs paramétrages sont possibles pour l'envoi d'un PDO, la seule condition est que le COB-ID de transmission de l'émetteur soit égal au COB-ID de réception du récepteur.

Par exemple, entre 2 SUPERVISOR, nous pouvons choisir la configuration suivante:

'SUPERVISOR 1 : Emission imposée à 202h, valeur de réception

'par défaut

CanSetup&(Can1,1800h,1,202h)

'SUPERVISOR 2 : Emission imposée à 201h, valeur de réception

'par défaut

CanSetup&(Can1,1800h,1,201h)

Ainsi, on obtient le tableau suivant :

	COB-ID transmission	COB-ID réception
SUPERVISOR 2	201h	202h
SUPERVISOR 1	202h	201h

Et on voit bien que, lorsque le SUPERVISOR 1 enverra un PDO, celui-ci sera lu par la SUPERVISOR 2, et réciproquement.

a) Comment envoyer un PDO

Il suffit tout simplement d'appeler la fonction :

PDO(Can1,numéro,contenu)

'Can1 : Nom du module Scan

'numéro : numéro du PDO, de 01h à 08h

'contenu : donnée, chaîne de caractères

Attention : tester la fonction « CanError » pour s'assurer que tout se passe correctement.

b) Comment savoir si un PDO est arrivé

On utilise la fonction :

Variable=PDOEVENT(Can1,numéro)

'Can1 : Nom du module Scan

'numéro : numéro du PDO, de 01h à 08h

'Variable : Bit

Cette fonction passe à 1 lorsque un PDO est reconnu, et repasse à zéro dès qu'elle est lue.

c) Comment lire un PDO

On utilise la fonction :

Contenu=PDO(Can1,numéro)

'Can1 : Nom du module Scan

'numéro : numéro du PDO, de 01h à 08h

'contenu : donnée, chaîne de caractères

9-4-2- Liaison CANopen entre un SUPERVISOR et un module d'entrées/sorties

Divers modules d'entrées/sorties par Can Open sont disponibles dans le commerce. Par exemple, les drivers pour les modules Wago, Selectron, Weidmuller, sont disponibles sur le CD Rom.

Le paramétrage de la communication entre un SUPERVISOR et un module I/O consiste à attribuer un NodeID a chacun. Le NodeID d'un module I/O est généralement paramétré par des dipswitchs. Une communication par SDO et PDO est alors possible.

Les COBID par défaut des serveurs SDO sont 600h+NodeID en réception et 580h+NodeID en émission. Les COBID par défaut du premier PDO sont 200h+NodeID pour la réception et 180h+NodeID pour l'émission. On paramètre donc les clients respectifs en conséquence.

↳ Initialisation du SUPERVISOR

```
'Démarrage de la carte à 500KBits/s sur le nœud 1
StartCan(Can1,1,5)
'COBID ClientSDO Rx SUPERVISOR= COBID ServerSDO Tx I/O
CanSetup&(Can1,1280h,1,582h)
'COBID ClientSDO Tx SUPERVISOR= COBID ServerSDO Rx I/O
CanSetup&(Can1,1280h,2,602h)
'COBID TxPDO SUPERVISOR = COBID RxPDO I/O
CanSetup&(Can1,1800h,1,202h)
'COBID RxPDO SUPERVISOR = COBID TxPDO I/O
```

```
CanSetup&(Can1,1400h,1,182h)
```

Les modules I/O nécessite l'envoi du Message « NMT Start » pour qu'ils deviennent opérationnels. L'envoie de ce message utilise les fonctions CAN générales :

```
SetupCan(Can1, 0, 0)  ' Utiliser le COBID 0 pour accédér au serveur NMT
Nmt$=Chr$(1)+Chr$(2)  ' Le NodeID du module est 2.
Can(Can1,Nmt$)
```

La lecture et l'écriture des I/O par SDO peut se faire de la façon suivante :

```
A#=CanRemote#(Can1,6000h,1)        'Lecture des entrées 1 à 8
A#=CanRemote#(Can1,6000h,2)        'Lecture des entrées 9 à 16
CanRemote#(Can1,6200h,1,01000100b) 'Mise à 1 des sorties 3 et 7
```

Il est possible de recevoir l'état des entrées et de modifier l'état des sorties par PDO. Le contenu des PDO dépend du mapping défini par la construction.

```
Wait PDOEvent(Can1,1)        'Attente d'un changement sur les entrées
E$=PDO(Can1,1)               'Lecture du PDO
E1#=ASC(MID$(E$,1,1))        'Lecture du premier bloc d'entrée
If E1#.3 Then ...            'Utilisation de la 3éme entrée
S$=Chr$(00010011b)           'Ecriture des sorties 1, 2 et 5
PDO(Can1,1,S$)               'Envoi du PDO
If CanError(Can1) Then Goto ErrorDetected 'Test de l'erreur
```

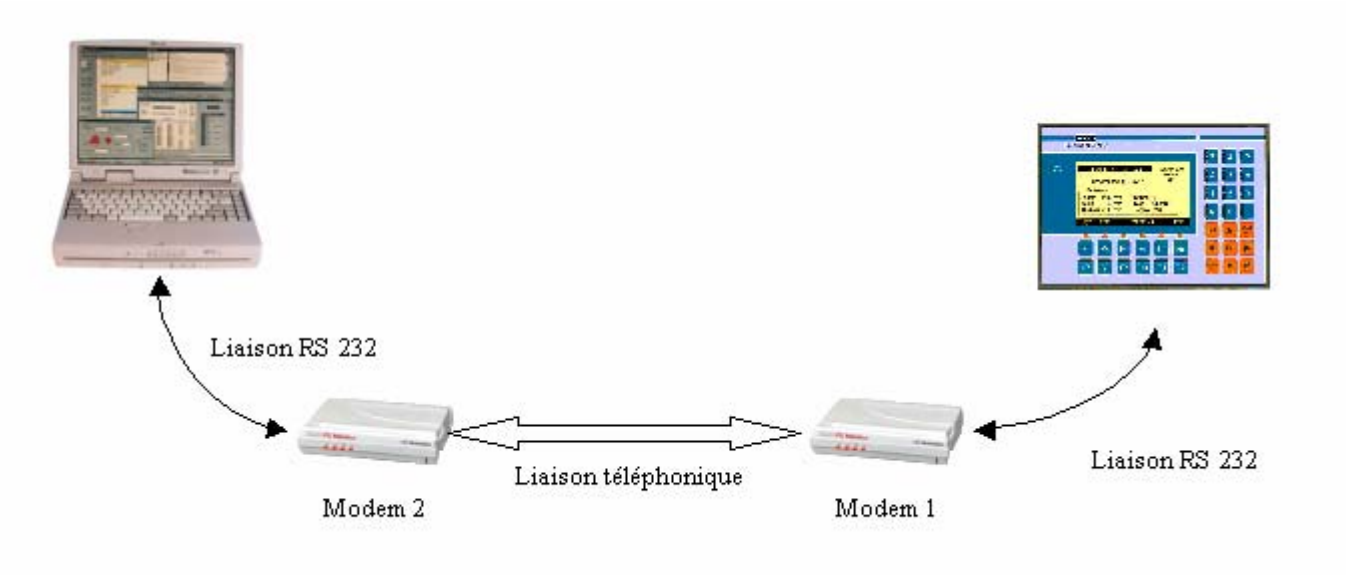
10- TELEMAINTENANCE

10-1- Raccordement

La télémaintenance permet par une liaison téléphonique de contrôler à distance une SUPERVISOR à l'aide du logiciel SPL. La télémaintenance se compose d'un numéroteur téléphonique intégré au logiciel SPL, de deux modems reliés entre eux par une ligne téléphonique.

- **Architecture**

Les différents éléments sont connectés de la façon suivante :



- **Liaison RS 232 entre le modem 1 et la SUPERVISOR**

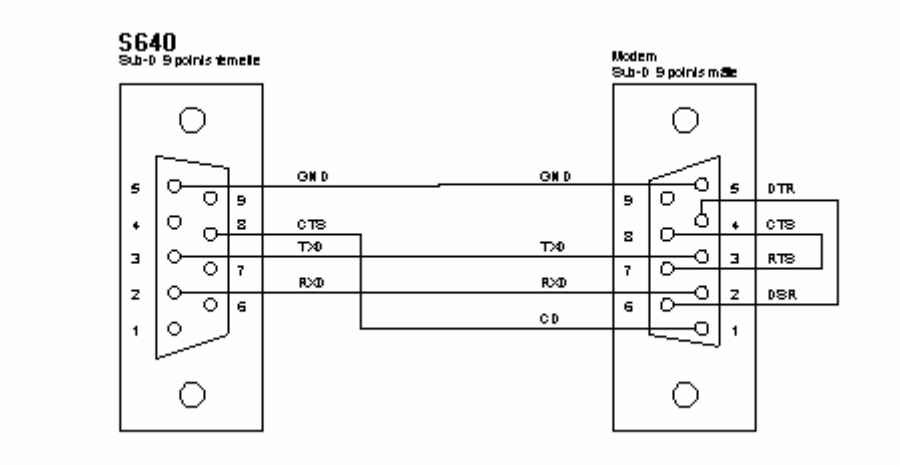
Brochage des connecteurs SUBD 9 points :

SUPERVISOR	**Modem**
	CD
RXD	RXD
TXD	TXD
	DTR
GND	GND
	DSR
	RTS

CTS CTS

Prévoir un câble blindé avec blindage relié aux deux extrémités.

Câblage :



- **Liaison RS 232 entre le modem 2 et le PC**

  La liaison entre le modem et le PC est réalisée directement par le câble fourni avec le modem.

10-2- Etablissement de la liaison

1. Paramétrage du modem 1 relié à la SUPERVISOR

Le paramétrage du modem relié à la SUPERVISOR s'effectue en reliant ce dernier à un PC. Pour ce faire on utilise un logiciel terminal pour envoyer les commandes au modem.

Ce paramétrage a pour but d'effectuer les opérations suivantes :

- Initialiser le modem
- Définir le nombre de sonnerie avant le décrochage du modem pour permettre l'établissement automatique de la liaison.
- Supprimer les contrôles de flux matériel et logiciel.
- Stocker cette configuration dans la mémoire non volatile du modem
- Définir ces paramètres en mémoire non volatile comme paramètres à utiliser à la mise sous tension.

Exemple :

Paramétrage d'un modem de type « 3Com Us Robotics Sportster » :

- Commande : AT&F0

Signification : Chargement des paramètres d'usine.

- Commande : ATS0=3

Signification : Décrochage automatique au bout de 3 sonneries.

- Commande : AT&H0

Signification : Désactive le contrôle de flux en émission

- Commande : AT&I0

Signification : Désactive le contrôle de flux en réception

- Commande : AT&W0

Signification : Stockage des paramètres courant dans la mémoire non volatile N°0

- Commande : ATY0

Signification : Définir les paramètres en mémoire non volatile N°0 comme paramètres à utiliser à la mise sous tension.

Lorsque ces commandes sont prise en compte par le modem celui-ci répond « OK » .

Paramétrage d'un modem de type « Westermo TD31 ou TD32 » :

- Commande : AT&F

Signification : Chargement des paramètres d'usine.

- Commande : ATS0=3

Signification : Décrochage automatique au bout de 3 sonneries.

- Commande : AT&C1

Signification : Active DCD à la connection

- Commande : AT&K0

Signification : Désactive le contrôle de flux

- Commande : AT&W0

Signification : Stockage des paramètres courant dans la mémoire non volatile N°0

- Commande : AT&Y0

Signification : Définir les paramètres en mémoire non volatile N°0 comme paramètres à utiliser à la mise sous tension.

Lorsque ces commandes sont prise en compte par le modem celui-ci répond « OK » .

2. Paramétrage du modem 2 relié au PC

Le paramétrage du modem relié à la PC s'effectue à la rubrique « Modem » du fichier SPL.INI se trouvant dans le répertoire principal de Windows (C :\Windows ou C :\Winnt par exemple).

Ce paramétrage a pour but d'effectuer les opérations suivantes :

- Initialiser le modem
- Supprimer la prise en compte des signaux DSR et DTR pour éviter un raccrochage automatique en cas de fermeture du port de communication.
- Définir la méthode d'appel et de raccrochage du modem.
- Définir les messages renvoyés par le modem.

Exemple :

Paramétrage d'un modem de type « 3Com Us Robotics Sportster » :

- Paramètre : Init1

Valeur : ATZ

Signification : Chargement des paramètres d'usine.

- Paramètre : Init1TimeOut

Valeur : 5

Signification : Délai en 1/10 de seconde d'attente maxi de la réponse du modem.

- Paramètre : Init2

Valeur : AT&D0&S0

Signification : Suppression de la prise en compte de DTR et DSR

- Paramètre : Init2TimeOut

Valeur : 5

Signification : Délai en 1/10 de seconde d'attente maxi de la réponse du modem.

- Paramètre : Dial

Valeur : ATDT pour numérotation vocale. ATDP pour numérotation impulsionnelle

Signification : Définition de la méthode d'appel.

- Paramètre : DialTimeOut

Valeur : 600

Signification : Délai en 1/10 de seconde d'attente maxi avant la connexion.

- Paramètre : Ok

Valeur : OK

Signification : Réponse du modem si la commande est exécutée correctement.

- Paramètre : Connect

Valeur : CONNECT

Signification : Définir le message renvoyé par le modem à la connexion.

- Paramètre : Busy

Valeur : BUSY

Signification : Définir le message renvoyé par le modem si la ligne est occupée.

- Paramètre : Hangup

Valeur : ATH

Signification : Définition de la méthode de raccrochage

- Paramètre : HangupOk

Valeur : NO CARRIER

Signification : Définir le message renvoyé par le modem lorsqu'il raccroche la ligne

- Paramètre : CommandTimeOut

Valeur : 20

Signification : Délai en 1/10 de seconde d'attente maxi avant le passage en mode commande.

- Paramètre : HangupTimeOut

Valeur : 20

Signification : Délai en 1/10 de seconde d'attente maxi avant le raccrochage.

- Ces paramètres sont automatiquement fixés aux valeurs par défauts indiquées lors de la première utilisation.

Paramétrage d'un modem de type « Westermo TD31 ou TD32» :

- Paramètre : Init1

Valeur : ATZ

Signification : Chargement des paramètres d'usine.

- Paramètre : Init1TimeOut

Valeur : 20

Signification : Délai en 1/10 de seconde d'attente maxi de la réponse du modem.

- Paramètre : Init2

Valeur : AT&F&K0

Signification : Suppression de la prise en compte de DTR et DSR

- Paramètre : Init2TimeOut

Valeur : 20

Signification : Délai en 1/10 de seconde d'attente maxi de la réponse du modem.

- Paramètre : Dial

Valeur : ATDT pour numérotation vocale. ATDP pour numérotation impulsionnelle

Signification : Définition de la méthode d'appel.

- Paramètre : DialTimeOut

Valeur : 600

Signification : Délai en 1/10 de seconde d'attente maxi avant la connexion.

- Paramètre : Ok

Valeur : OK

Signification : Réponse du modem si la commande est exécutée correctement.

- Paramètre : Connect

Valeur : CONNECT

Signification : Définir le message renvoyé par le modem à la connexion.

- Paramètre : Busy

Valeur : BUSY

Signification : Définir le message renvoyé par le modem si la ligne est occupée.

- Paramètre : Hangup

Valeur : ATH

Signification : Définition de la méthode de raccrochage

- Paramètre : HangupOk

Valeur : NO CARRIER

Signification : Définir le message renvoyé par le modem lorsqu'il raccroche la ligne

- Paramètre : CommandTimeOut

Valeur : 20

Signification : Délai en 1/10 de seconde d'attente maxi avant le passage en mode commande.

- Paramètre : HangupTimeOut

Valeur : 20

Signification : Délai en 1/10 de seconde d'attente maxi avant le raccrochage.

Le numéroteur téléphonique suppose que le modem est paramètré pour recevoir un écho aux commandes envoyées et un message texte en réponse. Dans le cas contraire la communication serait impossible. Il est possible de s'assurer du bon fonctionnement du modem en le configurant avec les paramètres usine par défaut .

Pour ce faire on utilise un logiciel terminal pour envoyer les commandes au modem.

Paramétrage d'un modem de type « 3Com Us Robotics Sportster » :

- Commande : AT&F

Signification : Chargement des paramètres d'usine.

- Commande : AT&W0

Signification : Stockage des paramètres courant dans la mémoire non volatile N°0

- Commande : ATY0

Signification : Définir les paramètres en mémoire non volatile N°0 comme paramètres à utiliser à la mise sous tension.

Paramétrage d'un modem de type « Westermo TD31 ou TD32» :

- Commande : AT&F

Signification : Chargement des paramètres d'usine.

- Commande : AT&W0

Signification : Stockage des paramètres courant dans la mémoire non volatile N°0

- Commande : AT&Y0

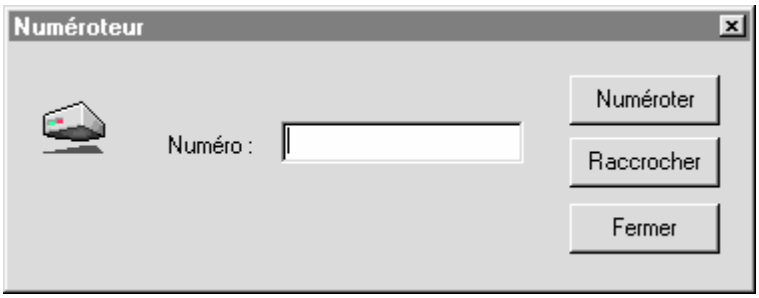
Signification : Définir les paramètres en mémoire non volatile N°0 comme paramètres à utiliser à la mise sous tension.

ATTENTION :

- Pour les modem Westermo, il est également recommandé de laisser la configuration des Dips par défaut (tous sur OFF).

3. Appel

A partir du numéroteur téléphonique intégré au logiciel SPL , on peut établir et interrompre la liaison téléphonique. Le numéroteur téléphonique est accessible à partir du menu communication / télémaintenance.



Après la saisie du numéro de téléphone, appuyer sur le bouton «Numéroter» pour établir la liaison. Le bouton «Raccrocher » permet quant a lui d'arrêter la liaison.

Ces actions ne sont possibles que lorsque le logiciel SPL n'utilise pas cette même liaison en mode Debug par exemple. Pendant la phase de connexion ou de déconnexion le port de communication est indisponible pour le reste de l'application SPL.

Lorsque la liaison est établie, on peut utiliser toutes les fonctions du SPL :

- Envoyer et recevoir la configuration
- Envoyer et recevoir les variables
- Envoyer les tâches
- Démarrer les tâches
- Arrêter les tâches
- Accès aux outils de debug : Hyperterminal, Scope, Trace, Mode manuel.

10-3- Liste de modems validés

- **3 Com / US Robotics**

  - Sportster Voice 33600 Fax Modem
  - Sportster 56 K Fax Modem

- **Westermo**

  - TD 31
  - TD 32

11- ANNEXES

11-1- Message d'erreur de compilation

Find <Type1> <Texte1> : <Type2> <Texte2> Expected

↳ Un identificateur <Texte1> de type <Type1> a été trouvé lors de la compilation au lieu d'un identificateur <Texte2> de type <Type2>.

L or H expected

↳ Pour transformer un entier en Byte on doit utiliser ".L" ou ".H".

<Texte> unexpected : Prog name expected

↳ Le nom d'un programme doit être un identificateur non précédement défini.

Prog bloc already defined

↳ Plus d'un bloc PROG ... END PROG est défini dans la tâche.

<Texte> unexpected : PROG or SUB expected

↳ Un bloc commence par PROG ou SUB. Une instruction a été ajoutée en dehors d'un bloc.

No defined PROG

↳ Le bloc courant n'était pas fini avant la fin du fichier de source.

Undefined Label

↳ Une étiquette inconnue a été utilisée dans une instruction Goto.

Undefined Sub

↳ Un identificateur de sous programme inconnu a été utilisé dans une instruction Call.

Undefined Event

↳ Un événement généré par Signal n'est attendu par aucune tâche ou une tâche attend un événement qui ne sera jamais généré.

Undefined Prog

↳ Un identificateur de programme inconnu a été utilisé dans une instruction Run, Halt, Suspend ou Continue.

SRV15 Card Expected

↳ Pour pouvoir utiliser l'entrée de prise d'origine d'une carte d'axe, dans le paramètre InpHome_p le nom de l'entrée de prise d'origine doit être le même que celui de la carte d'axe auquelle elle appartient.

Instruction expected

↳ Une instruction est attendue.

Buzzer : Bit constant expected

↳ L'instruction Buzzer doit être suivie d'une constante de type bit.

Goto or Call instruction expected

↳ Une instruction Call ou Goto est attendue dans un Case

Invalid exit instruction

↳ Une instruction Exit Sub doit être utilisée uniquement dans un sous-programme.

<Texte> Expected

↳ La variable compteur d'une boucle For doit également être utilisée dans l'instruction Next.

If : Instruction expected

↳ Une instruction est attendue après un If.

Else : Instruction expected

↳ Une instruction est attendue après un Else.

SERIAL1: or SERIAL2: Expected

↳ Dans l'instruction Open le nom du port de communication est soit SERIAL1: ou SERIAL2:.

POS, VEL, ACC or DEC expected

↳ L'instruction TRAJ n'accepte que POS, VEL, ACC ou DEC comme paramètre.

Undefined variable

↳ Le contenu d'une variable est utilisé avant d'avoir été définie par une affectation.

String expression expected

↳ Une expression de type chaîne de caractères est attendue.

Bit expression expected

↳ Une expression de type bit est attendue

Comment bloc : Unexpected end of file.

↳ Un bloc de commentaire débute par '{{'

Comment bloc : Unexpected char

↳ Un caractère autre que '{' a été trouvé.

String constant : Unexpected end of file.

↳ Une constante chaîne de caractères doit être terminée par des guillemets.

Comment bloc : Unexpected end of line

↳ Un bloc de commentaire se termine par '}}'

Bad hex number

↳ Un nombre hexadécimal utilise les caractères 0 à 9 et A à F

Bad binary number

↳ Un nombre binaire utilise les caractères 0 et 1

Not an hex value

↳ Un nombre hexadécimal utilise les caractères 0 à 9 et A à F

Not a binary value

↳ Un nombre binaire utilise les caractères 0 et 1

Not a decimal value

↳ Un nombre décimal utilise les caractères 0 à 9

Real constant : Unexpected end of line

↳ Une constante réelle doit être terminée par un chiffre après le point décimal.

<Texte> unexpected : Char from 0 to 9 expected

↳ Un nombre décimal ou réel utilise les caractères 0 à 9

System constant : Unexpected end of file

↳ Une constante système incomplète a été trouvée.

<Texte> unexpected : System constant expected

↳ Une constante système est attendue.

Number : Unexpected end of file

↳ Un numéro se termine par un chiffre

<Texte> unexpected : Number from 0 to 9 expected

↳ Un numéro se termine par un chiffre

'<Caractère>' unexpected

↳ Un caractère inattendu a été trouvé.

11-2- Messages d'erreur sur l'écran du SUPERVISOR

Liste des messages d'erreurs :


ERREUR N°20 :

L'erreur 20 indique que les données dans la mémoire sauvegardée ont été altérées et qu'il est nécessaire de recharger la configuration et les variables sauvegardées. Le système ne charge pas les paramètres et ne lance pas les tâches utilisateur.


ERREUR N°21 :

L'erreur 21 apparaît à la mise sous tension du SUPERVISOR si un paramètre de la configuration est erroné. Les paramètres doivent être corrigés et transférés avant de pouvoir redémarrer le SUPERVISOR. Le système ne lance pas les tâches utilisateur.


ERREUR N°23 :

L'erreur 23 indique qu'il n'y a pas de tâches utilisateur chargées dans le SUPERVISOR.


ERREUR N°30 :

Lorsqu'un programme utilisateur provoque une division par zéro l'erreur n°30 est affichée.


ERREUR N°31 :

Cette erreur est due à un appel récursif infini d'un sous-programme et indique un débordement de pile.


ERREUR N°32 :

Cette erreur est générée lors d'un dépassement de capacité en virgule flottante provoqué par un nombre trop grand.


ERREUR N°34 :

Lorsqu'une opération invalide en virgule flottante a été détectée cette erreur est générée. Elle se produit avec la fonction REALTOLONG si le nombre réel est trop grand pour être converti en entier long.


ERREUR N°35 :

Cette erreur est générée par un dépassement de capacité arithmétique lors d'un calcul. Ceci se produit lorsque le résultat d'une opération est trop grand pour être stocké dans la variable prévue pour le recevoir.

Ces cinq dernières erreurs sont générées uniquement pendant l'exécution d'un programme utilisateur. Sur la détection d'une erreur quelconque, toutes les tâches seront arrêtées, le message sera visualisé sur l'afficheur, le chien de garde s'ouvrira.


ERREUR N°36 :

Index de tableau de variables sauvegardées hors limites.


ERREUR N°37 :

Index de tableau de variables non sauvegardées hors limites.


ERREUR N°520 :

Erreur d'accès au bus global interne, le bus interne est défectueux.


ERREUR N°530 :

Erreur d'accès au bus global interne, le bus interne est défectueux.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X