Le contrôle de mouvement

Système modulaire MCS 32 EX
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 de la MCS32 EX

1-1-1- Généralités

La MCS32 EX est une commande numérique spécialisée dans le contrôle de mouvement.

Elle offre différents choix de configurations:

- seule
- avec un terminal opérateur Dialog 80 ou Dialog 640
- avec un PC superviseur

Ses ports séries RS232/RS485 lui permettent de se connecter à tout autre périphérique:

- écran tactile
- imprimante
- automate, etc...

De conception modulaire, l'unité centrale accepte jusqu'à dix modules. Ceci permet à la MCS32 EX de s'adapter de façon optimale à toutes applications.

1-1-2- Performances

⇨ Unité centrale:

- Processeur 32 bits à 33 MHz
- 4Mbits de RAM sauvegardée
- 8Mbits de mémoire Flash
- 2 ports séries - 1200 à 9600 bauds
- Gestion de 1 à 8 axes
- Gestion de 8 à 180 entrées/sorties
- Horloge temps réel
- Chien de garde

⇨ Cartes Servo SRV 85:

- DSP 40 MHz
- 2 entrées de capture de position très rapide (0,1µs)
- Codeur incrémental 1,5MHz
- Sortie analogique +/- 10V (servo cycle time 330µs)
- Entrée capteur 24 Vcc

1-1-3- Modularité

La MCS32 EX offre un important choix de modules pour lui permettre de s'adapter à de nombreuses applications:

- Codeur, Servo - incrémental ou SSI
- Entrées/sorties TOR - 8,16 ou 24 voies

- Entrées/sorties analogiques - 2 ou 4 voies sur 12 bits
- Carte de communication CANOpen

1-2- Description du logiciel MCB EX

1-2-1- Généralités

MCB EX est un logiciel de développement puissant sous environnement WINDOWS 95, 98 ou NT.

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

1-3- Applications

- Tables X, Y - manipulateurs 1 à 8 axes
- Mise au pas de produits
- Trancannage
- Came électronique
- Interpolation linéaire et circulaire
- Coupe au vol, mise en registre
- Arbre électrique à rapport variable
- Dépose d'étiquettes à la volée
- Synchronisation de couleurs
- Boîte à cames

2- INSTALLATION / MISE EN OEUVRE

2-1- Conditions d'utilisation

2-1-1- Conditions d'utilisation

La MCS doit être installée verticalement pour assurer un refroidissement naturel par convection.

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

Caractéristiques techniques :

- Alimentation du rack :100 à 250Vac 30VA
- Chien de garde : Contact NO libre de potentiel 48Vac maxi 2A maxi
- Température de service : 0 à 45°C
- Température de stockage : -10 à 70°C

La MCS est munie d'une alimentation 5Vdc 1.6A pour alimenter les codeurs.

2-2- Sécurités

2-2-1- 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 de la MCS 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.
- Le paramètre "Erreur de poursuite maxi" de chaque axe devra être réglé.
- Des capteurs, pour limiter la course de chacun des axes devront être reliés en série dans la boucle d'arrêt d'urgence.
- Il est recommandé d'utiliser les butées logicielles dans le cas des axes finis.
- Les validations des variateurs devront être gérées par la MCS (utiliser la sortie watch dog ou une sortie quelconque -*voir instruction AXIS_S*)
- 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.
- Sur détection d'une erreur de poursuite, la MCS passe tous les axes en boucle ouverte (consigne analogique forcé à 0) et ouvre le chien de garde. Si un autre traitement est souhaité, il faut utiliser l'instruction SECURITY.

2-3- Raccordements

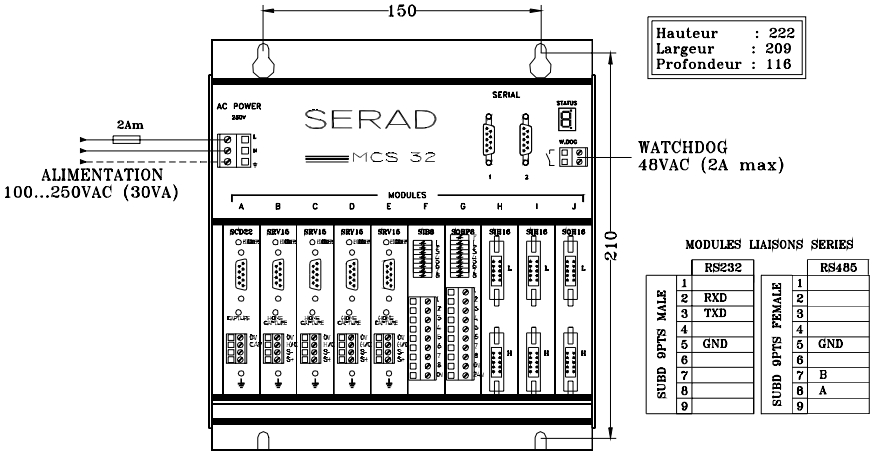
2-3-1- Explication générale

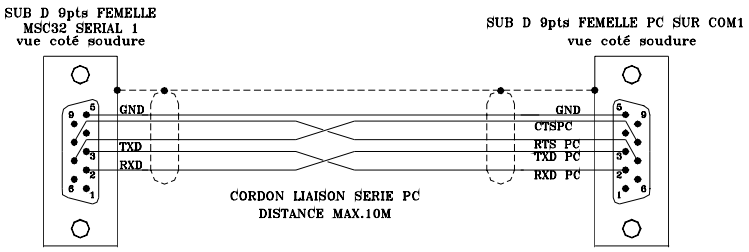
- Il est recommandé d'alimenter la MCS par un transformateur de séparation avec écran électrostatique.
- Les câbles codeur, consigne analogique, terminal opérateur devront être blindés, la tresse étant reliée de chaque côté au chassis.

↳ Le câble de liaison MCS / PC devra être blindé, la tresse étant reliée de chaque côté au chassis. Il devra être débranché de la MCS 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- Unité centrale





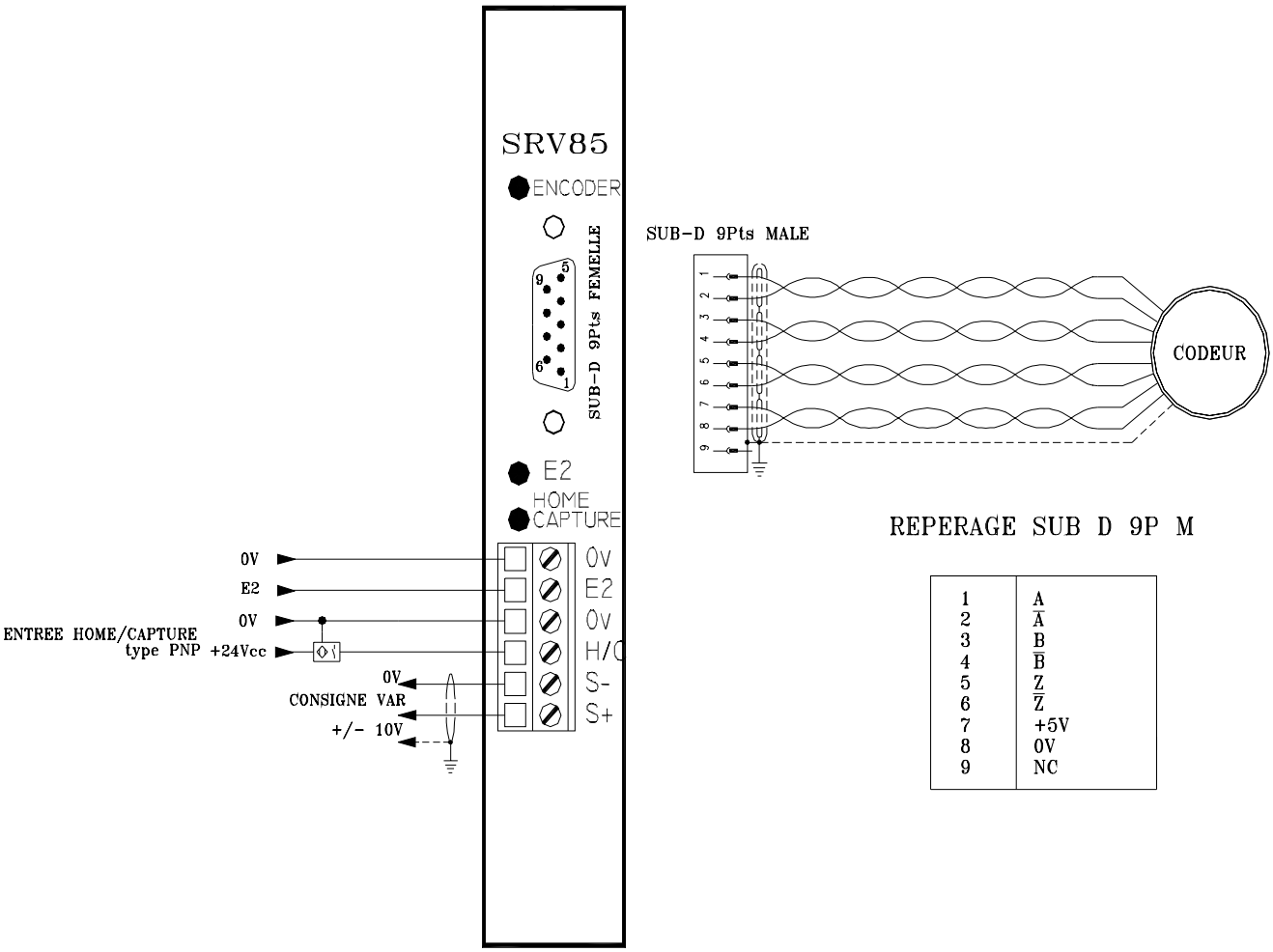
WATCHDOG est un contact normalement ouvert libre de potentiel de 48 VACmaxi et de 2A maxi

2-3-3- Module servo : SRV85

Caractéristiques

- Codeur incrémental 5 Volts - driver de ligne - Fréquence max : 1.5 Mhz.
- Le codeur est alimenté par la MCS 32 (1.6A Max pour tous les codeurs).
- 2 Entrées de capture à 0,1µs ou entrée prise d' origine de type PNP : 12mA / 20..33 Vcc
- Sortie analogique +/- 10V / 5mA Max, Résolution 12 bits, 0V référencé par rapport au chassis via une résistance de 470KΩ et un condensateur de 4,7nF
- Led de visualisation de la voie Z du codeur
- Led de visualisation de l'entrée de capture/prise d'origine

Connexions



REPERAGE SUB D 9P M

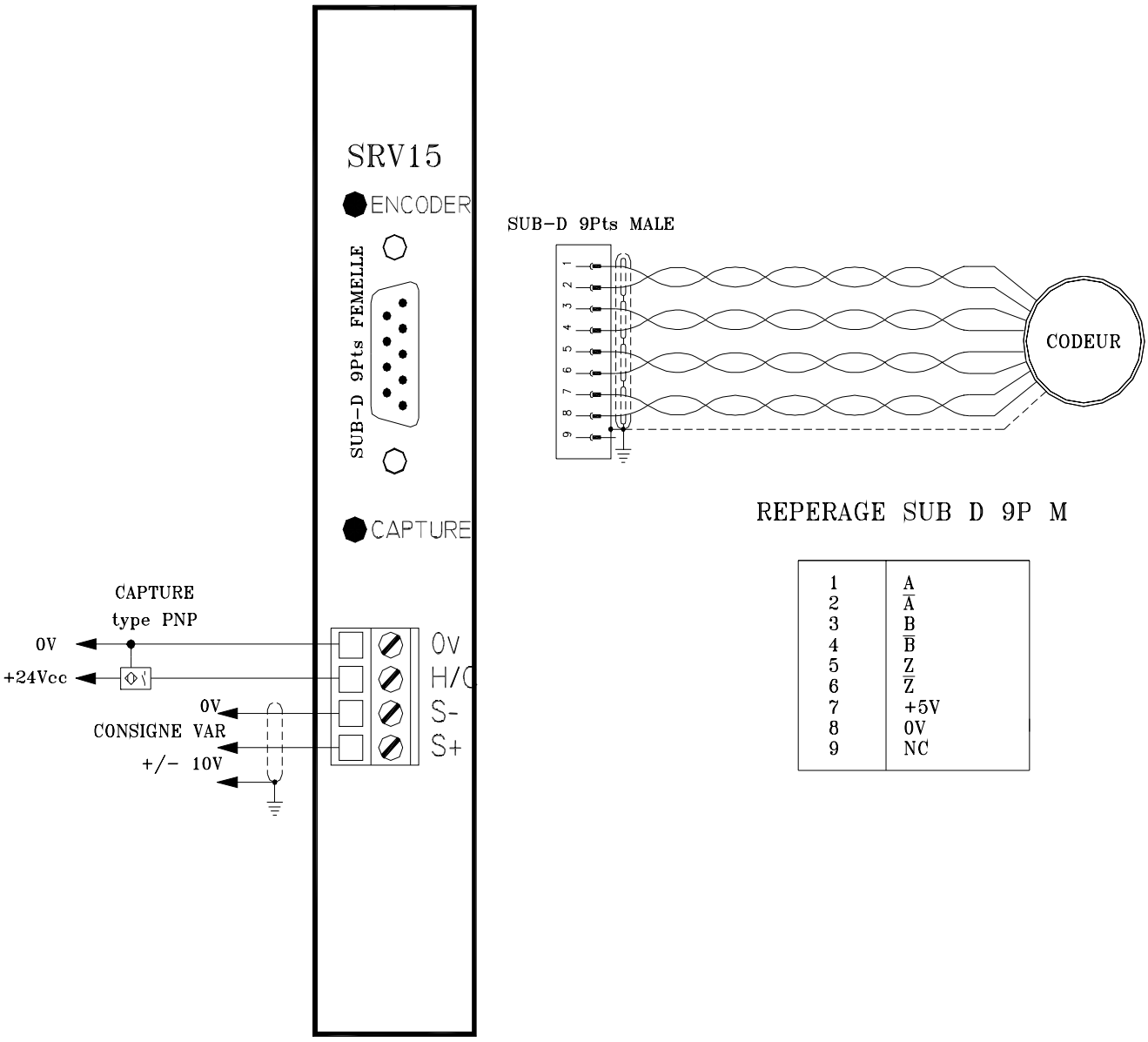
1	A
2	$\overline{A}$
3	B
4	$\overline{B}$
5	Z
6	$\overline{Z}$
7	+5V
8	0V
9	NC

2-3-4- Module servo : SRV15

Caractéristiques

- Codeur incrémental 5 Volts - driver de ligne - Fréquence max : 1.5 Mhz.
- Le codeur est alimenté par la MCS 32 (1.6A Max pour tous les codeurs).
- Entrée de capture à 0,1µs ou entrée prise d' origine de type PNP : 12mA / 20..33 Vcc
- Sortie analogique +/- 10V / 5mA Max, Résolution 12 bits, 0V référencé par rapport au chassis via une résistance de 470KΩ et un condensateur de 4,7nF
- Led de visualisation de la voie Z du codeur
- Led de visualisation de l'entrée de capture/prise d'origine

Connexions



REPERAGE SUB D 9P M

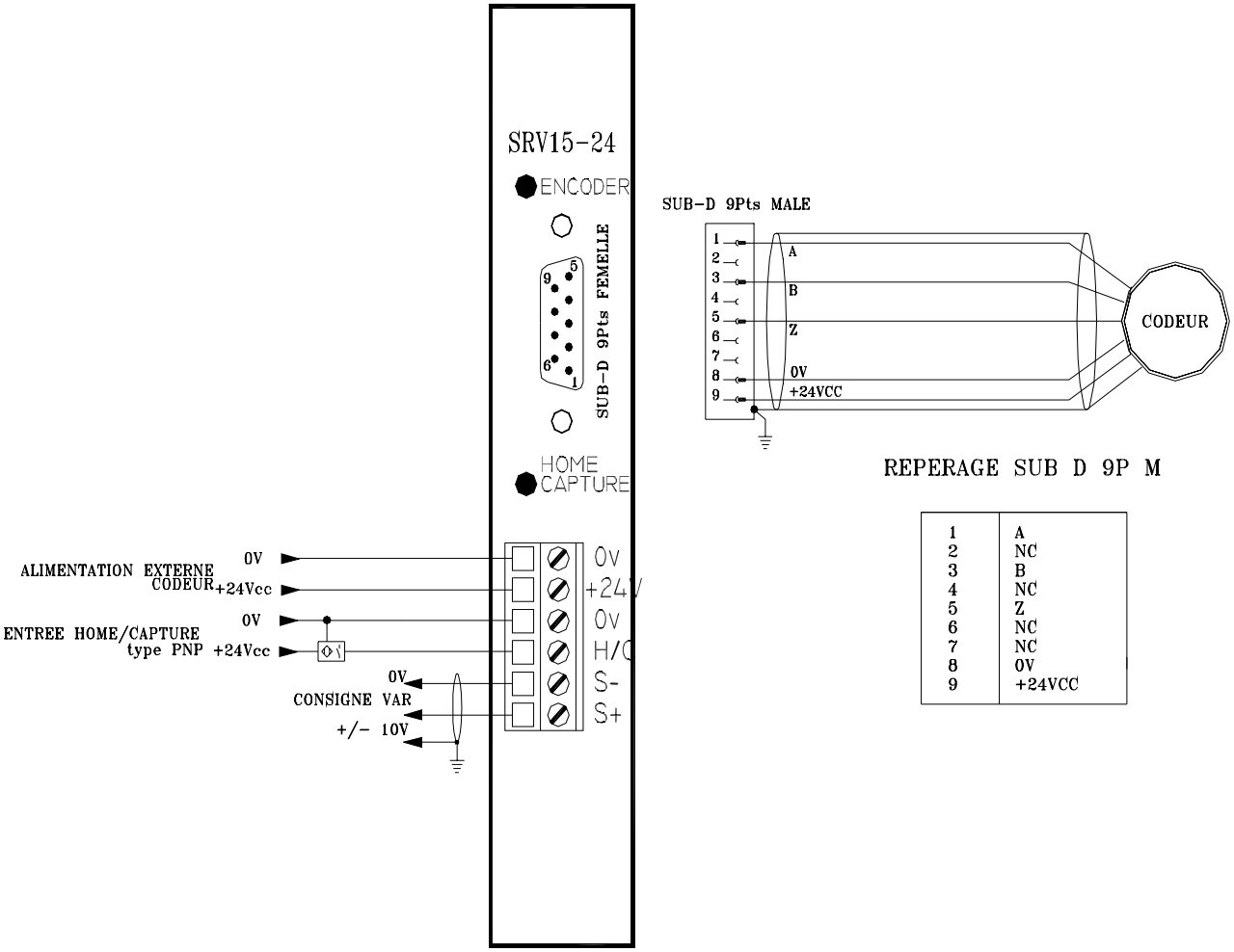
1	A
2	$\overline{A}$
3	B
4	$\overline{B}$
5	Z
6	$\overline{Z}$
7	+5V
8	0V
9	NC

2-3-5- Module servo : SRV15-24

Caractéristiques

- Codeur incrémental 24 Volts - driver de ligne - Fréquence max : 1.5 Mhz.
- Le codeur est alimenté par la MCS 32 (1.6A Max pour tous les codeurs).
- Entrée de capture à 0,1µs ou entrée prise d' origine de type PNP : 12mA / 20..33 Vcc
- Sortie analogique +/- 10V / 5mA Max, Résolution 12 bits, 0V référencé par rapport au chassis via une résistance de 470KΩ et un condensateur de 4,7nF
- Led de visualisation de la voie Z du codeur
- Led de visualisation de l'entrée de capture/prise d'origine

Connexions



REPERAGE SUB D 9P M

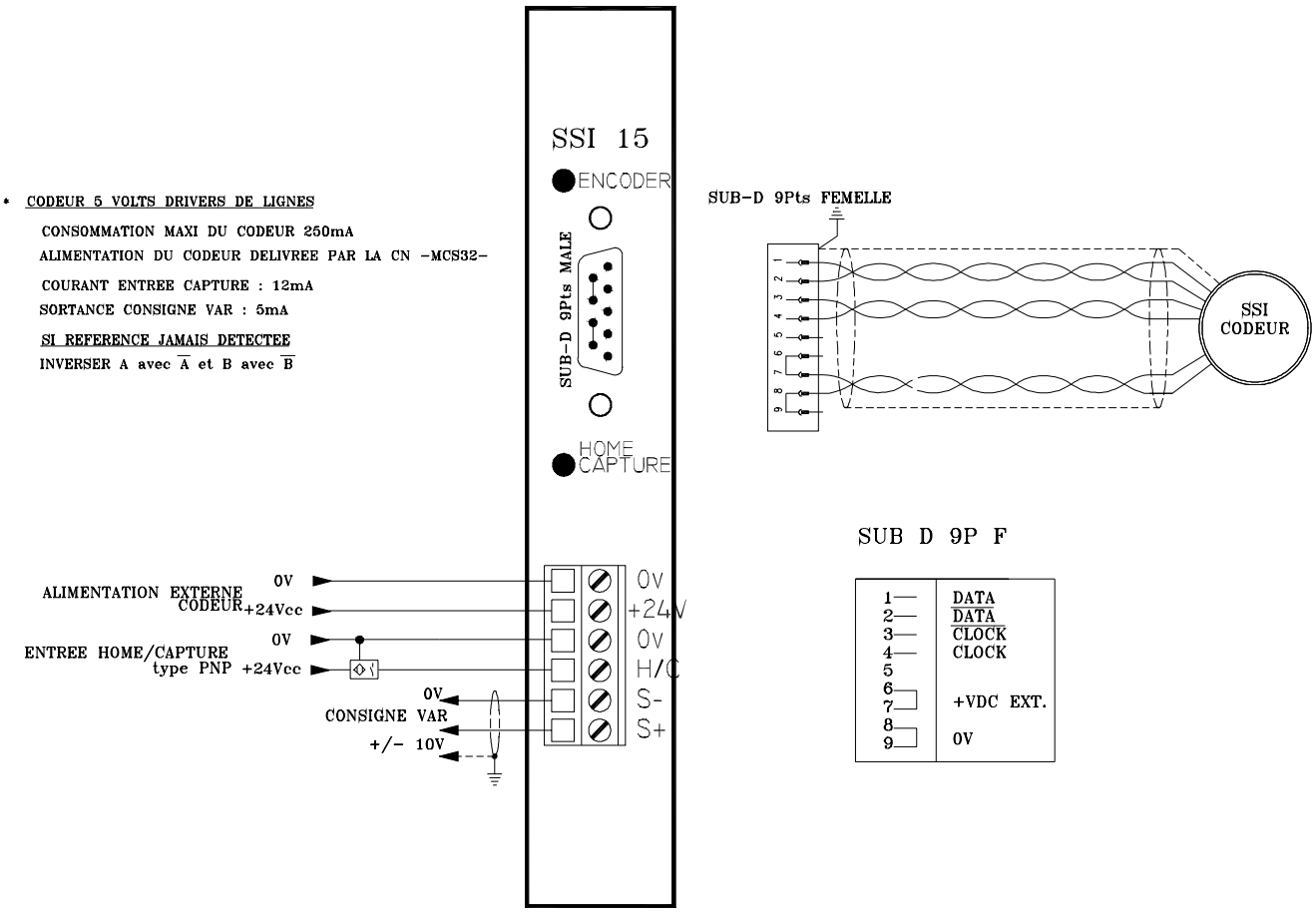
1	A
2	NC
3	B
4	NC
5	Z
6	NC
7	NC
8	0V
9	+24VCC

2-3-6- Module servo SSI : SSI15

Caractéristiques

- Codeur SSI 11..30V - driver de ligne. Alimentation externe 11..30V.
- Fréquence d'horloge : 100KHz..1MHz, code gray, sans bit de parité.
- Entrée de capture à 0,1µs ou entrée prise d' origine de type PNP : 12mA / 20..33 Vcc
- Sortie analogique +/- 10V / 5mA Max, Résolution 12 bits, 0V référencé par rapport au chassis via une résistance de 470KΩ et un condensateur de 4,7nF
- Led de visualisation de l'entrée de capture/prise d'origine

Connexions

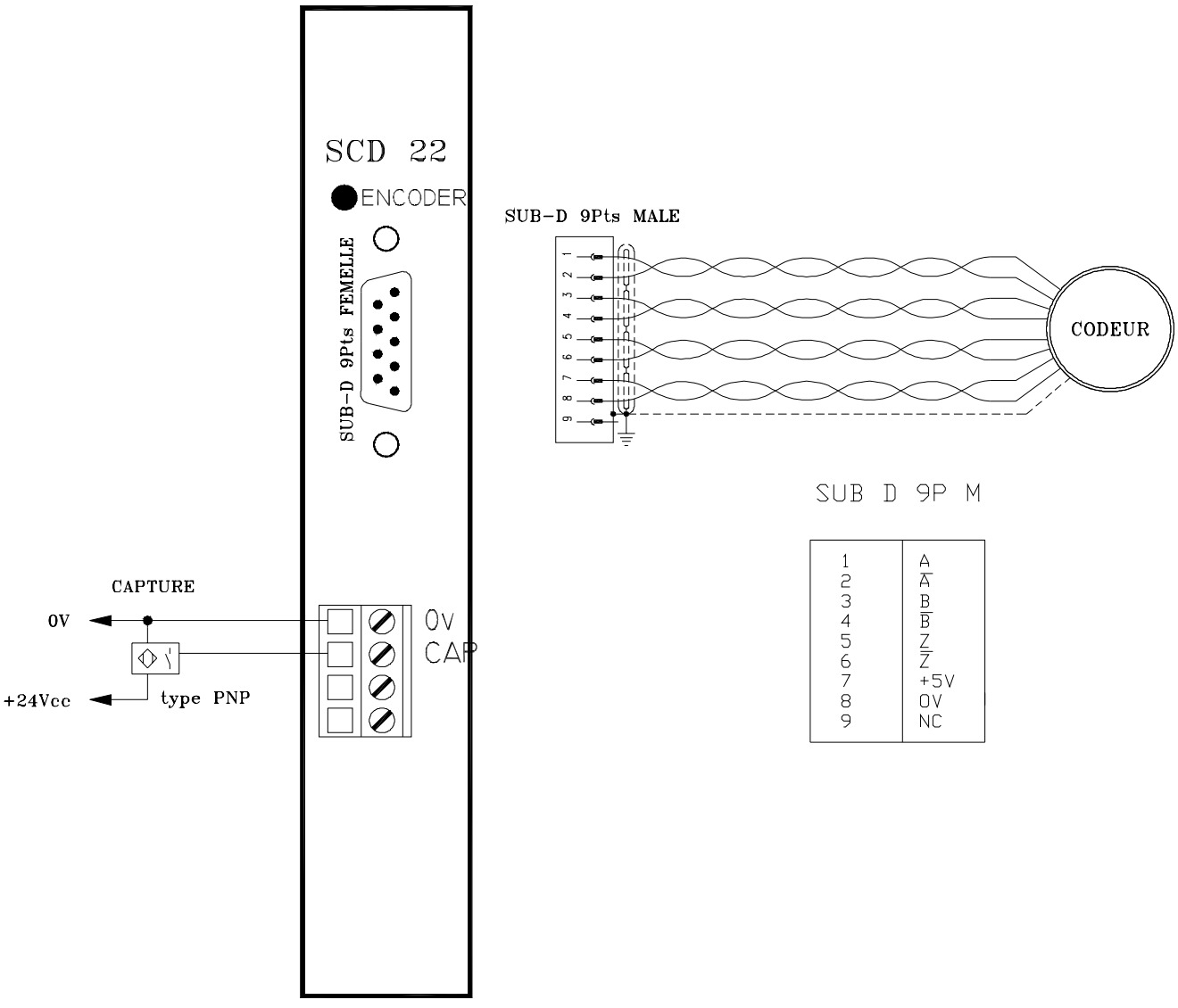


2-3-7- Module codeur : SCD22

Caractéristiques

- Codeur incrémental 5 Volts - driver de ligne - Fréquence max : 1.5 Mhz.
- Le codeur est alimenté par la MCS 32 (1.6A Max pour tous les codeurs).
- Entrée de capture à 0.1µs de type PNP : 12 mA / 20..33 Vcc
- Led de visualisation de la voie Z du codeur
- Led de visualisation de l'entrée de capture

Connexions

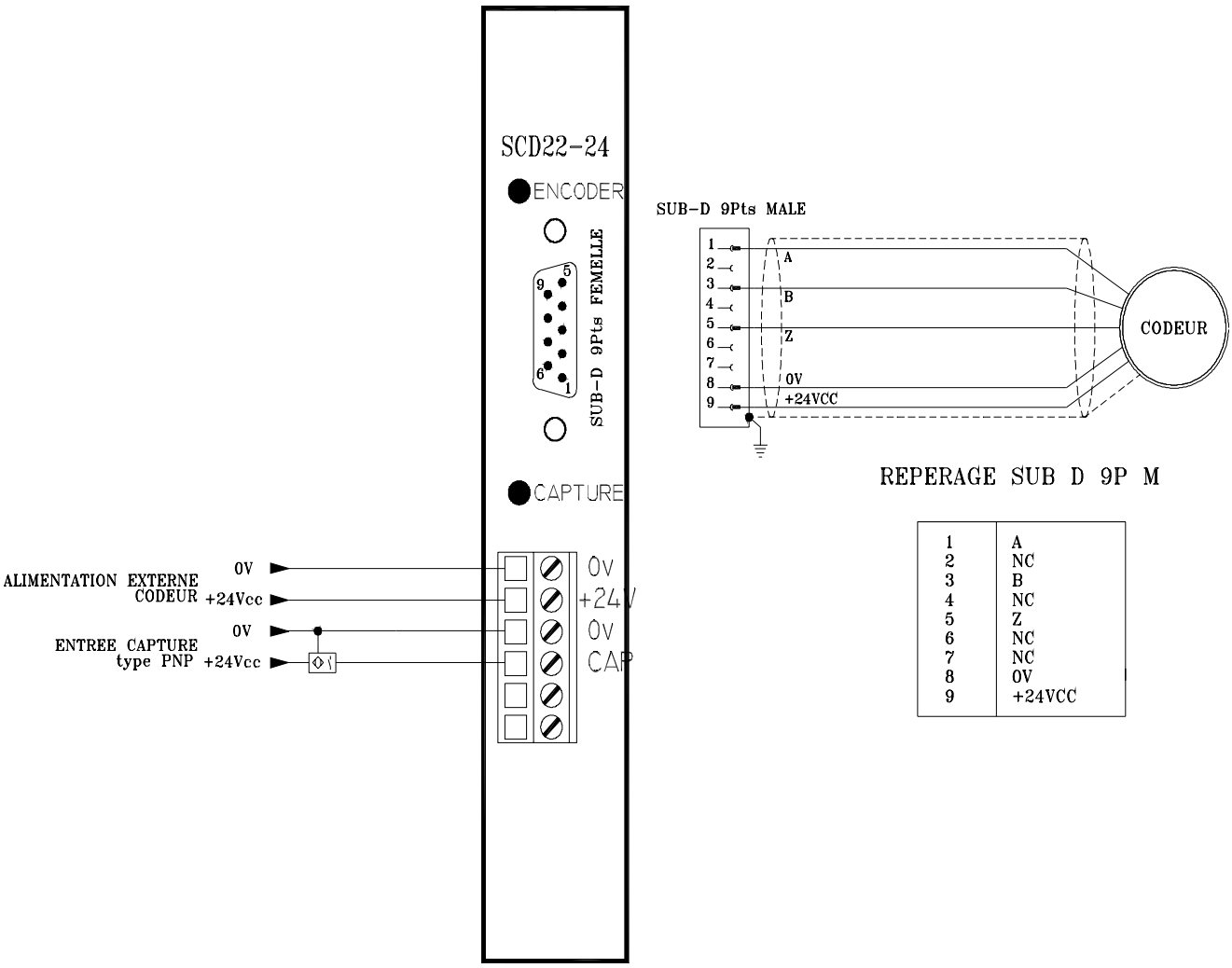


2-3-8- Module codeur : SCD22-24

Caractéristiques

- Codeur incrémental 24 Volts - driver de ligne - Fréquence max : 1.5 Mhz.
- Le codeur est alimenté par la MCS 32 (1.6A Max pour tous les codeurs).
- Entrée de capture à 0.1µs de type PNP : 12 mA / 20..33 Vcc
- Led de visualisation de la voie Z du codeur
- Led de visualisation de l'entrée de capture

Connexions



REPERAGE SUB D 9P M

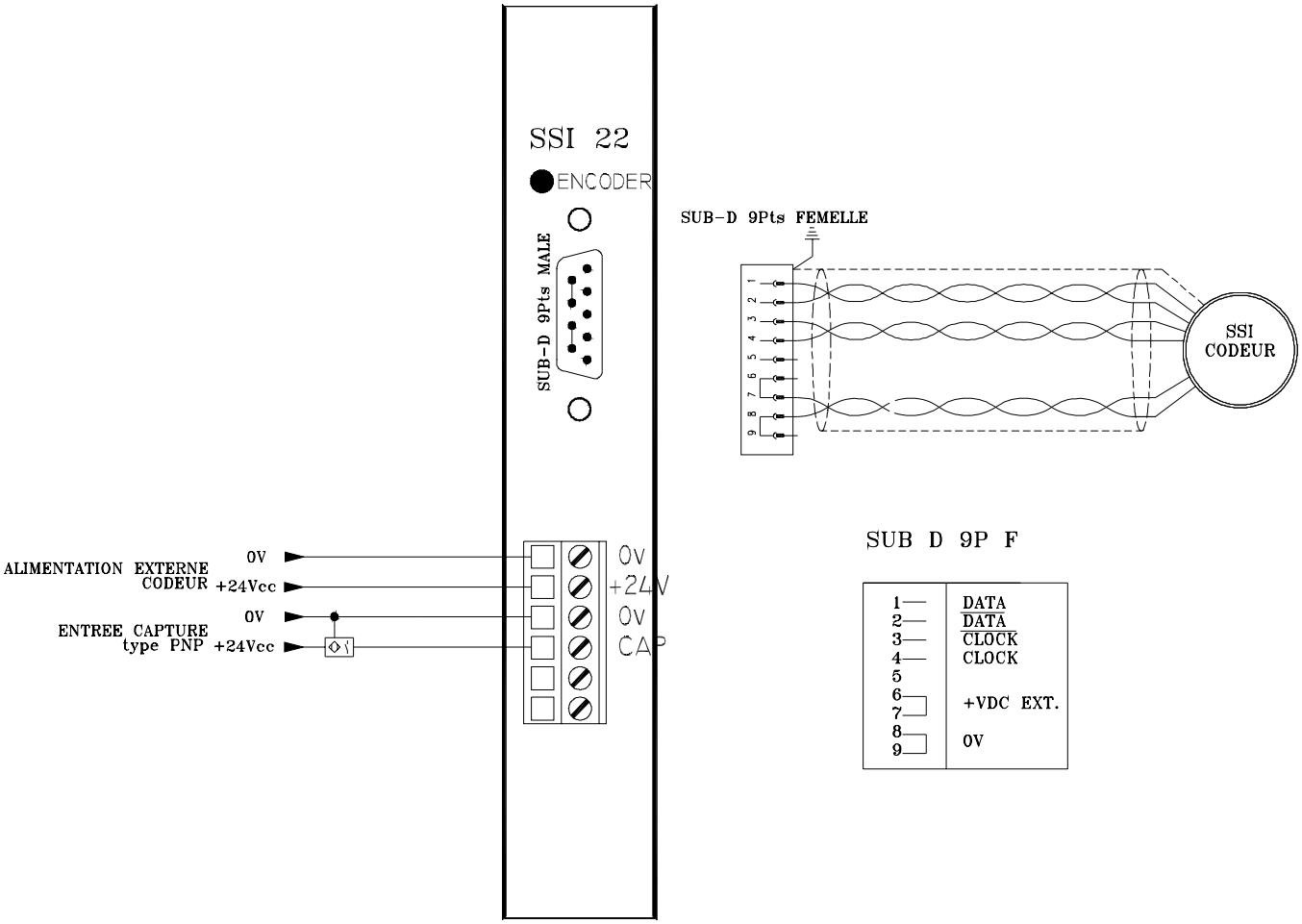
1	A
2	NC
3	B
4	NC
5	Z
6	NC
7	NC
8	0V
9	+24VCC

2-3-9- Module codeur SSI : SSI22

Caractéristiques

- Codeur SSI 11..30V - driver de ligne.
- Fréquence d'horloge : 100Khz ..1MHz; code gray, sans bit de parité
- Entrée de capture à 0.1µs de type PNP : 12 mA / 20..33 Vcc
- Led de visualisation de l'entrée de capture

Connexions

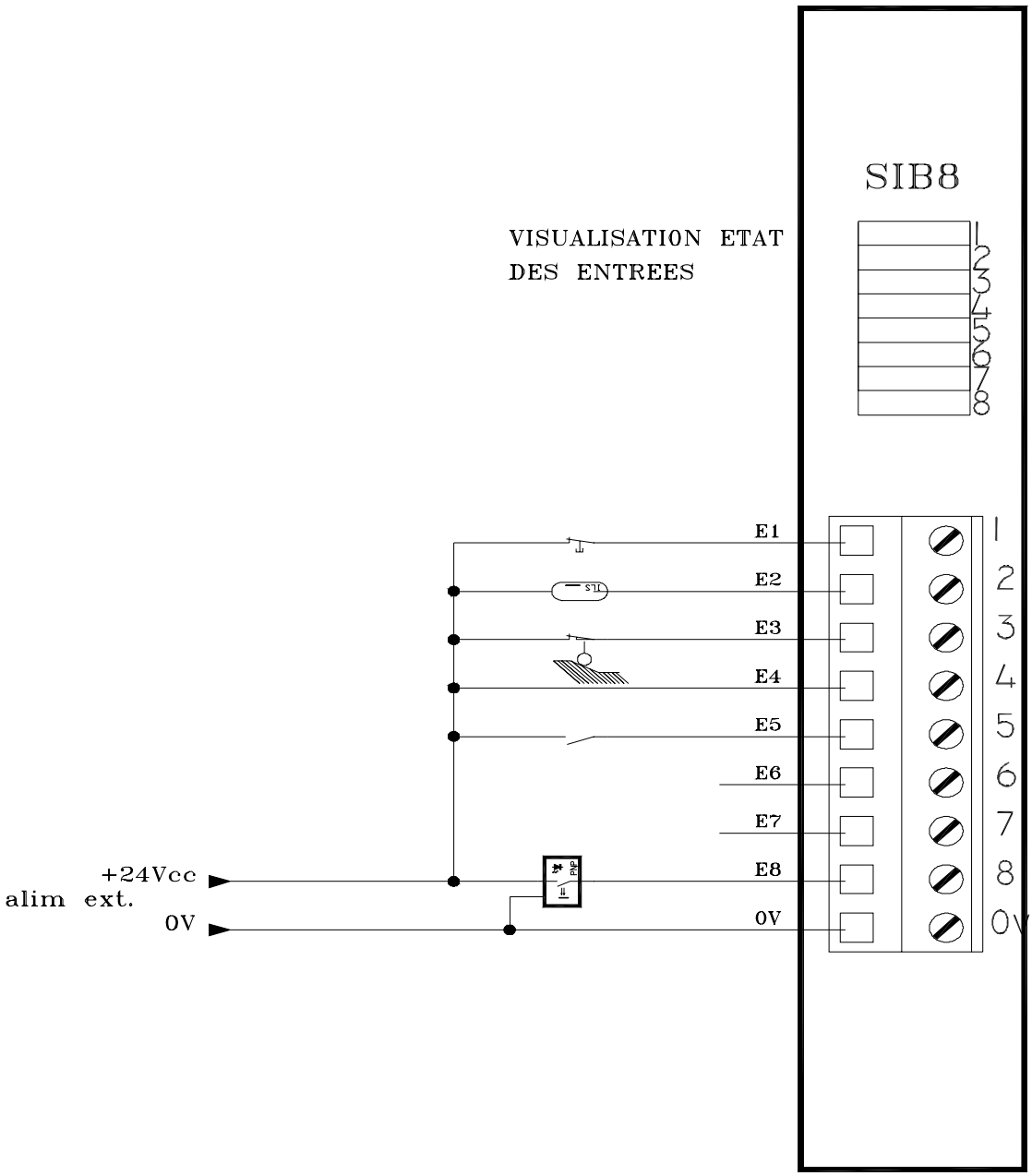


2-3-10- Module 8 entrées TOR : SIB8

Caractéristiques

- Entrées opto-isolées de type PNP.
- Courant d'entrée : 12mA par entrée.
- Alimentation entre 20 à 33 Vcc
- Période de lecture des entrées : dépend directement de la valeur du filtrage logiciel paramétrables à partir du menu manuel de la carte. (de 1 à 255ms)

Connexions

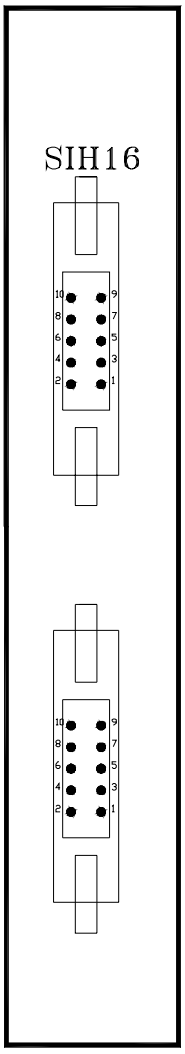


2-3-11- Module 16 entrées TOR : SIH16

Caractéristiques

- Entrées Opto-isolées type PNP
- Alimentation entre 20 à 33 Vcc
- Directement compatible avec les modules d'interface 8 entrées MIHB 8
- Courant d'entrée : 12mA par entrée.
- Période de lecture des entrées : dépend directement de la valeur du filtrage logiciel paramétrables à partir du menu manuel de la carte. (de 1 à 255ms)

Connexions



CONNECTEUR A HE10

1	ENTREE 8
2	ENTREE 7
3	ENTREE 6
4	ENTREE 5
5	ENTREE 4
6	ENTREE 3
7	ENTREE 1
8	ENTREE 2
9	0V
10	0V

CONNECTEUR B HE10

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

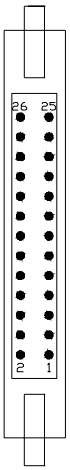
2-3-12- Module 24 entrées TOR : SIH24

Caractéristiques

- Entrées Opto-isolées type PNP
- Alimentation entre 20 à 33 Vcc
- Directement compatible avec les modules d'interface 8 entrées MIHB 24
- Courant d'entrée : 12mA par entrée.
- Période de lecture des entrées : dépend directement de la valeur du filtrage logiciel paramétrables à partir du menu manuel de la carte. (de 1 à 255ms)

Connexions

SIH 24



CONNECTEUR A HE10

1	0V
2	0V
3	ENTREE 12
4	ENTREE 24
5	ENTREE 11
6	ENTREE 23
7	ENTREE 10
8	ENTREE 22
9	ENTREE 9
10	ENTREE 21
11	ENTREE 8
12	ENTREE 20
13	ENTREE 7
14	ENTREE 19
15	ENTREE 6
16	ENTREE 18
17	ENTREE 5
18	ENTREE 17
19	ENTREE 4
20	ENTREE 16
21	ENTREE 3
22	ENTREE 15
23	ENTREE 2
24	ENTREE 14
25	ENTREE 1
26	ENTREE 13

CARACTERISTIQUES

ENTREES OPTO-ISOLEES
ALIMENTATION 20 à 33VCC
MODULE INTERFACE 24 ENTREES MIB 24 DIRECTEMENT COMPATIBLE
ENTREES TYPE PNP
COURANT PAR ENTREE : 12mA

2-3-13- Module 8 sorties TOR : SOBP8

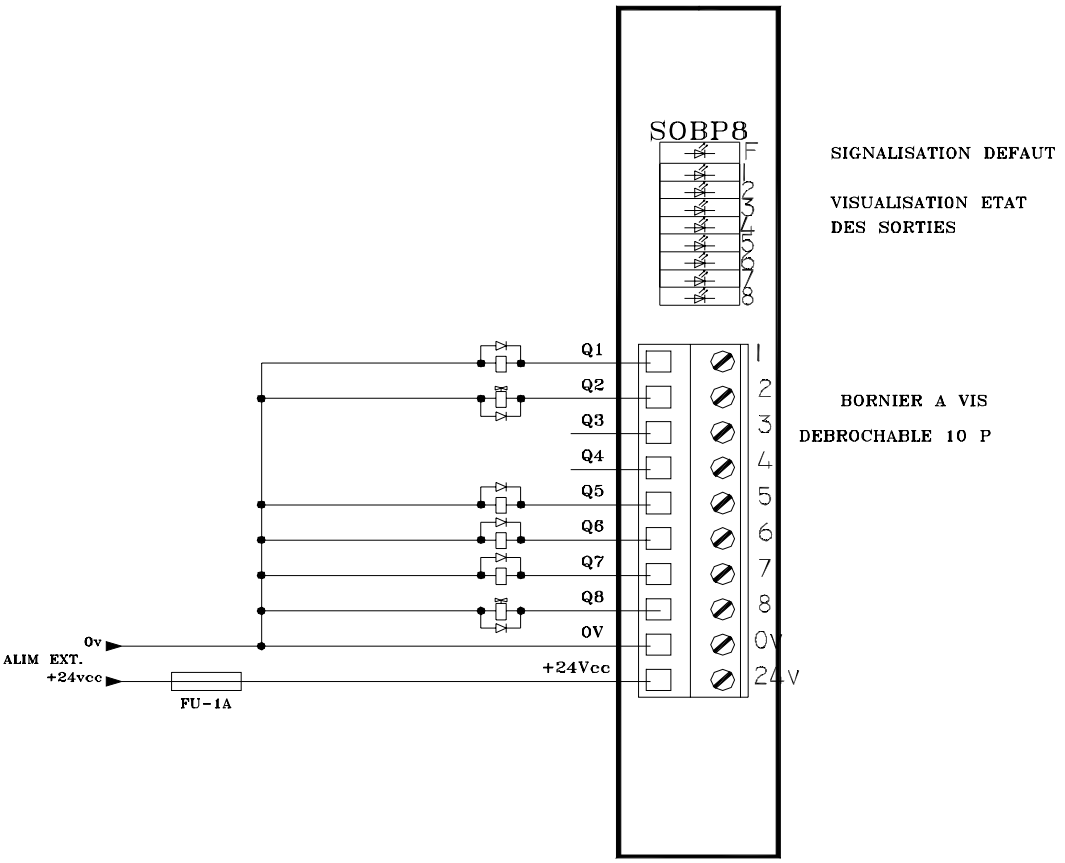
Caractéristiques

- Sorties opto-isolées type PNP (compatible automate).
- Courant de sorties : 350mA maxi par sortie.
- Courant de sortie total sur 8 voies : 800mA maxi.
- Protégé contre les court-circuits et les surchauffes
- Courant de pointe maximal : 350mA
- Période de rafraîchissement des sorties : 1ms

Remarques :

- Si un court-circuit est détecté , l'alimentation 24V doit être arrêtée pour réarmer le module
- Ajouter des diodes de roue libre sur les charges inductives
- Les charges capacitives sont interdites

Connexions



CARACTERISTIQUES

COMPATIBLE AVEC LES ENTREES AUTOMATE PNP
350mA MAX PAR VOIE
800mA MAX TOTAL DES 8 VOIES
PROTECTION THERMIQUE ET COURT-CIRCUIT
REARMEMENT PAR COUPURE DE 24 VOLTS
COURANT APPEL 350mA MAX

IMPORTANT

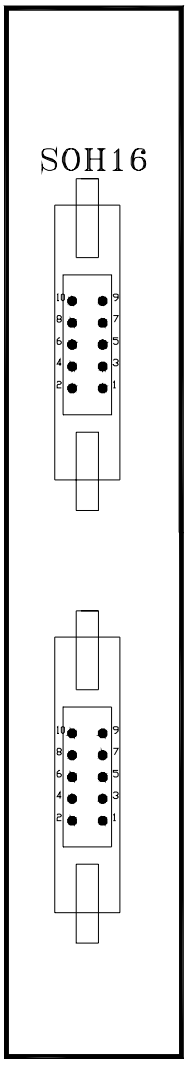
PREVOIR UNE DIODE DE ROUE LIBRE SUR CHARGE INDUCTIVE
CHARGE CAPACITIVE INTERDITE

2-3-14- Module 16 sorties TOR : SOH16

Caractéristiques

- Sorties statiques NPN isolées (non-protégées)
- Collecteur ouvert
- Alimentation entre 20 à 33 Vcc
- Directement compatible avec le module interface 8 relais : MRHB 8
- Courant de sortie : 100mA par sortie.
- Période de rafraîchissement des sorties : 1ms

Connexions



CONNECTEUR A HE10

1	0V
2	SORTIE 6
3	SORTIE 7
4	+24VCC
5	SORTIE 8
6	SORTIE 5
7	SORTIE 1
8	SORTIE 4
9	SORTIE 2
10	SORTIE 3

CONNECTEUR B HE10

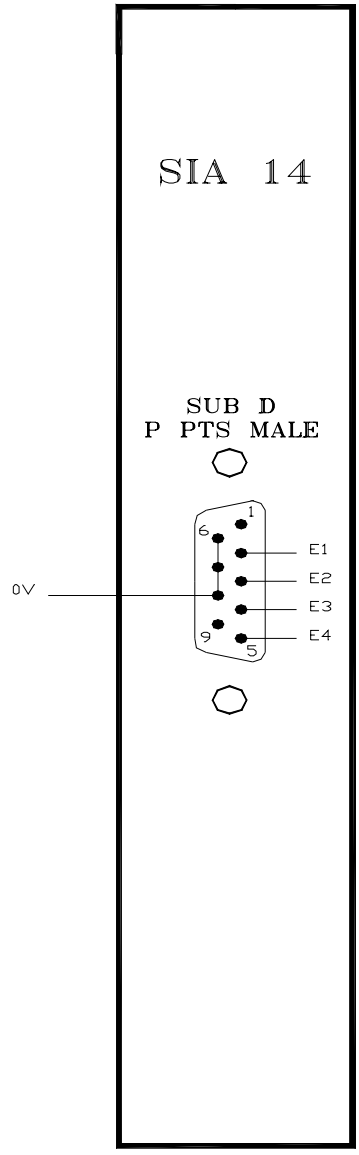
1	0V
2	SORTIE 14
3	SORTIE 15
4	+24VCC
5	SORTIE 16
6	SORTIE 13
7	SORTIE 9
8	SORTIE 12
9	SORTIE 10
10	SORTIE 11

2-3-15- Module 4 entrées analogiques : SIA14

Caractéristiques

- Entrées opto-isolées
- Tension d'entrée +/-10V
- Tension d'entrée maximale admissible ±20V
- Impédance d'entrée 1MΩ
- Résolution (1 LSB) 4.88 mV
- Convertisseur analogique numérique 12 bits
- Période de lecture des entrées : 9ms

Connexions



REPERAGE SUB D 9P MALE

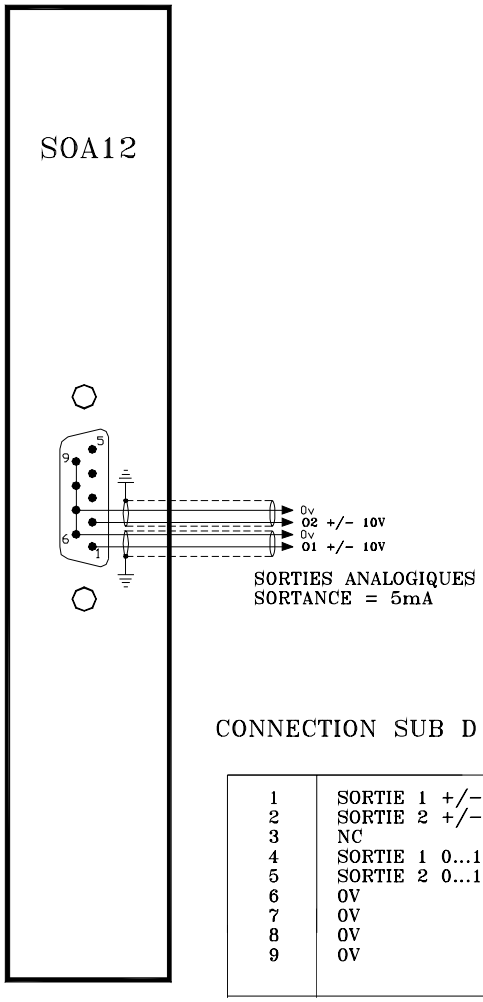
1	NC
2	E1
3	E2
4	E3
5	E4
6	0v
7	0v
8	0v
9	NC

2-3-16- Module 2 sorties analogiques : SOA12

Caractéristiques

- Sorties opto-isolées
- Tension de sortie +/-10V
- Résolution (1 LSB) 4.88 mV
- Courant de sortie maximal : 5mA
- Convertisseur numérique analogique 12 bits
- Période de rafraîchissement des sorties : 9ms

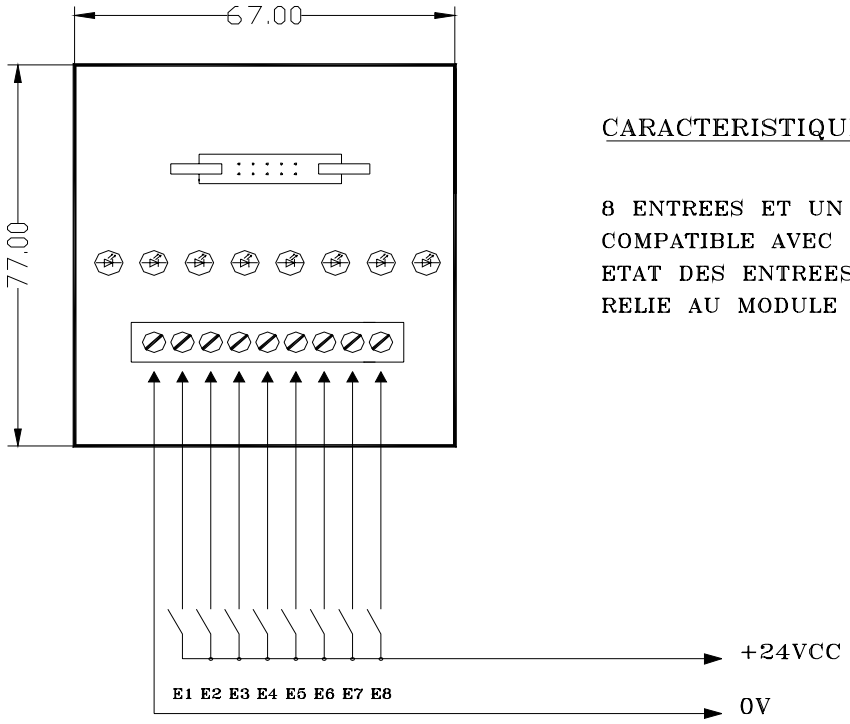
Connexions



CONNECTION SUB D 9P F.

1	SORTIE 1 +/- 10V
2	SORTIE 2 +/- 10V
3	NC
4	SORTIE 1 0...10V
5	SORTIE 2 0...10V
6	0V
7	0V
8	0V
9	0V

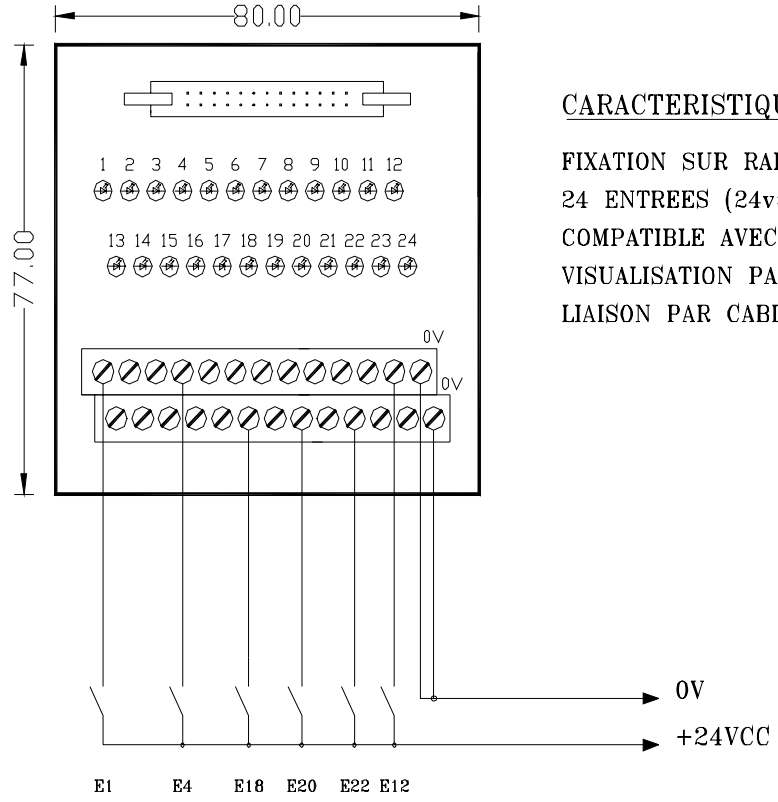
2-3-17- Module interface 8 entrées : MIHB8



CARACTERISTIQUES

8 ENTREES ET UN COMMUN
COMPATIBLE AVEC LES CAPTEURS PNP
ETAT DES ENTREES PAR LED
RELIE AU MODULE SIH16 PAR UN CABLE PLAT

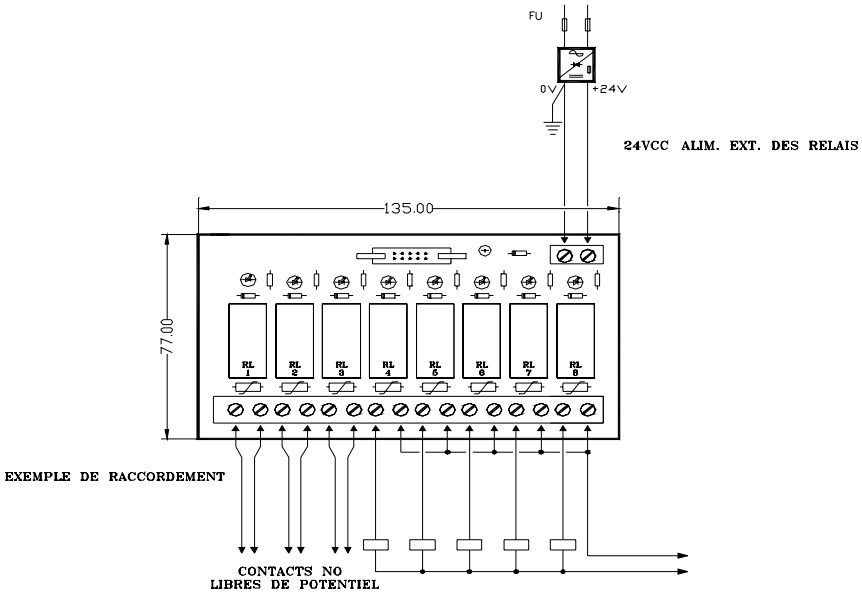
2-3-18- Module interface 24 entrées : MIHB24



CARACTERISTIQUES

FIXATION SUR RAIL DIN PAR SIMPLE ENCLIQUETAGE
24 ENTREES (24v=) 1 COMMUN SUR BORNIER A VIS
COMPATIBLE AVEC DETECTEUR PNP
VISUALISATION PAR LED DE L'ETAT DES ENTREES
LIAISON PAR CABLE PLAT AU MODULE SIH 24

2-3-19- Module interface 8 relais : MRHB8



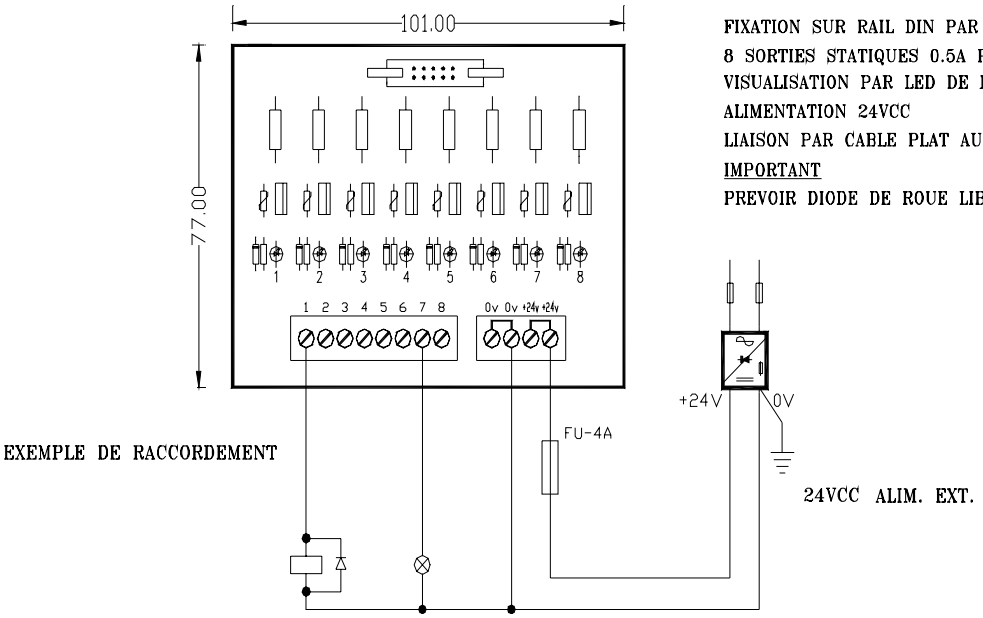
CARACTERISTIQUES

FIXATION SUR RAIL DIN PAR SIMPLE ENCLIQUETAGE
8 SORTIES RELAIS 5A A CONTACT NO PROTEGE PAR VARISTANCE
2 BORNES PAR SORTIE
VISUALISATION PAR LED DE L'ETAT DES SORTIES
ALIMENTATION 24VCC (de 20 à 33 volts) 6VA
LIAISON PAR CABLE PLAT AU MODULE MOH 16
IMPORTANT
PREVOIR DIODE DE ROUE LIBRE SUR CHARGE INDUCTIVE EN CONTINU
PREVOIR FILTRE RC SUR CHARGE INDUCTIVE EN ALTERNATIF

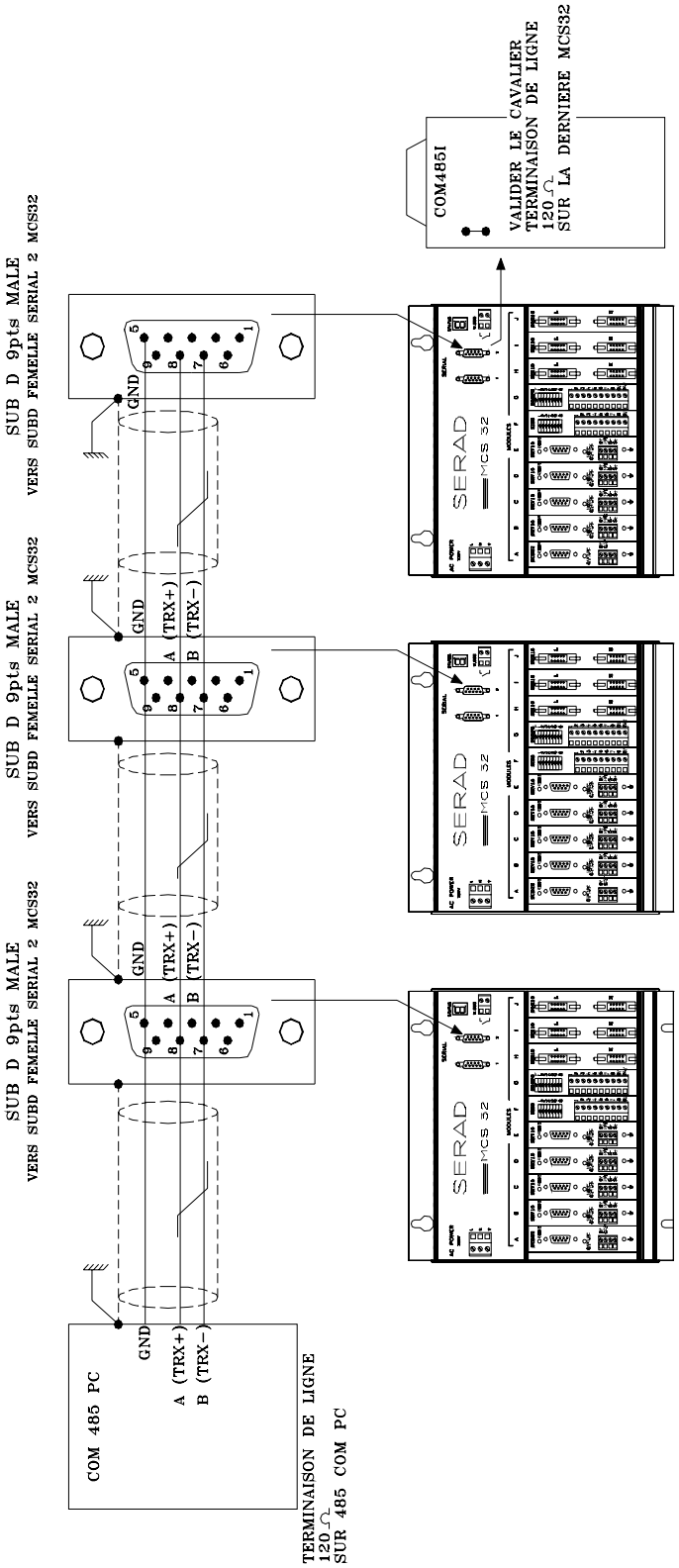
2-3-20- Module interface 8 sorties statiques : MSHB8

CARACTERISTIQUES

FIXATION SUR RAIL DIN PAR SIMPLE ENCLIQUETAGE
8 SORTIES STATIQUES 0.5A PROTEGEES PAR FUSIBLE REARMABLE
VISUALISATION PAR LED DE L'ETAT DES SORTIES
ALIMENTATION 24VCC
LIAISON PAR CABLE PLAT AU MODULE MOH 16
IMPORTANT
PREVOIR DIODE DE ROUE LIBRE SUR CHARGE INDUCTIVE



2-3-21- Module interface RS485 : COM485



2-4- Mise en route

2-4-1- Mise en route

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

- On commence par définir l'emplacement des cartes avec l'écran de configuration du MCB.
- 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 MCB.
- On définit les données globales.
- On transfère la valeur initiale des variables sauvegardées en utilisant "Envoyer les données" à partir du menu communication du MCB.
- 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 la MCS. Une modification de la configuration nécessite un nouvel envoi de la configuration dans la MCS.

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

2-5- Procédure de réglage d'un axe

2-5-1- Procédure de réglage d'un axe

L'asservissement de position d'un axe est constitué de deux ensembles :

1 : Le variateur - moteur (avec tachy si moteur courant continu).

2 : La commande numérique - codeur (ou règle de mesure)

Le codeur indique à la MCS la position de l'axe.

La MCS envoie une tension analogique au variateur pour commander le moteur.

1) Réglage de l'axe en boucle ouverte

La partie variateur-moteur doit d'abord être réglée.

Dans le cas d'un moteur courant continu, si à la mise sous tension, le moteur s'emballe, inversez les fils de la tachy.

Branchez ensuite une tension continue variable de -9V à +9V (souvent disponible sur le variateur ou alors à partir du menu debug de la carte « Forçage d'une tension en mode débrayé ») sur l'entrée analogique du variateur et vérifier le bon comportement du moteur.

2) Réglage de l'axe en boucle fermée

Reliez le codeur à la commande numérique.

Reliez la sortie consigne analogique de la MCS sur l'entrée analogique du variateur.

Entrez des paramètres cohérents dans la MCS : Gain=500, Dérivée=0, Intégrale=0, Résolution codeur, unités par tour, vitesse, accélération...

Si à la mise sous tension le moteur s'emballe, on peut résoudre ce problème de la manière suivante :

Inversez le paramètre "Inversion du sens codeur" ou le paramètre "Inversion de la consigne".

Le système doit être stable.

Sens de comptage

Ajustez les valeurs des paramètres "Inversion de la consigne" et "Inversion du sens codeur" pour obtenir un sens de comptage correct.

P.I.D.

Réglez les paramètres de P.I.D. Dans la plupart des cas, seule une valeur de gain proportionnel suffit, la dérivée et l'intégrale étant forcées à zéro.

Un gain trop faible donne une mauvaise précision à l'arrêt. Un gain trop fort rend le système instable (oscillations plus ou moins fortes).

Si l'axe est utilisé avec les fonctions de synchronisation, mettez le gain proportionnel à une valeur faible, réglez les gains d'anticipation de vitesse et d'accélération pour tendre vers une erreur de poursuite nulle, augmentez le gain proportionnel tant que le système est stable.

Résolution et unité / tour codeur :

La résolution et l'unité / tour codeur correspondent au nombre de points du codeur utilisé x 4 et à l'avance de l'axe pour un tour codeur. Si ces paramètres sont erronés, la cote indiquée sur l'écran ne correspondra pas à la cote réelle.

Erreur de poursuite maxi : indique l'écart maximum admis entre la position théorique instantanée et la position réelle au-delà duquel l'axe passe en défaut. **Une valeur trop basse fait passer le système en défaut de manière intempestive. Une valeur très grande peut être dangereuse pour la sécurité des personnes et de la machine car lors d'un blocage de l'axe, le passage en défaut ne sera pas immédiat.**

Fenêtre de position mini : Par rapport à la cote finale, c'est la tolérance admise à l'intérieur de laquelle l'axe considère qu'il a atteint sa position.

3- LOGICIEL MCB EX

3-1- Installation du logiciel MCB EX

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 :

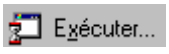
- PC Pentium® 75 ou plus
- RAM 16 Mo
- Disque dur (35 Mo disponibles)
- Microsoft® Windows™ 95 ou Microsoft® Windows™NT 4.0 (service pack 3)
- Lecteur de CD-ROM (4X)
- Ecran SVGA
- Souris ou autre périphérique de pointage

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 MCBEX

Le logiciel Motion Control Basic est fourni sous forme de disquette (à la demmande) ou de CD-ROM avec le système MCS. L'installation du logiciel se fait comme suit :

- Vérifier la configuration requise pour installer le logiciel
- Insérer la disquette ou 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 la disquette ou le CD-ROM.
- Sélectionner 

puis 

dans la boîte de dialogue « Parcourir ».
- Sélectionner 

dans la boîte de dialogue « Exécuter ».
- ⇨ Le programme d'installation du logiciel MCB 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 de la MCB 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 MCB EX 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\Mcb EX** ».

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

Relier le port SERIAL1 de la MCS32 EX sur COM1 ou COM2 du PC

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 et un 'c' sur l'afficheur STATUS de la MCS.

⇨ Ensuite, la programmation s'effectue et un 'P' apparaît sur l'afficheur STATUS de la MCS.

⇨ Lorsque la programmation est terminée, la MCS redémarre en indiquant 'E23' sur l'afficheur STATUS car aucun programme utilisateur n'est présent.

Recompiler les tâches du projet puis les transférer dans la MCS.

3-2- Architecture du logiciel MCB EX

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 de la MCS

Help : contient les fichiers d'aide pour la MCS32EX et MCBEX.

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

3-2-2- Contenu d'un projet

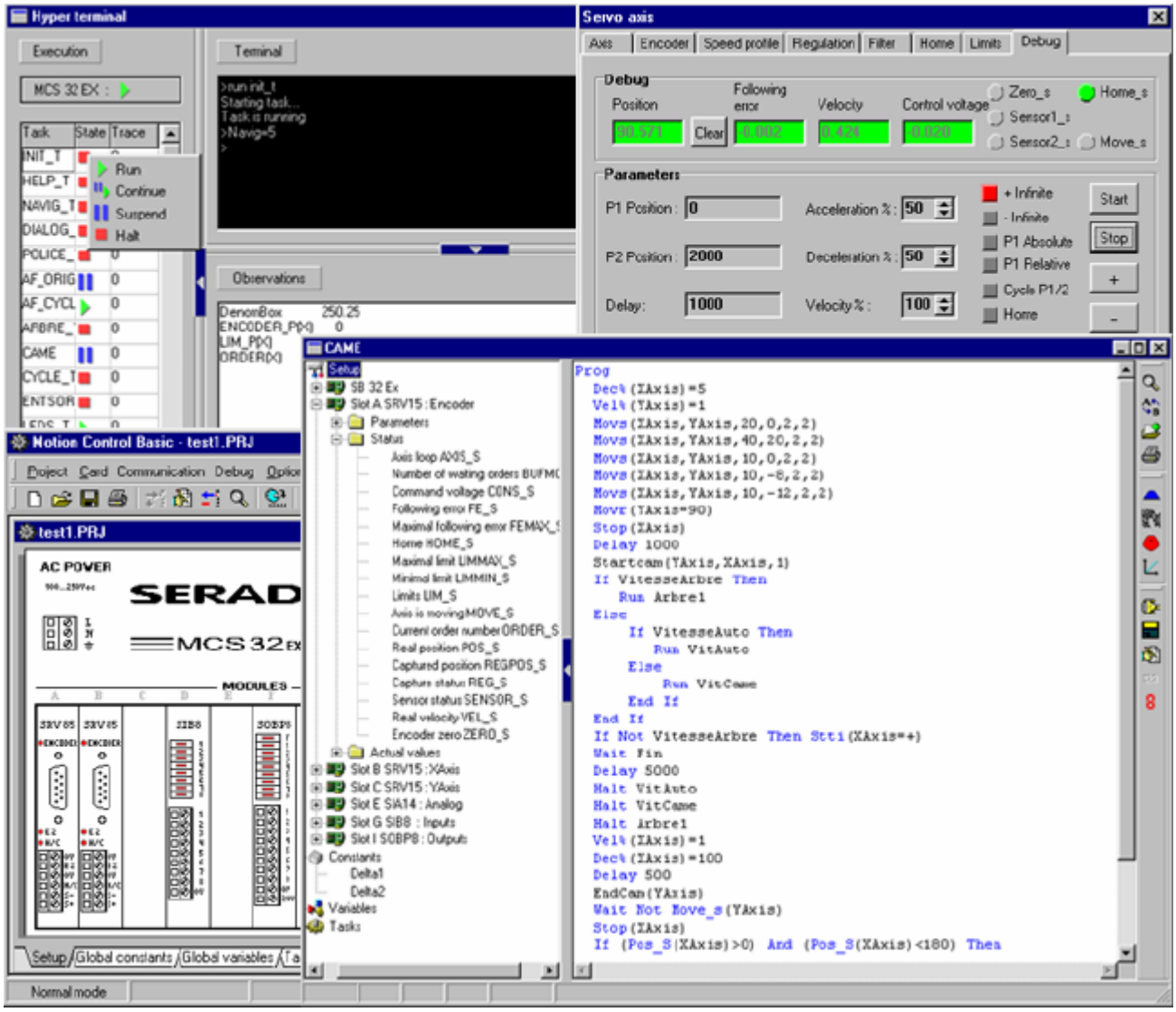
Le répertoire “project” est l’emplacement réservé par défaut pour stocker les projets utilisateurs. Chaque projet développé se décompose en un fichier “Nom Projet.prj” et un répertoire “Nom Projet.rep”. Ce répertoire comporte les fichiers suivants:

- 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 MCB

3-3- Présentation

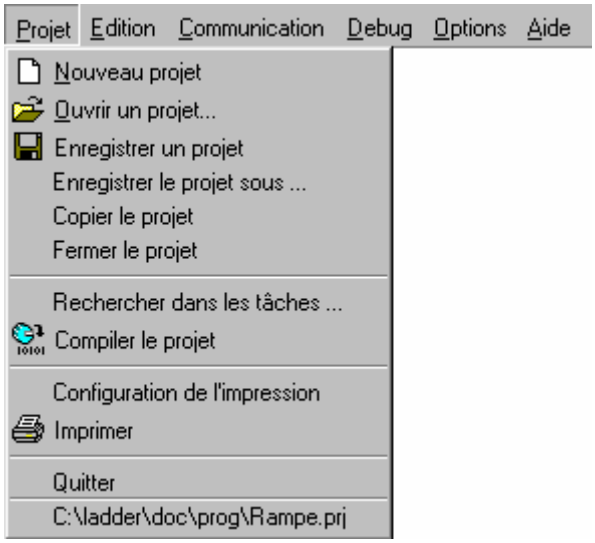
3-3-1- Ecran initial

Le logiciel MCB EX 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

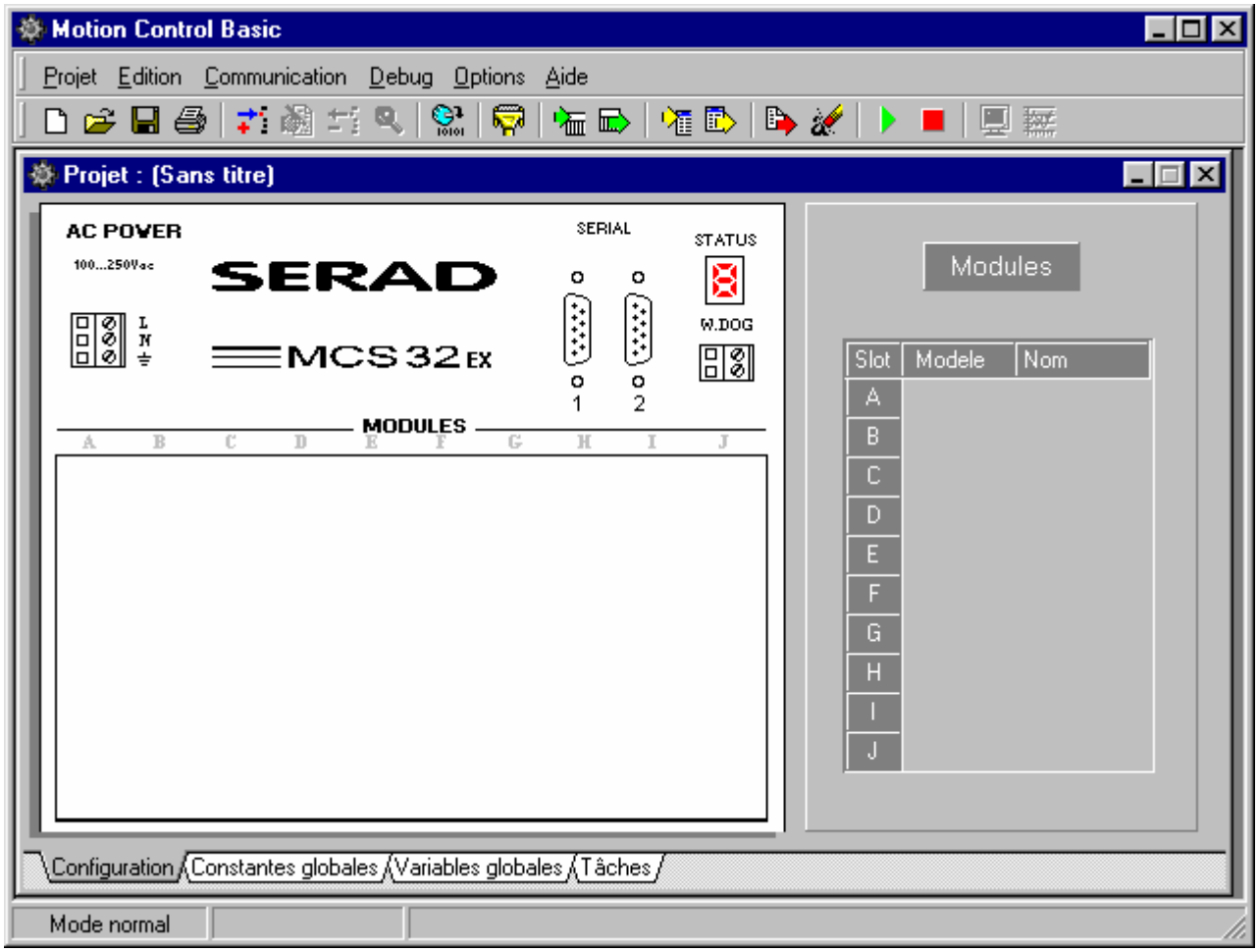
3-4-1- Menu Projet



Nouveau Projet

Icône: 

Action: Cette commande permet à l'utilisateur de définir un nouveau projet. Le dernier projet en cours se trouve alors fermé. 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

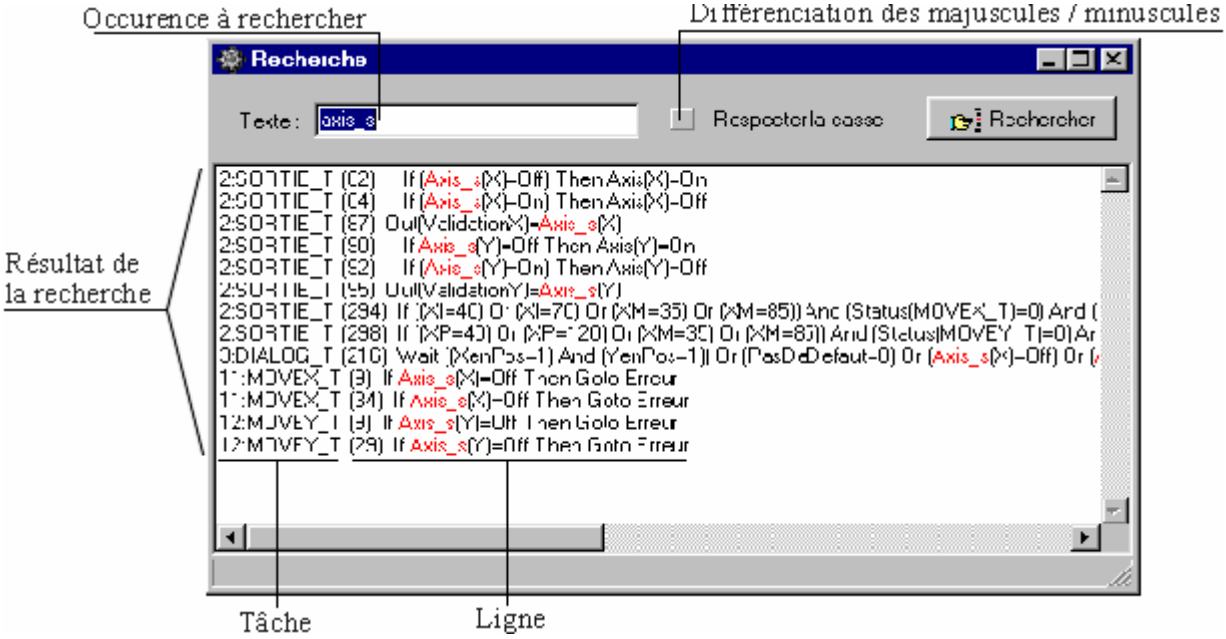
Action: Cette commande a les mêmes fonctionnalités que Enregistrer le projet sous… . L’intérêt de cette commande est qu’elle ne modifie pas les dates de création du projet.

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.



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.

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 de la MCS, 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 Edition

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: 

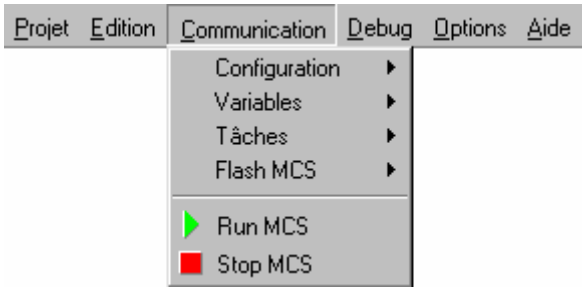
Action : Cette commande permet de supprimer une carte, une constante globale, une variable globale ou une tâche suivant l'onglet sélectionné.

Visualiser

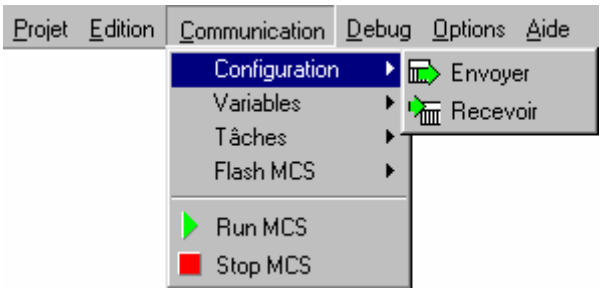
Icône: 

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



Envoyer la configuration

Icône: 

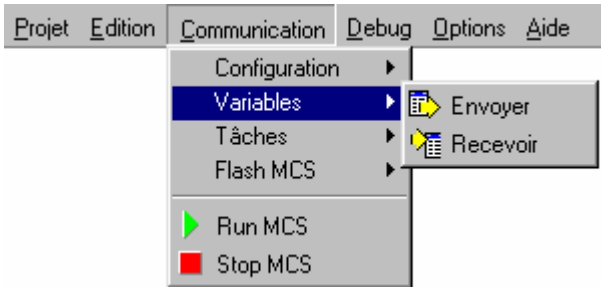
Action: L'envoi de la configuration permet d'initialiser dans la MCS les paramètres définis dans les écrans de configuration des cartes. Le transfert commence par le test de la configuration de la MCS par rapport à la configuration indiquée sur le PC. Si une différence est détectée le transfert est stoppé 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 la MCS 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: 

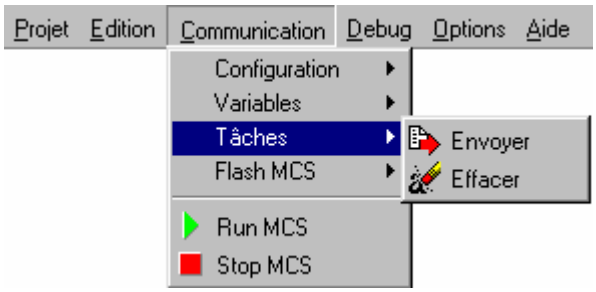
Action: L'envoi des variables sauvegardées permet d'initialiser dans la MCS, les valeurs affectées à ces variables. Ainsi il n'est pas nécessaire de les initialiser dans le programme. Le transfert commence par le test de la configuration de la MCS par rapport à la configuration indiquée sur le PC. Si 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 de la MCS 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: 

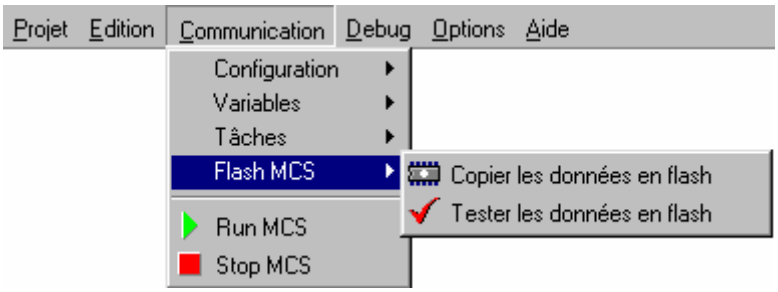
Action: Cette commande permet d'envoyer la totalité des tâches compilées dans la MCS. L'exécution des tâches est lancée automatiquement dès la fin du transfert. Le transfert commence par le test de la configuration de la MCS par rapport à la configuration indiquée sur le PC. Si une différence est détectée le transfert est stoppé et un message apparaît en indiquant le slot dont le contenu est incorrect. Le test de la configuration étant correct, on procède ensuite à l'effacement de la zone mémoire devant recevoir les nouvelles tâches. Pendant cette phase, le symbole "c" apparaît sur l'afficheur status de la MCS et une fenêtre indiquant l'évolution du transfert apparaît sur l'écran du PC.

Effacer les tâches

Icône: 

Action: Cette commande permet d'écraser les tâches présente dans la MCS. Suite à cette commande, la MCS passe en erreur 23.

Flash MCS



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 de la MCS, un calcul de checksum est effectué pour tester la cohérence des données en RAM. Si une erreur est détecée, la MCS 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, la MCS passe en erreur 20.

Tester les données en flash

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

Run MCS

Icône: 

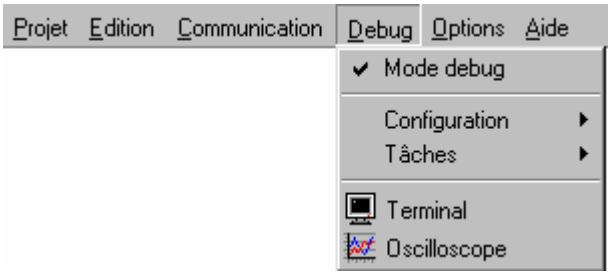
Action: Cette commande lance l'exécution des tâches de la MCS.

Stop MCS

Icône: 

Action: Cette commande permet d'arrêter l'exécution des tâches de la MCS. Le WatchDog passe à l'état OFF. Toutes les cartes servo passent en boucle ouverte (consigne analogique=0). L'instruction Security n'est pas prise en compte par la MCS.

3-4-4- Menu Debug



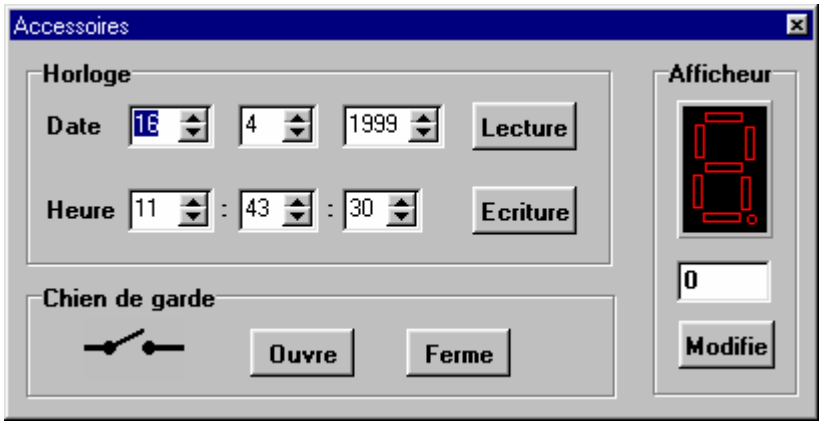
Mode Debug

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 de la MCS apparaît. Ces différents paramètres peuvent-être modifiés. Il faut aussi noter que pendant une exécution des tâches de la MCS, 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.



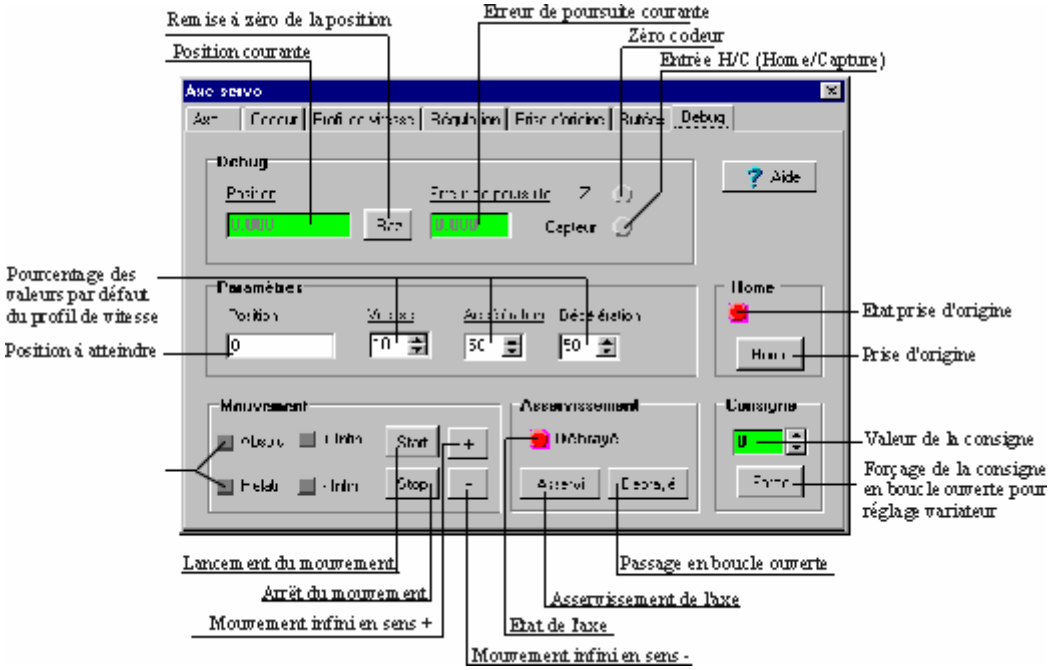
↳ La fenêtre de Debug des cartes d'entrées / sorties TOR (SIB8, SIH16, SIH 24, SOBP8, SOH16) permet de visualiser l'état des entrées ou des sorties de la carte par des leds. Un clique sur l'une des leds de visualisation permet de modifier l'état de l'entrée ou de la sortie.

↳ la fenêtre de Debug des cartes d'entrées / sorties analogiques (SIA14, SOA12) permet de visualiser la tension présente sur les entrées ou les sorties de la carte. Les champs peuvent-être modifié manuellement.

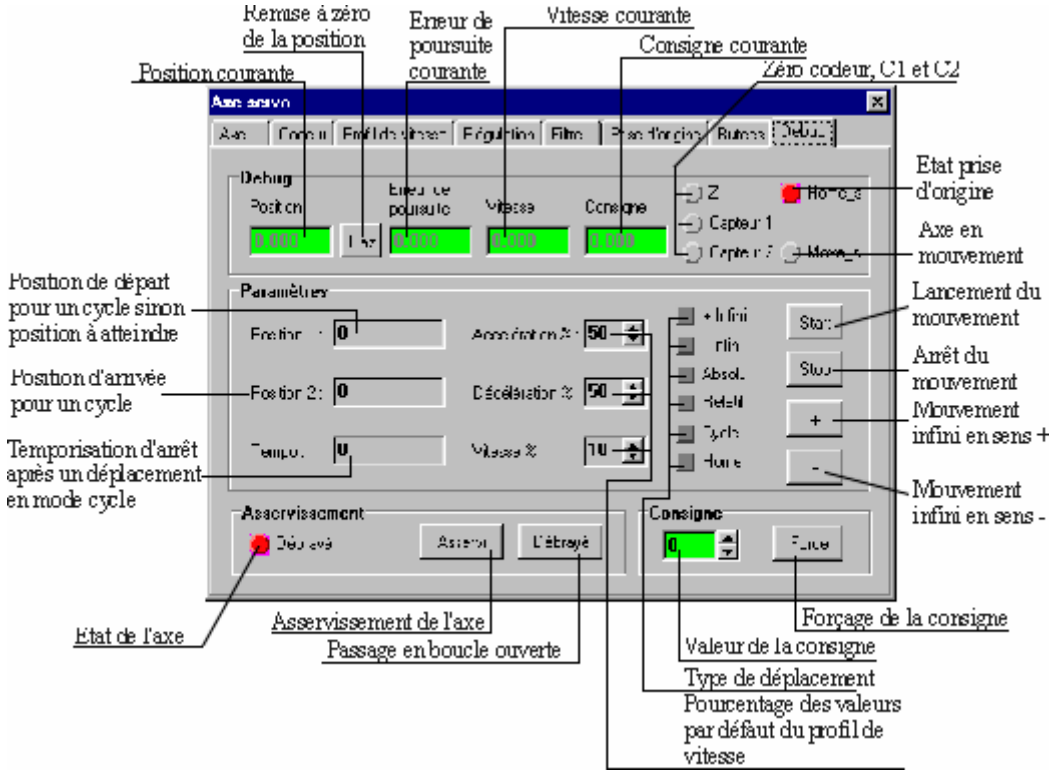
↳ la fenêtre de Debug des cartes codeurs (SCD22, SCD2224) permet de visualiser la position, l'état du zéro codeur et l'état du capteur afin de faciliter le contrôle du câblage. La position peut également être remise à zéro.

↳ la fenêtre de Debug des cartes codeurs SSI (SSI22) permet de visualiser la position et l'état du capteur afin de faciliter le contrôle du câblage.

↳ la fenêtre de Debug des cartes servo SRV15 et SRV15-24 permet de tester la boucle de positionnement de l'axe. Il est préférable de commencer par vérifier le comportement du moteur/variateur en forçant la consigne à une valeur comprise entre +10V et –10V (L'axe doit être en mode débrayé). On peut ensuite passer en mode asservi et régler les paramètres d'asservissement. Les modifications effectuées sur les différentes pages de configuration sont prise en compte par la CN lors du retour sur la page Debug. Si toutefois un paramètre s'avère erroné, l'écran contenant ce paramètre est affiché et le paramètre en question est sélectionné. Lors de l'utilisation du mode Debug, les paramètres affichés sur fond vert correspondent aux valeurs stockées dans la MCS. Si une modification est effectuée, elle est directement envoyée dans la MCS. Si l'on souhaite sauvegarder ces modifications, il faut faire une sauvegarde du projet avant de quitter le logiciel MCB EX.



↳ la fenêtre de Debug de la carte servo SRV85 permet de tester la boucle de positionnement de l'axe. Il est préférable de commencer par vérifier le comportement du moteur/variateur en forçant la consigne à une valeur comprise entre +10V et –10V (L'axe doit être en mode débrayé). On peut ensuite passer en mode asservi et régler les paramètres d'asservissement. Les modifications effectuées sur les différentes pages de configuration sont prise en compte par la CN lors du retour sur la page Debug. Si toutefois un paramètre s'avère erroné, l'écran contenant ce paramètre est affiché et le paramètre en question est sélectionné. Lors de l'utilisation du mode Debug, les paramètres affichés sur fond vert correspondent aux valeurs stockées dans la MCS. Si une modification est effectuée, elle est directement envoyée dans la MCS. Si l'on souhaite sauvegarder ces modifications, il faut faire une sauvegarde du projet avant de quitter le logiciel MCB EX.



↳ la fenêtre de Debug de la carte servo SSI15 permet de tester la boucle de positionnement de l'axe. Il est préférable de commencer par vérifier le comportement du moteur/variateur en forçant la consigne à une valeur comprise entre +10V et –10V (L'axe doit être en mode débrayé). On peut commencer à régler les paramètres d'asservissement. Il est nécessaire de régler les paramètres du codeur avant d'activer le mode manuel car ensuit ils ne peuvent plus être modifié. Les modifications effectuées sur les différentes pages de configuration sont prises en compte par la CN lors du retour sur la page Debug. Si toutefois un paramètre s'avère erroné, l'écran contenant ce paramètre est affiché et le paramètre en question est sélectionné. Lors de l'utilisation du mode Debug, les paramètres affichés sur fond vert correspondent aux valeurs stockées dans la MCS. Si une modification est effectuée, elle est directement envoyée dans la MCS. Si l'on souhaite sauvegarder ces modifications, il faut faire une sauvegarde du projet avant de quitter le logiciel MCB EX. La fenêtre de Debug est la même que pour les cartes SRV15 et SRV15-24 avec l'absence de l'état du zéro codeur.

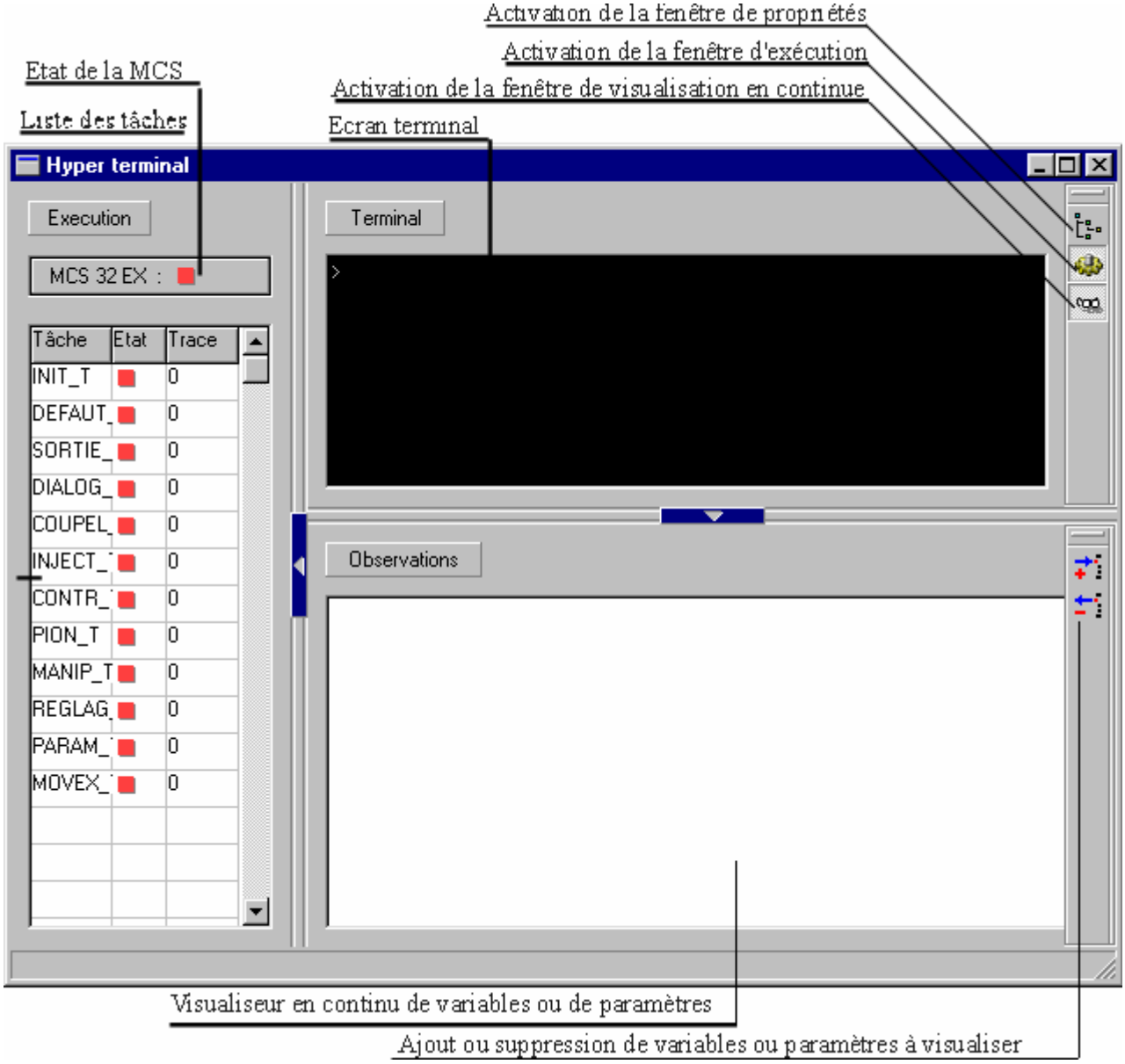
Tâches

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.

Terminal

Icône:

Action: Cette commande ouvre l'hyper terminal. Cet outil d'aide à la mise en œuvre permet d'interroger l'état de la MCS, 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 la MCS. 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 la MCS
- EXIT : ferme le terminal

Pour faciliter l'édition des variables ou paramètres, un éditeur des propriétés de la configuration de la MCS 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 de la MCS. La variable ou le paramètre doit être choisi parmi les cartes, variables globales ou tâches de la configuration.

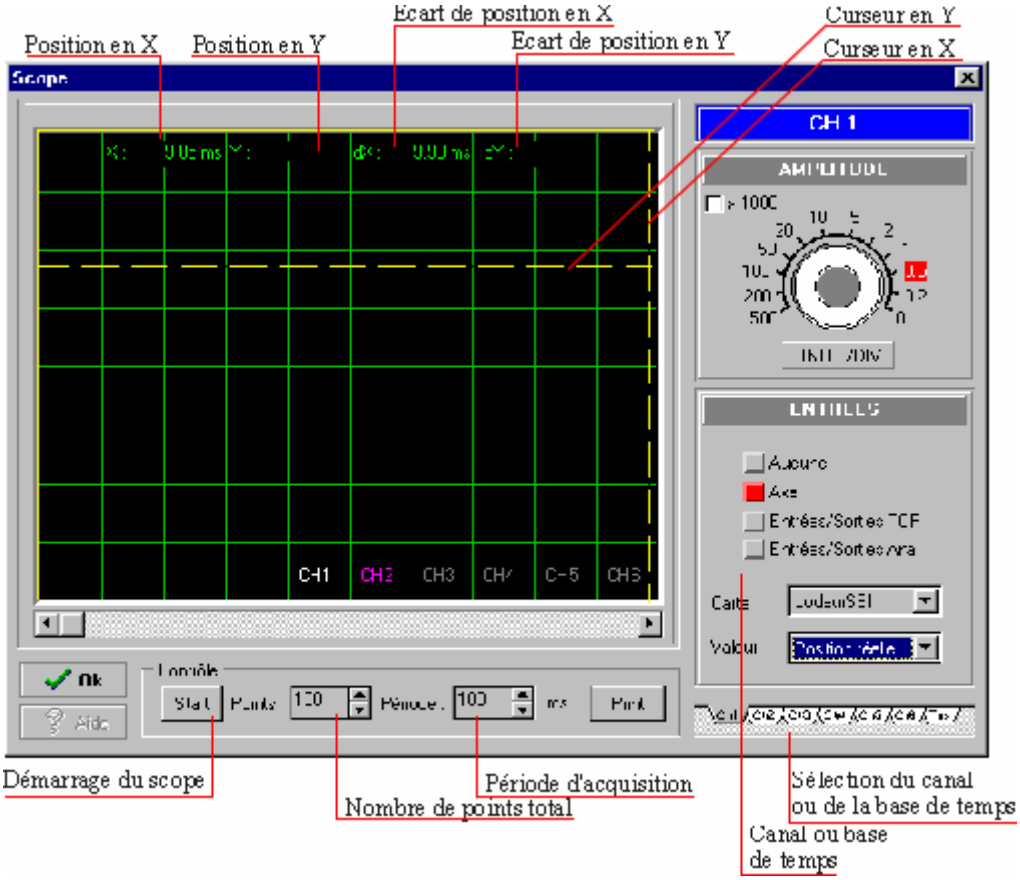
⇨ La fenêtre « status » du terminal permet la visualisation de l'état de la MCS et des tâches du projet. La MCS peut-être pilotée à distance en cliquant sur l'icône lecture ou stop qui est affiché. Le clique provoque le basculement d'un état à l'autre. De même les tâches peuvent être pilotées à distance, 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 la MCS.

Oscilloscope

Icône: 

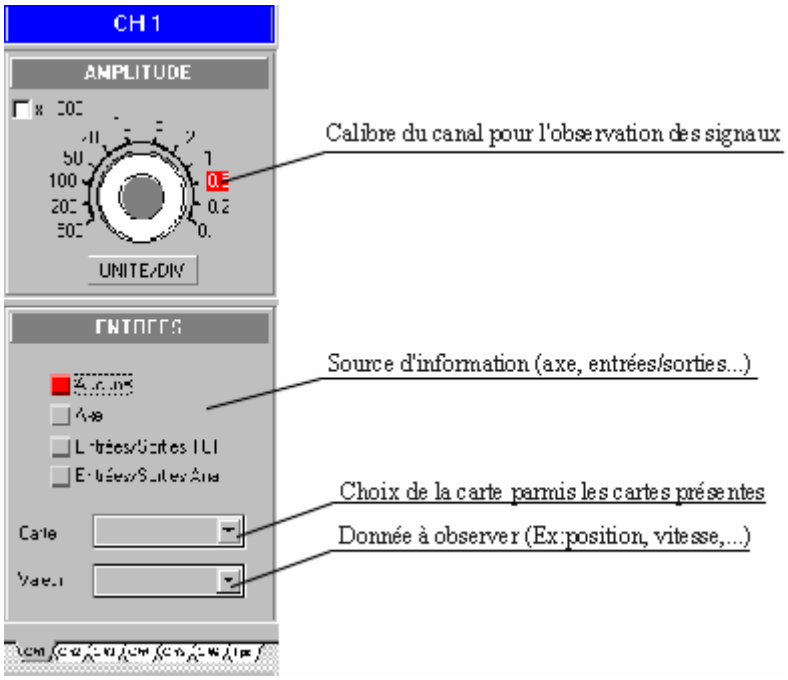
Action: Cette commande ouvre l'oscilloscope. Cet outil d'aide à la mise en œuvre permet d'enregistrer et de visualiser toutes les informations de cartes d'axes ou d'entrées / sorties. Il est capable d'enregistrer jusqu'à six variables différentes.

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.

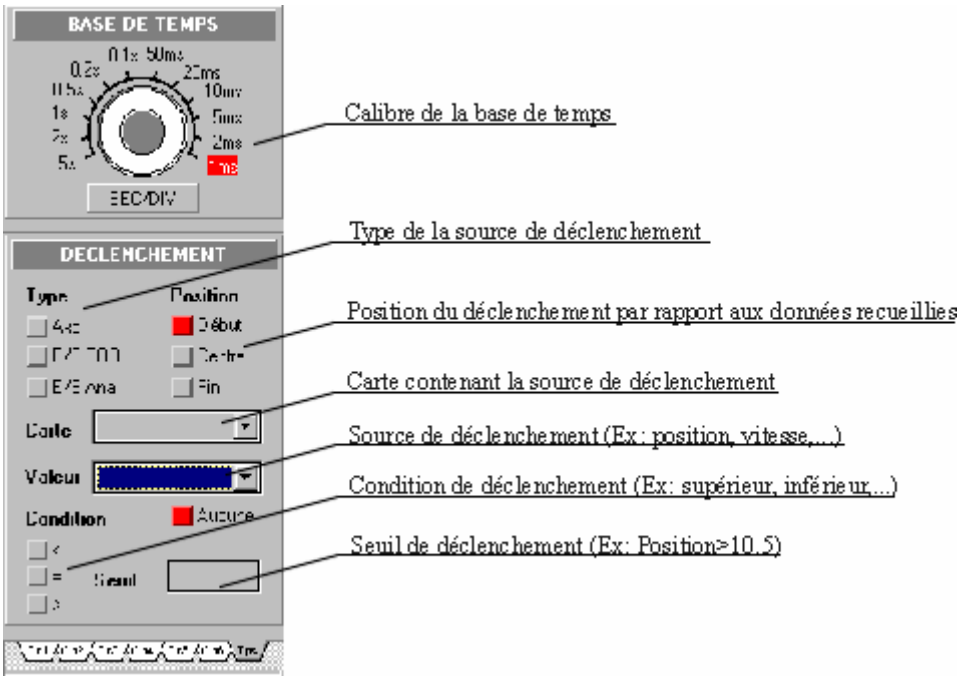


↳ La zone de configuration de l'acquisition permet de spécifier le nombre d'échantillon pendant la période d'acquisition. Elle permet aussi de lancer l'acquisition et d l'imprimer.

↳ 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.



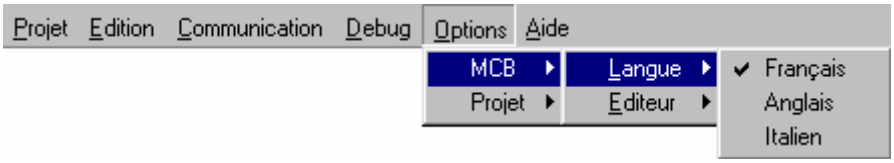
↳ La zone de visualisation affiche les six voies prédéfinies. En double-cliquant sur cette zone, la zone se place en plein écran. Il permet de plus de donner des informations en X et en Y (haut de l'écran) de la position du curseur. On peut aussi définir des références en X et en Y. Pour les définir, 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 Option

MCB

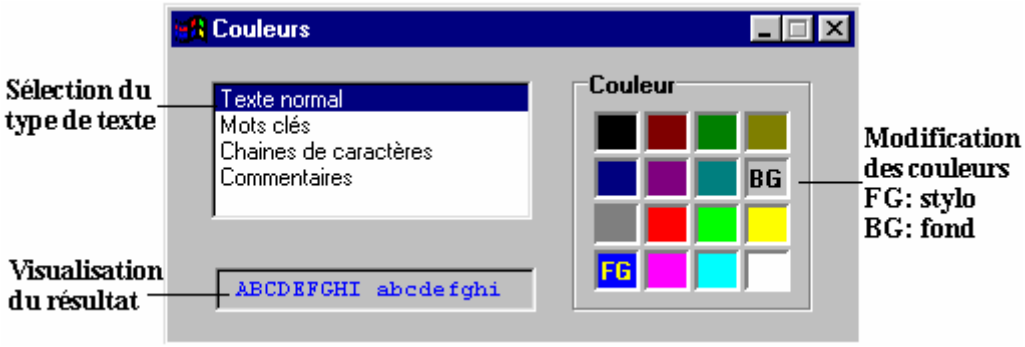
Langue



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

Editeur

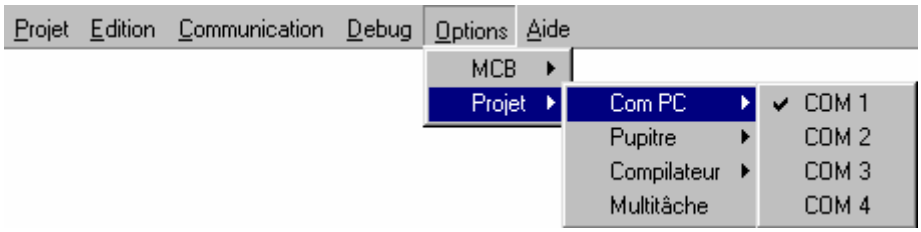
Ce sous-menu permet de personnaliser la couleur et le fond des textes, mots clés… de l'éditeur de tâche.



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.

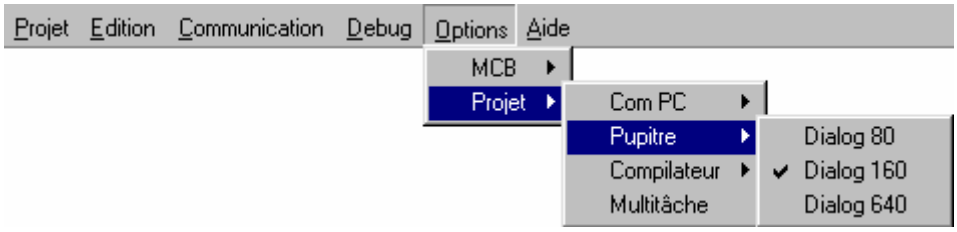
Projet

Com PC



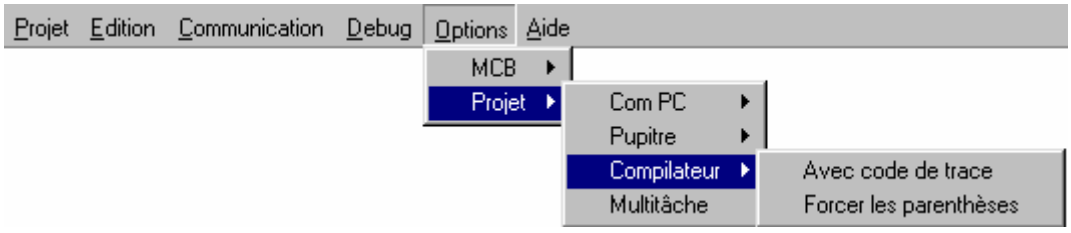
Ce sous-menu permet de sélectionner le port de communication du PC qui sera en liaison avec la MCS

Pupitre



Ce sous-menu est utilisé lorsque la MCS doit gérer un pupitre opérateur SERAD. Il permet de choisir le type de pupitre. Dans, l'éditeur de tâche, la boîte à outil pupitre correspondra au type choisi.

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.

3-4-6- Menu Aide

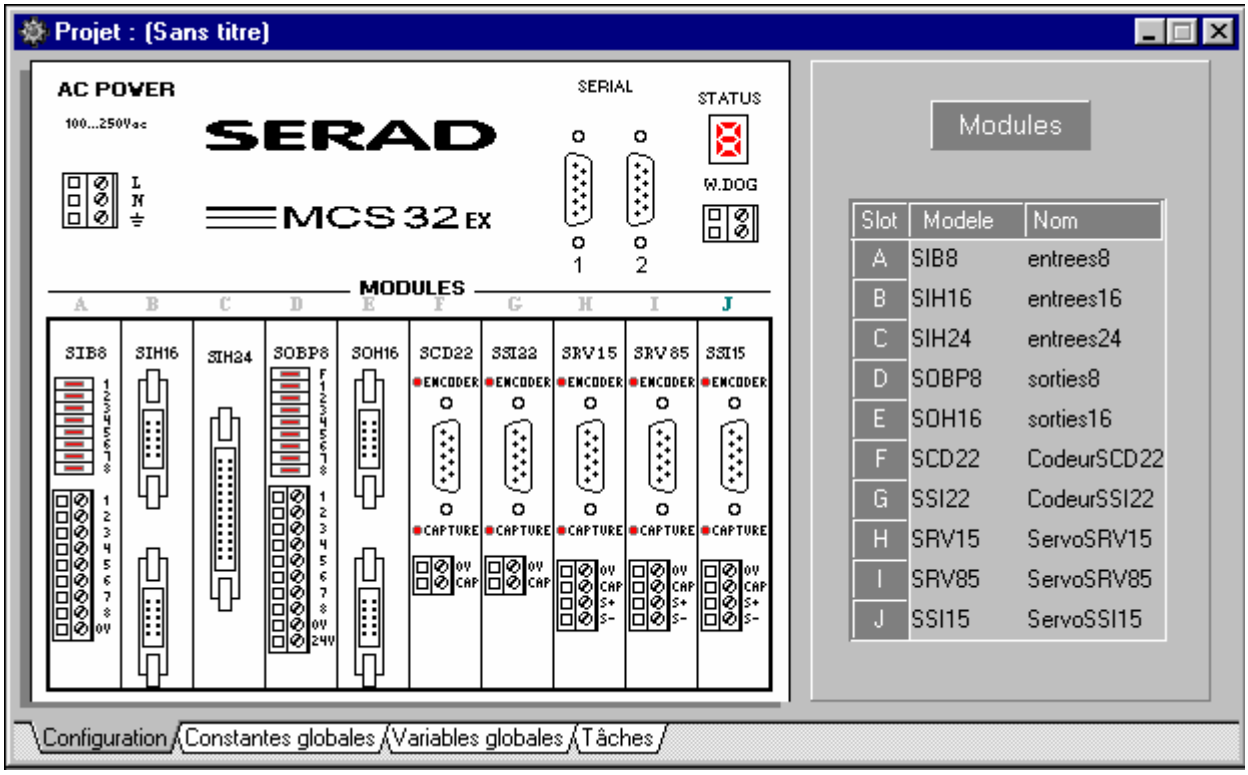
Assistant

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 Configuration



La fenêtre configuration se présente sous la forme de deux zones distinctes. La première située sur la gauche visualise la face avant de la MCS. C'est sur celle-ci que l'on vient réaliser la configuration de la MCS. Parmi les éléments paramétrable, on trouve l'horodateur, la SERIAL1, la SERIAL2 et les slots A, B, C, D, E, F, G, H, I, J. La deuxième zone intitulée "Modules" récapitule l'affectation des 10 slots (Nom et type de module). Sur chaque slot (A, B, C, D, E, F, G, H, I, J) de la MCS on peut ajouter, modifier, supprimer ou visualiser une carte. Les commandes d'ajout, de modification ou de suppression nécessitent une recompilation du projet et l'envoi de la configuration et des tâches dans la MCS.

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

- Par le menu Edition et en ayant sélectionné au préalable le slot d'insertion
- En double cliquant sur le slot libre désiré
- Un clique sur le bouton droit ouvre un menu qui comprend la commande « Ajouter » (après avoir sélectionné l'emplacement d'ajout).

La commande modifier (une carte) permet de modifier les caractéristiques de la cartes et s'obtient des façons suivantes :

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

La commande supprimer (une carte) permet d'enlever une carte d'un slot. Elle s'obtient des façons suivantes :

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

La commande visualiser (une carte) permet de configurer les paramètres d'une carte. Elle peut-être obtenue de trois manières différentes :

↳ Par le menu Edition et en ayant sélectionné au préalable la carte à configurer.

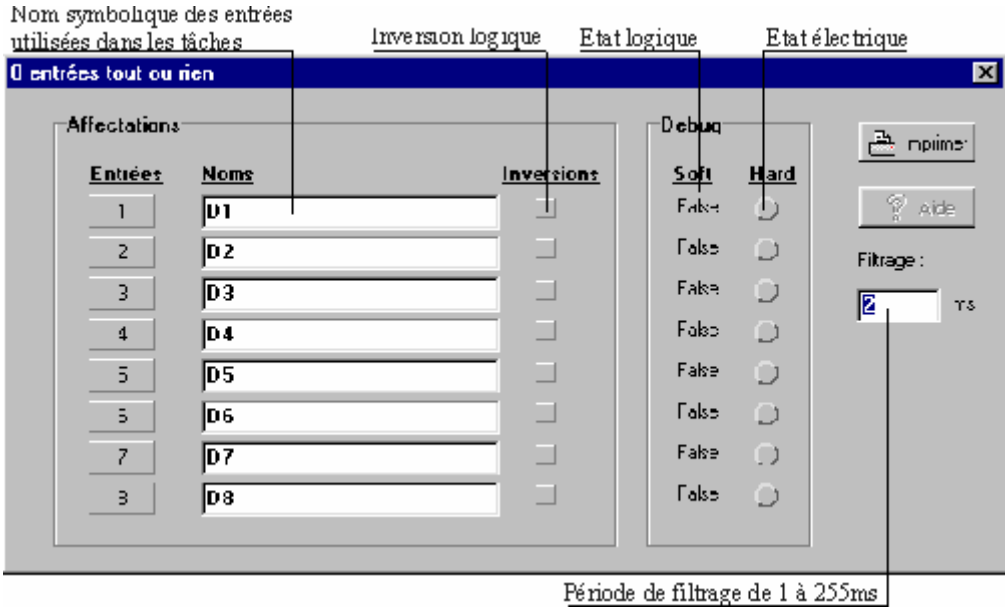
↳ En double cliquant sur le carte désirée

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

Paramétrage du SERIAL1 et du SERIAL2

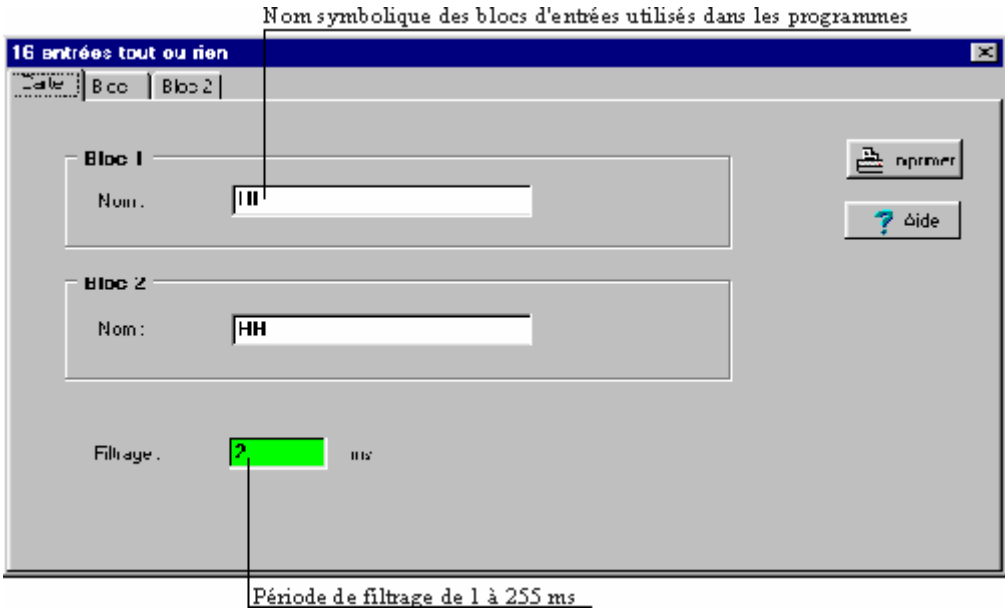
Pour paramétrer l'un des ports série, il faut double cliquer avec le bouton gauche de la souris sur celui qui doit être modifié. Une boîte de dialogue réunissant les informations de configuration de celui-ci apparaît. Le port série 1 est principalement dédié à la communication avec un PC et donc sa boîte de dialogue est munie d'une information supplémentaire concernant le port du PC auquel il est connecté. Pour une bonne communication, les différentes informations doivent coïncider entre le PC ou un terminal et la MCS.

Paramétrage du module 8 entrées TOR : SIB8



Paramétrage du module 16 entrées TOR : SIH16

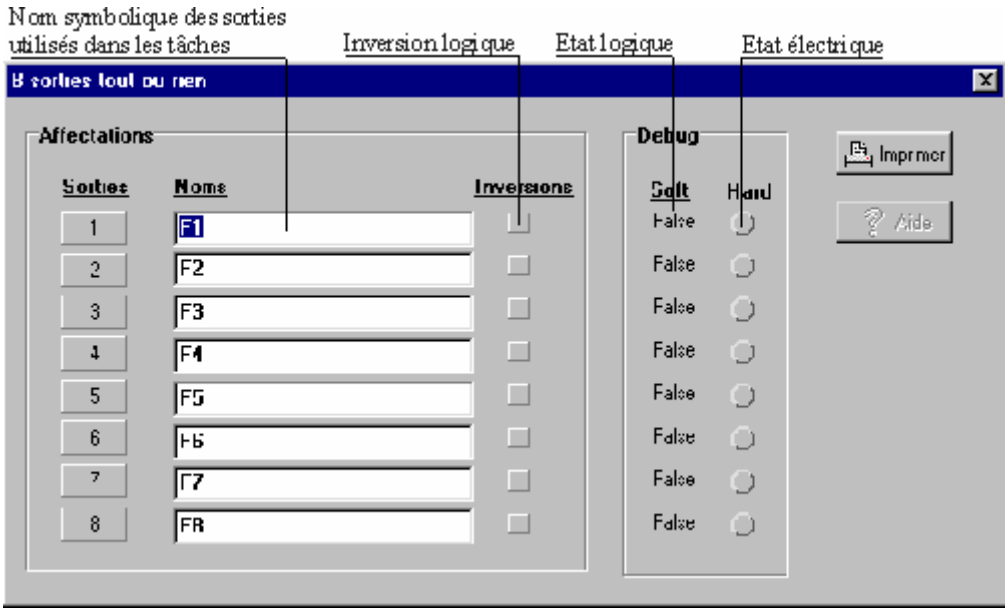
Pour les onglets bloc1 et bloc2, il faut se référer à la configuration de la carte SIB8.



Paramétrage du module 24 entrées TOR : SIH24

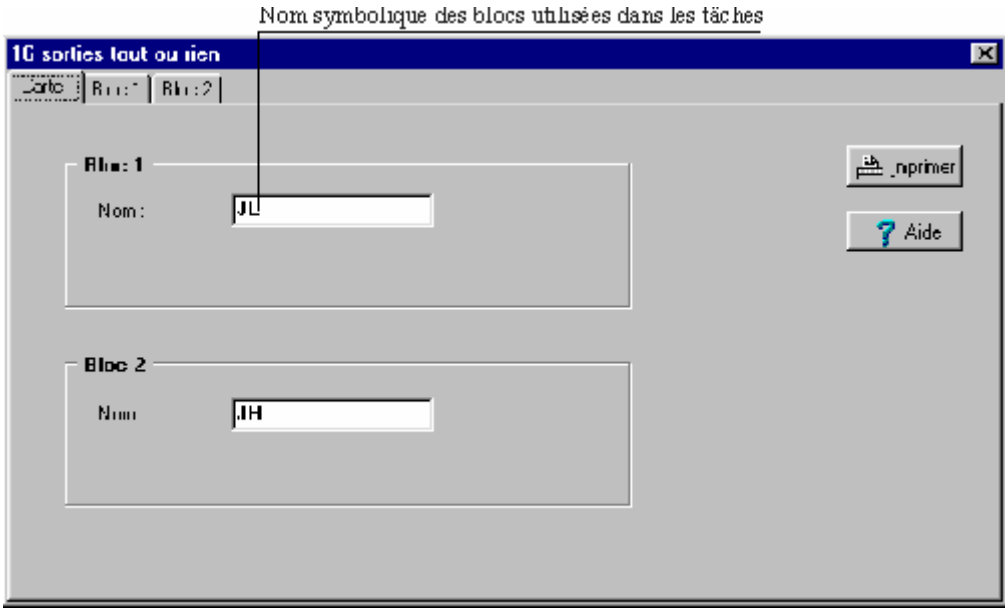
Pour les onglets bloc1, bloc2 et bloc3, il faut se référer à la configuration de la carte SIB8. Pour l'onglet carte, il faut se référer à la configuration du même onglet de la carte SIH16.

Paramétrage du module 8 sorties TOR : SOBP8

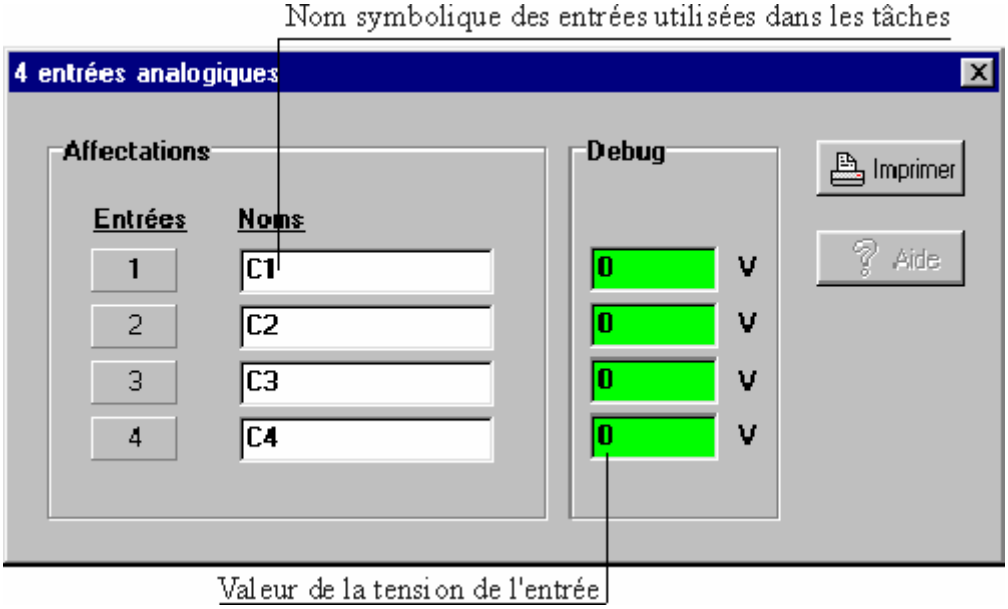


Paramétrage du module 16 sorties TOR : SOH16

Pour les onglets bloc1 et bloc2, il faut se référer à la configuration de la carte SOBP8.



Paramétrage du module 4 entrées analogiques : SIA14



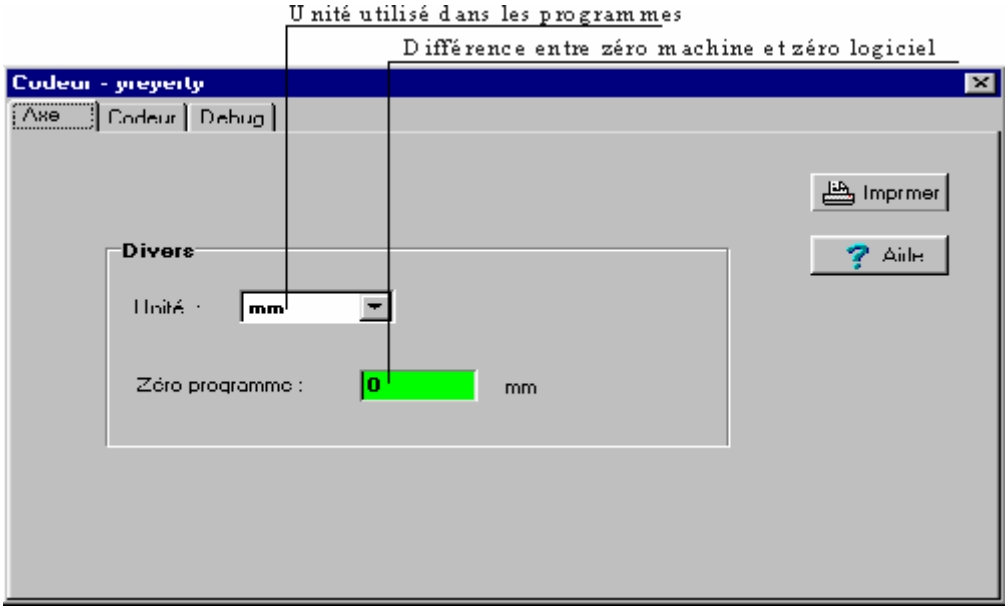
Paramétrage du module 2 sorties analogiques : SOA12

La configuration des sorties analogiques se fait de la même façon que la carte SIA14.

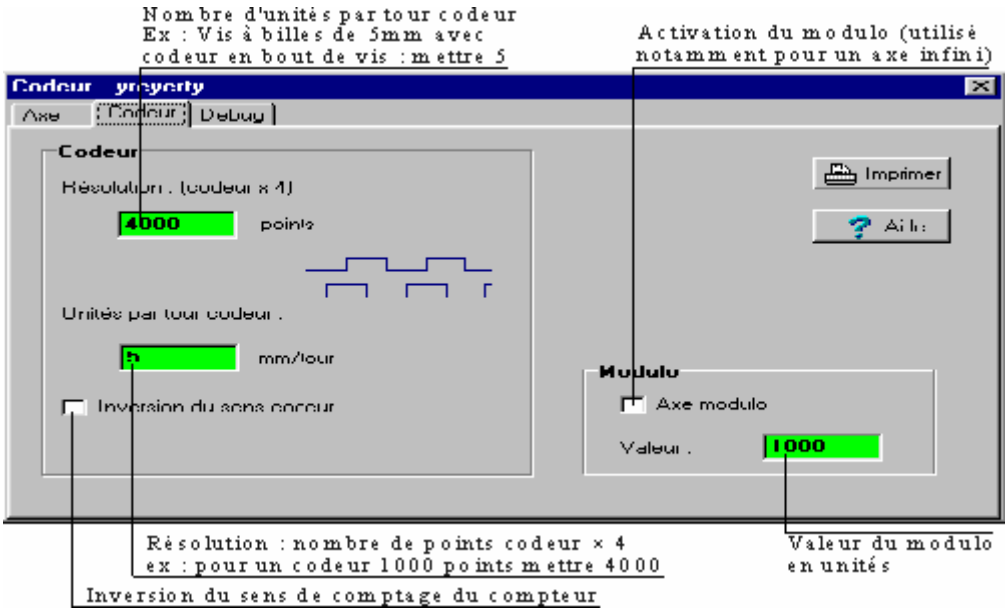
Paramétrage du module codeur : SCD22

Configuration de l'axe :

L'unité est utilisée pour définir les valeurs de distance. Si vous choisissez « mm », toutes les valeurs numériques écrites dans les programmes seront exprimées en « mm ». Le paramètre zéro programme spécifie la distance entre le zéro machine trouvé lors de la prise d'origine et le zéro que l'on désire utiliser dans les programmes.



Configuration du codeur :

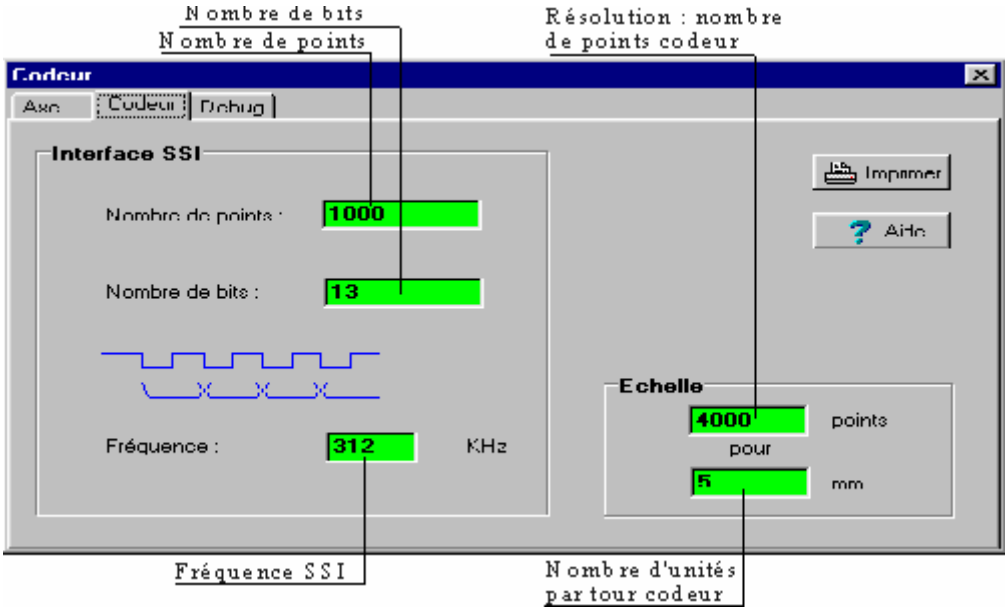


L'explication de l'onglet debug est faite avec le menu debug.

Paramétrage du module codeur : SSI22

Pour l'onglet Axe se reporter au module codeur SCD22.

Configuration du codeur :

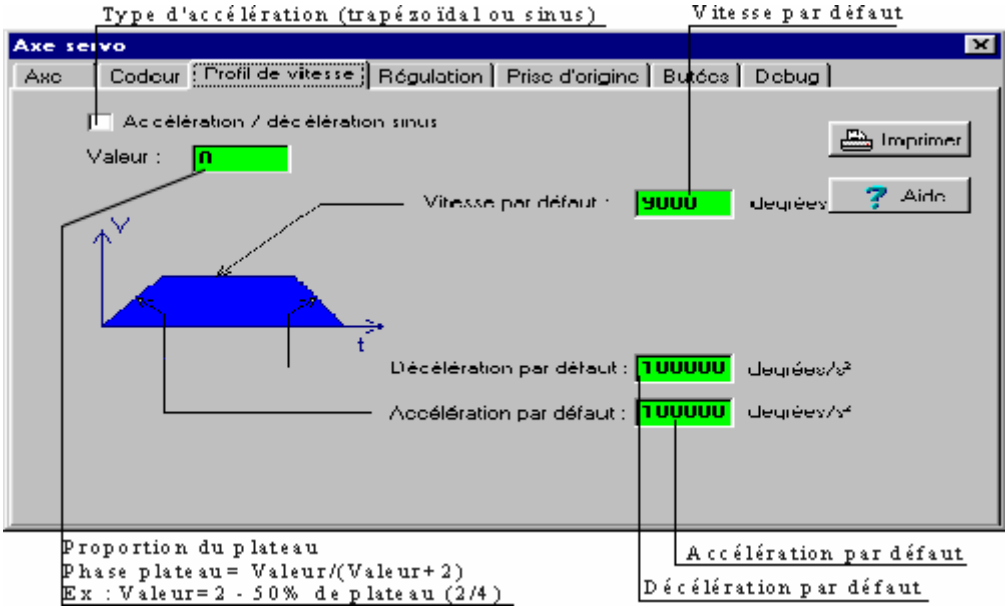


L'explication de l'onglet debug est faite avec le menu debug.

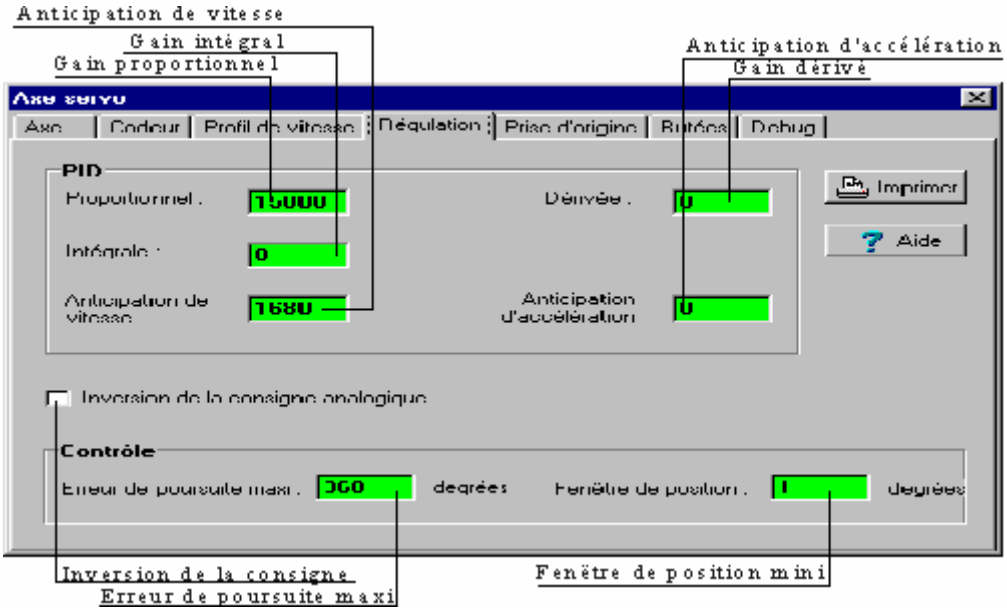
Paramétrage du module servo : SRV15, SRV15-24

La boîte de dialogue de configuration de ce module est composée de sept onglets. Pour la configuration des deux premiers (Axe et Codeur), il faut se référer aux onglets de la configuration du module codeur SCD22.

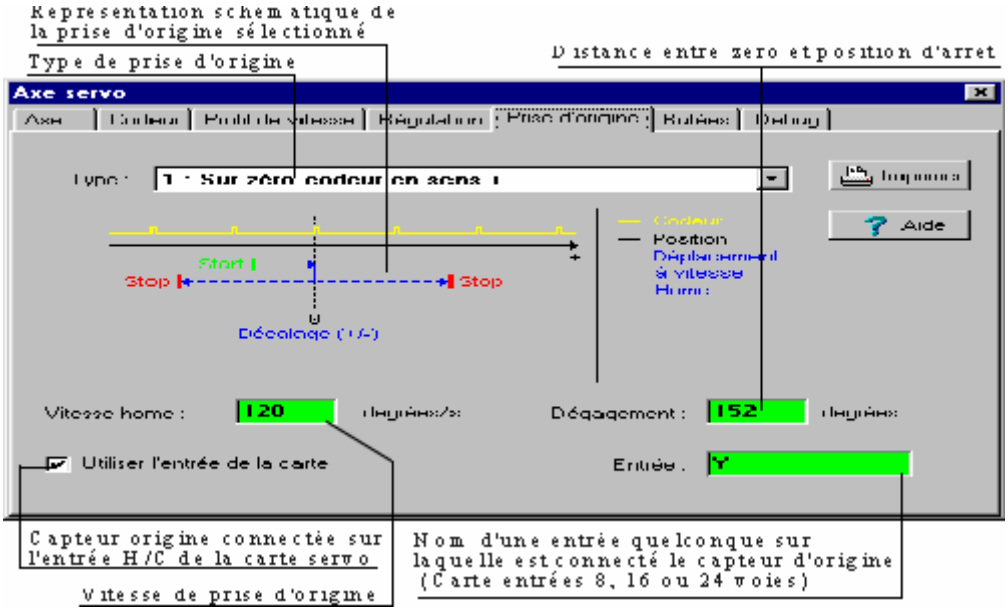
Configuration du profil de vitesse :



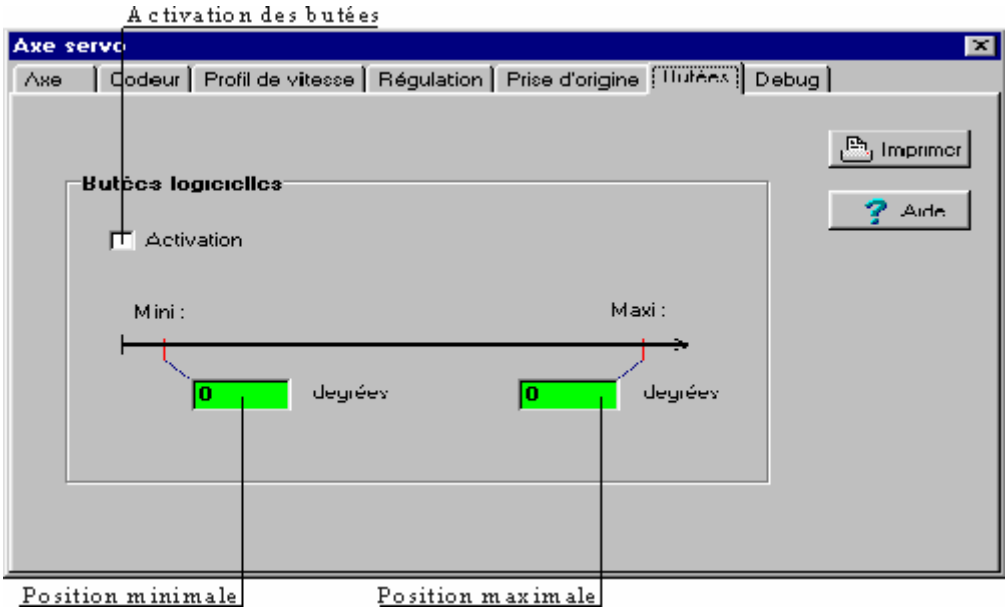
Configuration de la régulation :



Configuration de la prise d'origine :



Configuration des butées logicielles :

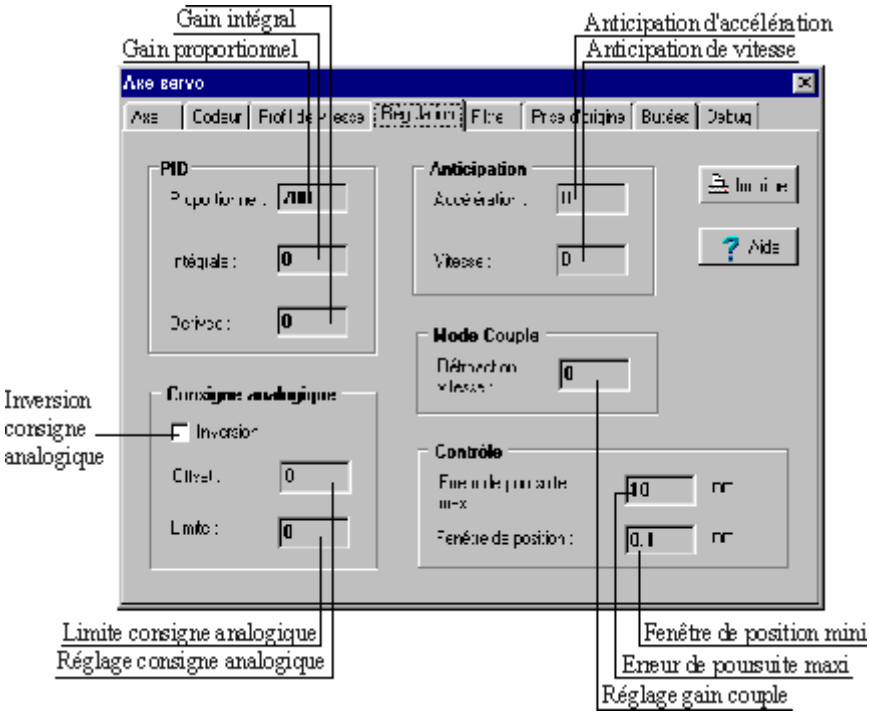


L'explication de l'onglet debug est faite avec le menu debug.

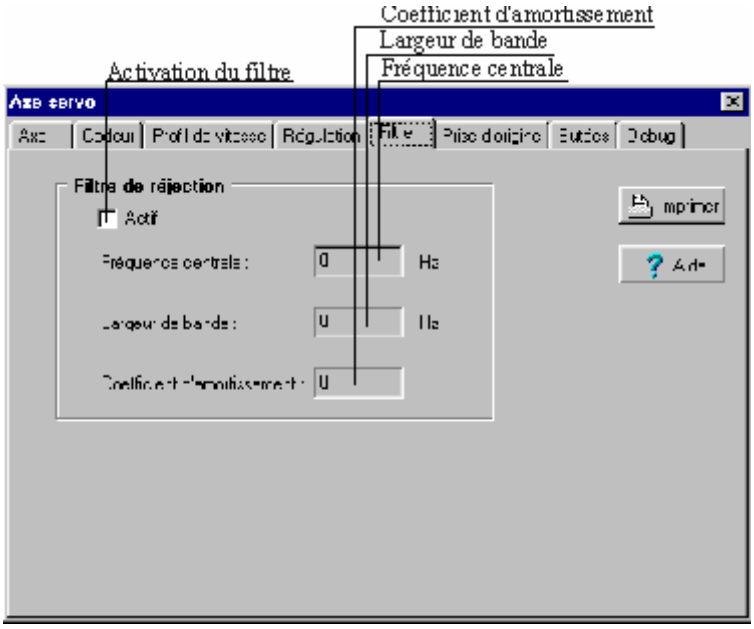
Paramétrage du module servo : SRV85

Pour la configuration des onglets Axe et Codeur, il faut se référer aux onglets de la configuration du module codeur SCD22. Pour les onglets Prise d'origine, Profil de vitesse et butées logicielles, il faut se référer aux onglets de la configuration des modules servo SRV15 et SRV15-24.

Configuration de la régulation :



Configuration du filtre :



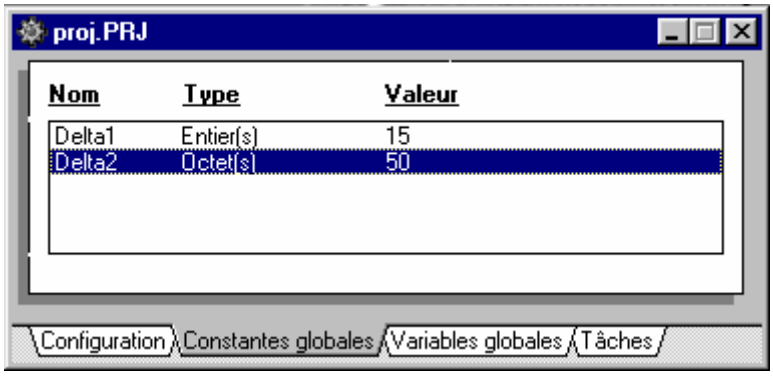
L'explication de l'onglet debug est faite avec le menu debug.

Paramétrage du module servo : SSI15

Pour la configuration des onglets Axe, Régulation, Profil de vitesse et Butées logicielles, il faut se reporter au paramétrage des modules servo SRV15, SRV15-24 et SRV85. La configuration de l'onglet Codeur se fait suivant celle declarée pour le module codeur SSI22. L'onglet «mode manuel» est décrit avec les outils d'aide à la mise en œuvre.

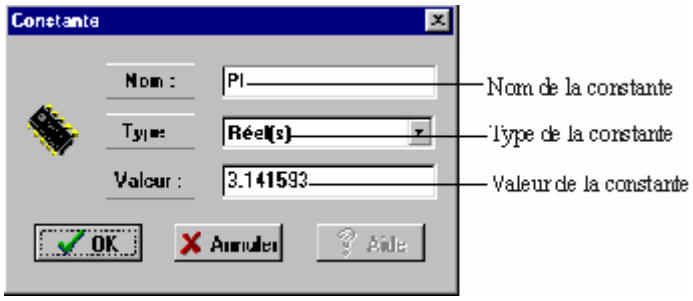
L'explication de l'onglet debug est faite avec le menu debug.

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 la MCS.

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 Edition

↳ 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 Edition et en ayant sélectionné au préalable la constante à modifier

↳ En double cliquant sur la constante à modifier.

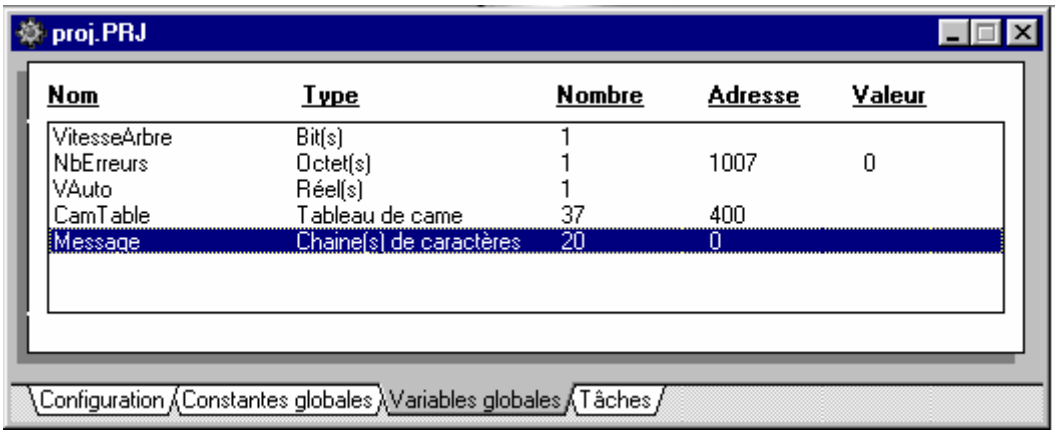
↳ Un clique sur le bouton droit ouvre un menu qui comprend la commande « Modifier » (après avoir sélectionné la constante à modifier).

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

↳ Par le menu Edition 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).

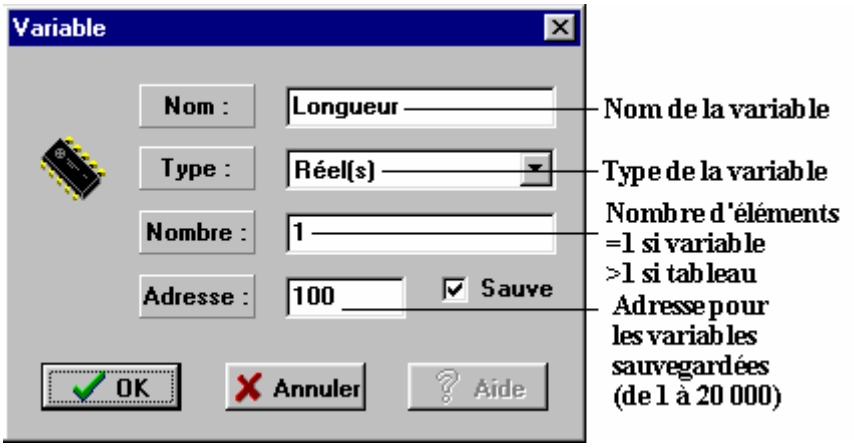
3-4-9- Onglet Variables globales



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 la MCS.

Dans le cas d'une variable sauvegardée, contenant une valeur, il faut également envoyer les variables dans la MCS.

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 Edition
- 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 Edition 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 :

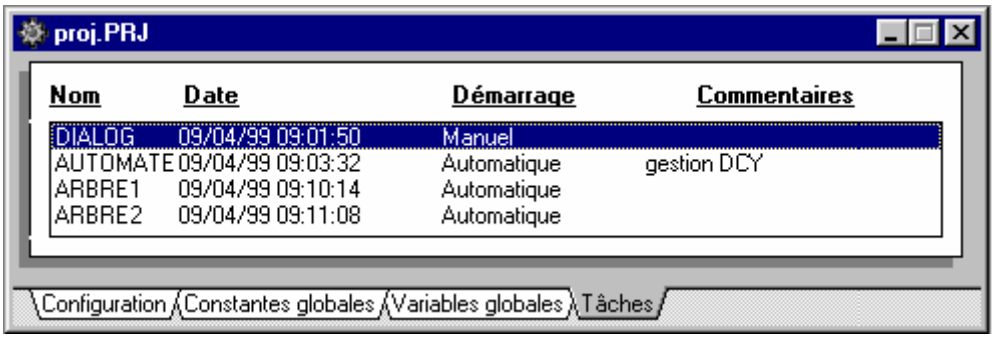
- Par le menu Edition et en ayant sélectionné au préalable la variable à supprimer
- Un clique sur le bouton droit ouvre un menu qui comprend la commande « Supprimer » (après avoir sélectionné la variable à supprimer).

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 Edition 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).

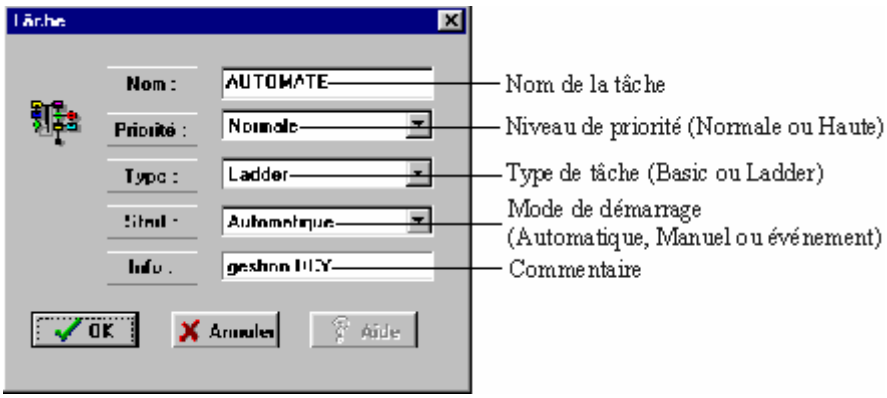
Lors d'une visualisation d'une variable de type tableau de came, on accéde à l'éditeur de came. Ce dernier permet de mettre en oeuvre la came désirée.

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 la MCS.

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 Edition
- 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 Edition 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 Edition 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 :

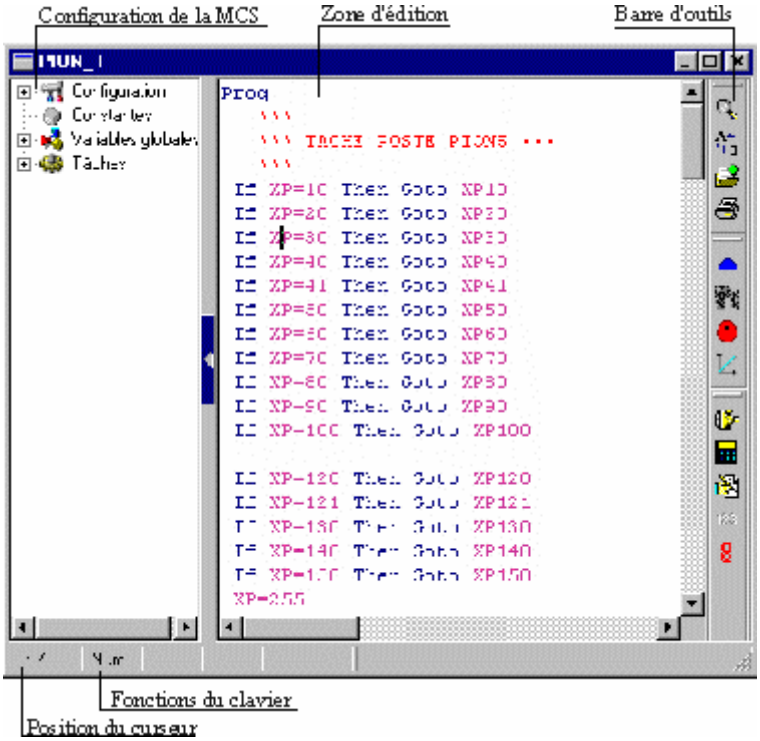
↳ Par le menu Edition et en ayant sélectionné au préalable la tâche à supprimer

↳ 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 de la MCS. La configuration de la MCS 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'importation permet d'adjoindre des fichiers d'une forme provenant d'autocad (format DXF).

↳ La commande d'impression

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

⇨ Les outils liés aux instructions du contrôle de mouvement

↳ L'outil d'édition de trajectoire  permet de définir les paramètres de l'instruction basic TRAJ.

↳ L'outil d'édition d'arbre électrique  permet de définir les paramètres de l'instruction basic GEARBOX.

↳ L'outil d'édition de came  permet de définir les paramètres de l'instruction basic CAM.

↳ L'outil d'édition de mouvement  permet de définir les paramètres de l'instruction basic MOVS.

⇨ Les outils liés aux instructions de communications

↳ La commande d'ouverture de port de communication  permet de définir les paramètres de l'instruction basic OPEN.

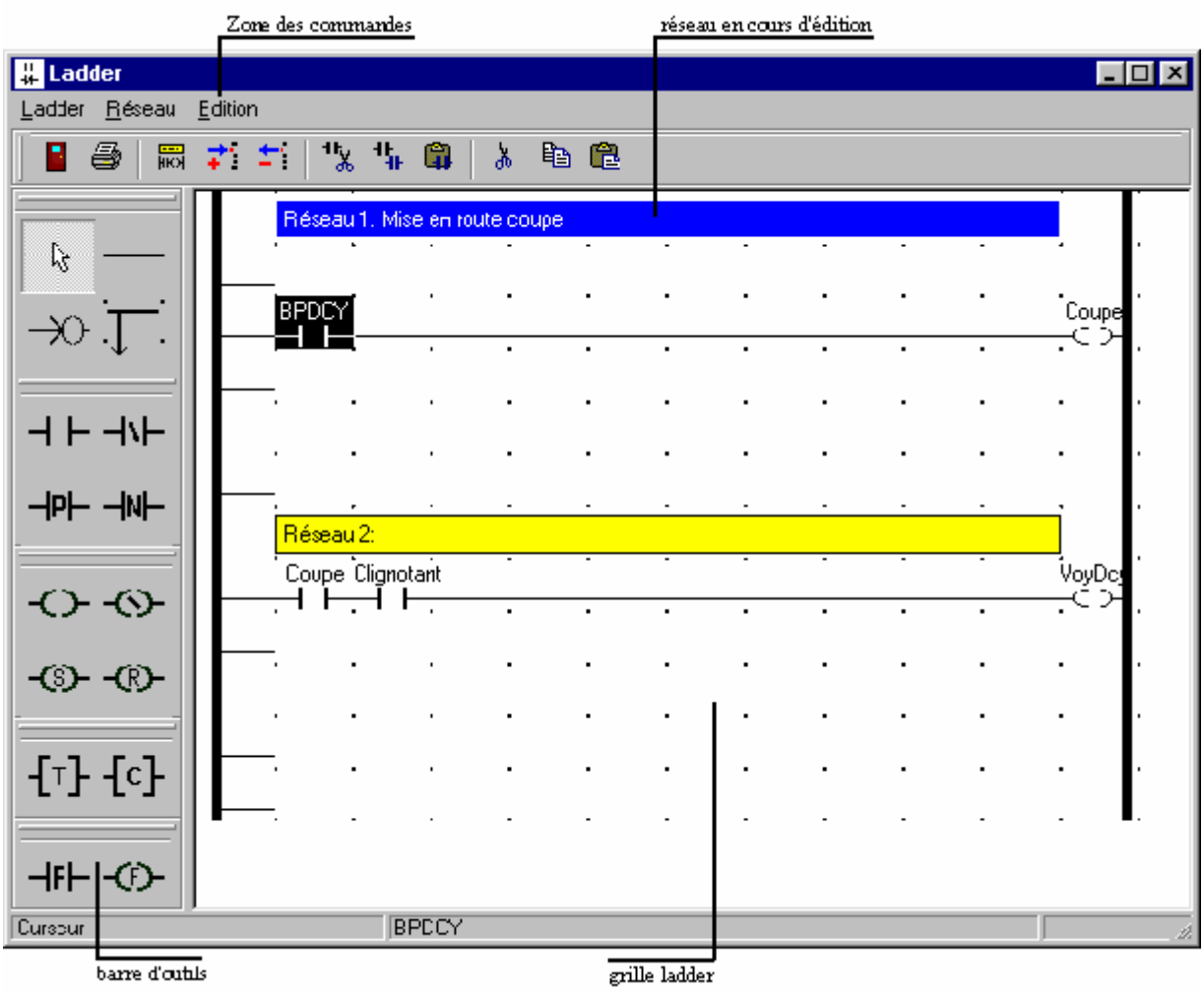
↳ L'outil Terminal  permet de définir tout le code concernant le texte à afficher sur un pupitre. La boîte de dialogue est définit par un écran avec le pupitre sélectionné dans le menu Option et quelques boutons de commande. Pour générer du code, il suffit d'inscrire sur l'écran sa configuration d'affichage 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.

↳ La commande de l'afficheur  permet de définir le paamètre de l'instruction DISPLAY.

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 de la MCS 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.

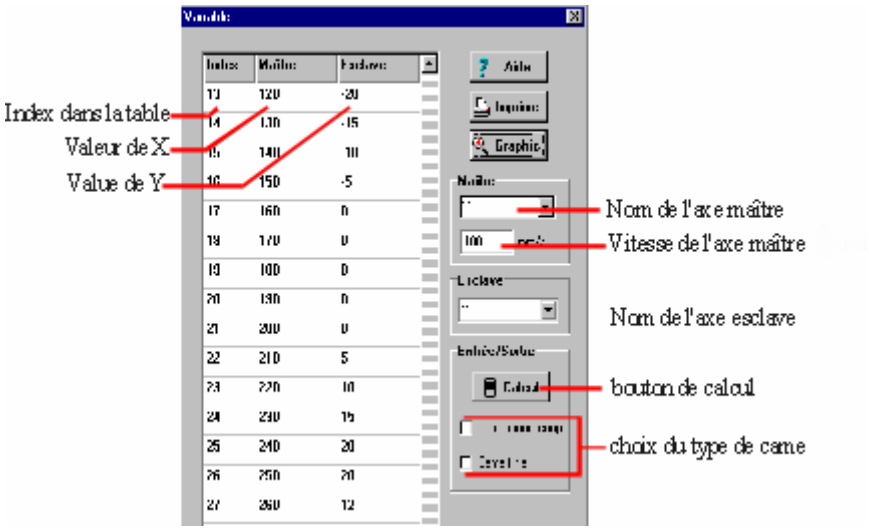
⇨ Les commandes pour couper  , copier  et coller un réseau  . Le réseau couper ou copier est le réseau détenant la focalisation. Le réseau coller est inséré avant le réseau détenant la focalisation.

⇨ 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.

3-5-3- Editeur de came

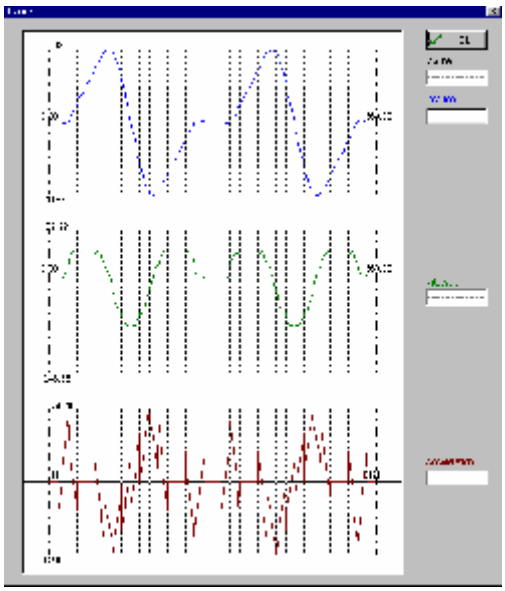
Cette éditeur s'obtient en visualisant une variable de type tableau de came dans l'onglet « variables globales ». Cette éditeur permet de configurer le nombre d'élément et les différents points de la came. Il est composé du tableau de came et d'outils d'aide à la mise en œuvre.



Le tableau de came est composé des différents points de la came caractérisés par un index variant de 1 à N et deux points supplémentaires « Entrée » et « Sortie ». Chaque point possède une distance maître et une distance esclave. Un clique sur le bouton droit de la souris fait apparaître un menu comprenant les commandes d'ajout et de suppression de points. Le menu n'apparaît que si le curseur se trouve sur les colonnes « maître » ou « esclave » et que la case n'a pas été au préalable sélectionnée.

Parmi les outils d'aide à la mise en œuvre se situant sur la droite du tableau de came, on observe les deux encadrées « Maître » et « Esclave » qui permettent de définir les cartes qui réaliseront le mouvement. Pour le « Maître », on doit en plus définir sa vitesse. Ce paramètre sert pour le calcul de point lors de la représentation graphique.

La représentation graphique du profil de came s'obtient avec le bouton « graphic ». La fenêtre qui apparaît présente trois graphiques en fonction de la distance maître parcourue : la distance esclave parcourue, la vitesse esclave et l'accélération esclave. Ces courbes sont intéressante pour observer les performances nécessaires pour les moteurs à condition de bien choisir la vitesse maître spécifier dans l'écran précédent. Lors d'une modification du tableau de came, il est nécessaire de valider de nouveau le bouton « graphic » pour que les modifications sur les graphiques soient prises en compte. Lorsque la fenêtre graphique est présente a l'écran, il est nécessaire de valider le bouton « OK » pour pouvoir quitter l'éditeur de tableau de came.



L'encadré « Entrée/Sortie » permet de faciliter la réalisation de came en fonction du type de came choisie. Cette encadré permet de définir le type de came. Et en fonction de celui-ci et

après validation du bouton « calcul » de calculer les points d'entrée et de sortie de la came. Les calculs permettent d'obtenir les caractéristiques d'entrées et de sorties définies dans le chapitre réservé a l'explication des cames.

- Came mono-coup validé : Les pentes de début et de fin seront identiques et égale a 0
- Came finie validé : Les pentes de début et de fin seront identiques de même que les positions de début et de fin de l'esclave.
- Came finie non validé (came infinie) : Seul les pentes de début et de fin seront identiques

Les caractéristiques mono-coup ou non et finie ou infinie peuvent être combinées. Les calculs réalisés permettent ainsi de définir les pentes d'enchaînement en fonction du type de came choisi. Il est nécessaire d'effectuer un nouveau calcul, lors d'une modification d'un paramètre du tableau de came ou du type de came.

Attention : Si l'on veut réaliser une came particulière ne correspondant pas à un type précisée dans l'encadré « Entrée/Sortie », il ne faut surtout pas valider la fonction de calcul.

De même, les valeurs de l'axe maître devront être cohérente avant de quitter l'éditeur.

4- LANGAGE DE PROGRAMMATION

4-1- Introduction

4-1-1- Description

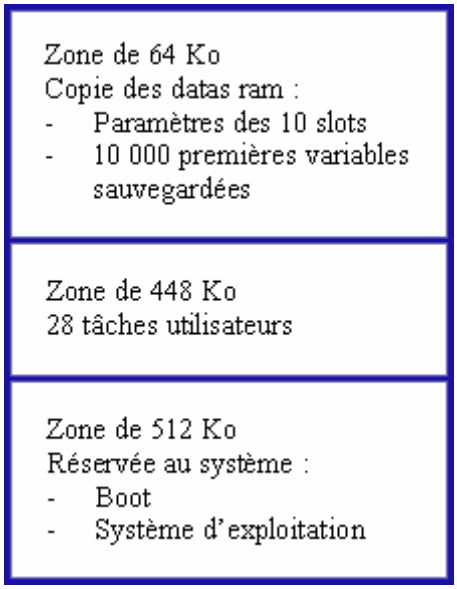
Motion Control Basic est un outil de programmation puissant et simple à utiliser. Il offre une architecture structurée rencontrée sur les langages de haut niveau. Pour une programmation flexible, ce langage est géré par un noyau temps réel multitâches, utilisant des instructions pseudo-basic et contenant également toutes les fonctions de contrôle de mouvement et 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 MCB 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 de la MCS

Mémoire Flash 1 Mo



Mémoire Ram 512 Ko sauvegardée par pile

Cette RAM comporte une zone de 128 Koctets correspondant au fichier de la MCS. Ce fichier permet à l'utilisateur de venir y déposer des données pendant l'exécution de la MCS. 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 MCB. 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 MCB.

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.

☏ Attention : Le type « tableau de came » a un mode de stockage particulier. Chaque élément est composé de deux variables réelles. Le tableau contient également 2 éléments internes supplémentaires : le point d'entrée, le point de sortie. Par exemple, si on déclare un tableau de came de 10 éléments à l'adresse 100, il sera constitué de (10 + 2) * 2 = 24 variables réelles d'adresse 100 à 123 inclus.

adresse 100 : point d'entrée maître	adresse 101 : point d'entrée esclave
adresse 102 : point 1 maître	adresse 103 : point 1 esclave
.....	.....
adresse 121 : point 10 maître	adresse 122 : point 10 esclave
adresse 122 : point de sortie maître	adresse 123 : point de sortie esclave

Tableau récapitulatif des différents types :

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 »
Element of Camtable	⇨ X : real ⇨ Y : real	Unstored variable: 12 bytes Stored variable: 12 bytes	

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 :

Source \ Destination	Bit	Octet	Entier	Entier long	Réel	Chaîne de caractères
Bit	X	Directe	Directe	Directe	X	Str$ ou Format$
Octet	'.1' à '.8'	X	Directe	Directe	Directe	Str$ ou Format$
Entier	'.1' à '.16'	'.L' ou '.H'	X	Directe	Directe	Str$, Mki$, Mkir$ ou Format$
Entier long	X	LongToByte	LongToInteger	X	Directe	Str$, Mkl$, Mklr$ ou Format$
Réel	X	RealToByte	RealToInteger	RealToLong	X	Str$ ou Format$
Chaîne de caractères	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
VarOctet=LongToByte(VarLong)        ' VarOctet=5
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
  - ⇨ Mova, Movap, Movac, Movr, Stop, Traj, Home
  - ⇨ 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 de la MCS, 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.

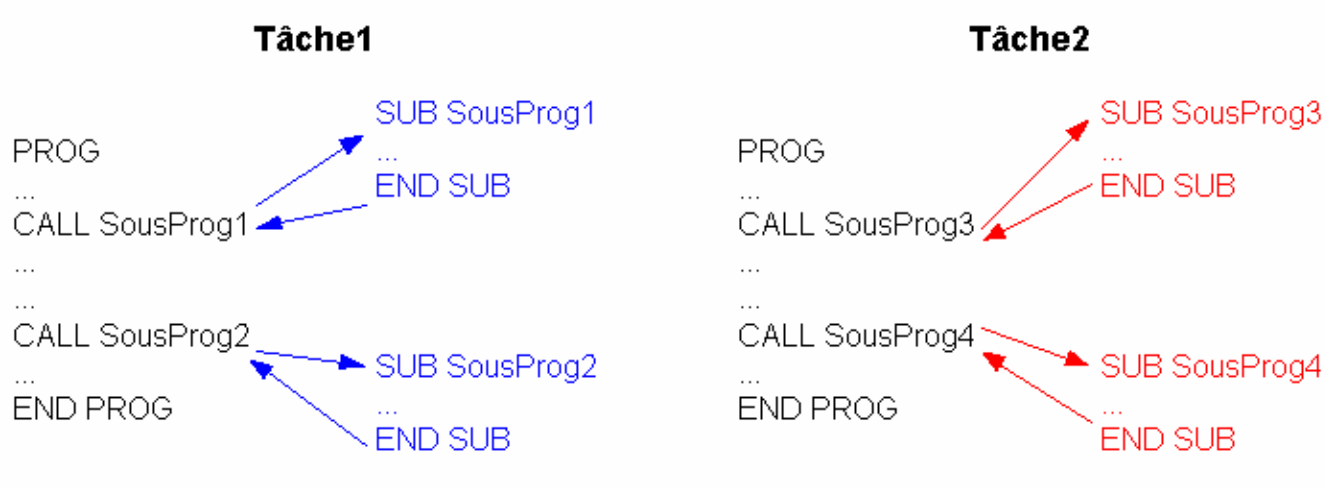
```
Exemple :
ProcessEnable  : variable globale de type bit
Tâche Process1                  Tâche Process2
Prog                            Prog
.....                           .....
ProcessEnable=1                 Wait ProcessEnable=1
Wait ProcessEnable=0            .....
....                       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

☏ Attention : L'arrêt ou la suspension de la tâche n'affecte pas les mouvements lancés par celle-ci

```
Exemple :
Tâche Contrôle                  Tâche Process
Prog                            Prog
.....                           .....
If ProcessError=1 Then          Mova(X=1000)
   Halt Process                 Out(S1)=1
   Stop(X)                      Mova(X=2000)
End If                          .....
....                            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.

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. Par exemple :

```
VEL%(X)=100                         ' Vitesse rapide
STTA(X=2000)                        ' Start absolu à la position 2000
MOVE_ON:
IF POS_S(X) >1000 THEN VEL%(X)=50  ' Vitesse lente à la moitié
                                    ' de la distance
IF POS_S(X)>1500 THEN OUT(S1)=1 ELSE OUT(S2)=1
IF POS_S(X)>1700 THEN
    FlagInfo1=1
    IF OUT(S2)=1 THEN OUT(S2)=NOT OUT(S2) ' Sortie clignotante
 ELSE
    FlagInfo2=1
    OUT(S2)=0                              ' Reset de la sorties
END IF
IF MOVE_S(X)=On THEN GOTO MOVE_ON
```

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.

Par exemple :

```
VEL%(X)=100                          ' Vitesse rapide à 100 %
STTA(X=2000)                         ' Start absolu en position 2000
WHILE  MOVE_S(X)  DO                 ' Tant que le moteur est en mouvement
IF POS_S(X) >1000 THEN VEL%(X)=50    ' Vitesse lente à la moitié
                                     ' de la distance
END WHILE                            ' Fin de boucle
```

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.

Par exemple :

```
VEL%(X)=100                          ' Vitesse rapide
STTA(X=2000)                         ' Start absolu en position 2000
REPEAT
  IF POS_S(X) >1000 THEN VEL%(X)=50       ' Vitesse lente à la moitié
                                          ' de la distance
UNTIL NOT MOVE_S(X)                  ' reboucler jusqu'à ce que
                                     ' le moteur soit arrêté
```

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 : 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 de la MCS, 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 (<Masque>,<Durée>)

<Masque> : Expression de type entier permettant de sélectionner les événements souhaités.

⇨ Bits 0...6 : activation des entrées n° 1 à n° 7 de la première carte entrée détectée par le système (la recherche se fait du slot A vers le slot J). 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 7 : base de temps

⇨ Bits 8...15 : activation de la capture associée au registre 1 (Capture1) des 1 à 8 cartes SRV 85 détectées par le système (la recherche se fait du slot A vers le J).

<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 utilisée.

Après l'affectation de la configuration, la tâche événementielle est exécutée dès qu'au moins un événements 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, MOVA ...).

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 JUMP EventE1
      IF Event%.2=1 THEN JUMP EventE2
      IF Event%.8=1 THEN JUMP EventTime
      IF Event%.9=1 THEN JUMP EventCapture1
      ....
      EventE1  :
      ....
       JUMP EndTask
      EventE2  :
      ....
       JUMP EndTask
      ....
      EndTask :
      END PROG
```

5- PROGRAMMATION DU CONTRÔLE DE MOUVEMENT

5-1- Introduction

La MCS peut gérer de 1 à 8 cartes servo. Chaque carte intègre un processeur DSP 32 MHz pour assurer un asservissement avec un «update time » de 330 µs, un FPGA pour traiter les signaux codeur jusqu'à 2 MHz.

Les ordres de mouvement sont envoyés aux cartes d'axes grâce au processeur central 32 bits qui décode les tâches pseudo-basic.

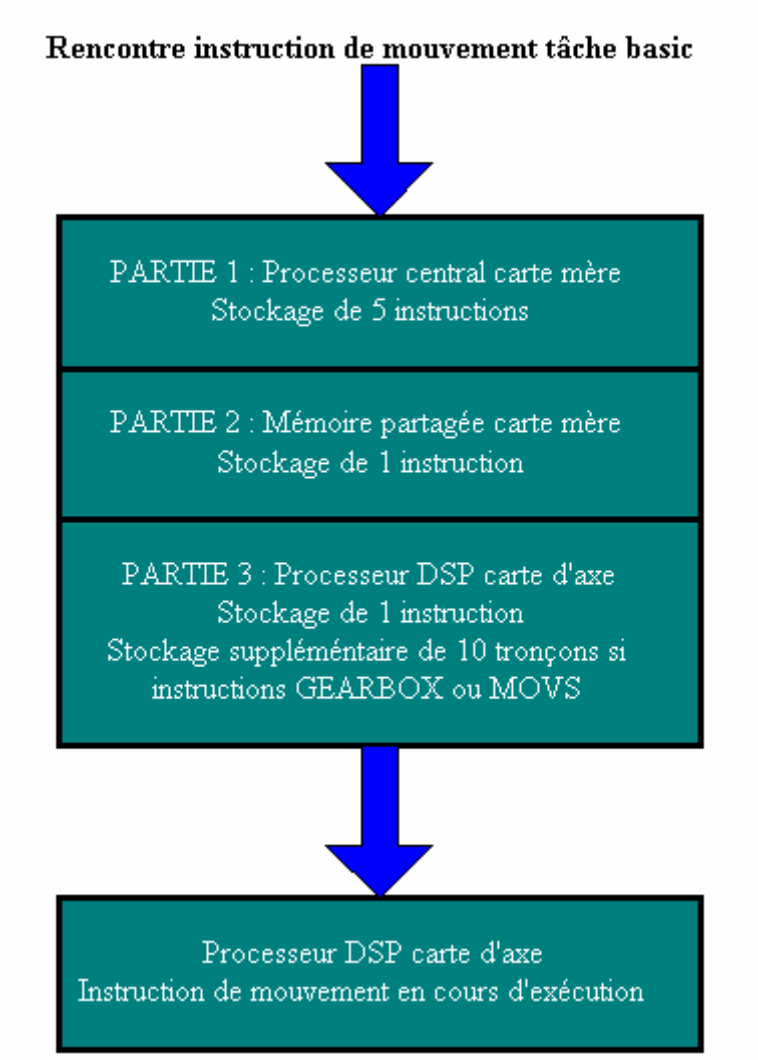
Le logiciel MCB contient de nombreuses instructions évoluées pour le contrôle de mouvement :

positionnement, arbre électrique, came électronique, superposition, compensation, interpolation...

5-2- Buffer de mouvements

Le processeur central de la MCS qui exécute les tâches compilées échange régulièrement des informations avec les cartes d'axes.

Les informations de mouvement (MOVA, STTA, MOVS, GEARBOX, STOP ...) sont transmises à travers un buffer logiciel. Chaque axe possède un buffer divisé en 3 parties :



Lorsqu'une tâche rencontre une instruction de mouvement, elle l'envoie dans le buffer de l'axe concerné :

- S'il n'y a pas d'instruction de mouvement en cours d'exécution et que le buffer est vide, l'instruction passe à travers les parties 1, 2, 3 du buffer et est exécutée immédiatement.
- Si une instruction de mouvement est déjà en cours, la nouvelle instruction est stockée dans le buffer.
- Si le buffer est déjà plein, la tâche est bloquée jusqu'à ce qu'une place soit disponible.

La partie 3 du buffer comporte une zone de stockage de 10 tronçons supplémentaires pour les instructions GEARBOX, GEARBOXM, MOVS, MOVSM, MOVSC :

- GEARBOX ou GEARBOXM est composé d'un tronçon s'il n'y a pas de paramètre d'accélération.
- GEARBOX ou GEARBOXM est composé de 2 tronçons s'il y a un paramètre d'accélération.
- MOVS ou MOVSM ou MOVSC est composé de 1 à 3 tronçons selon les valeurs de « Distance esclave », « Distance accélération », « Distance décélération ». Par exemple :

```
MOVS(Y,X,Dmaitre,100,0,0)          '1 tronçon
MOVS(Y,X,Dmaitre,100,100,0)        '1 tronçon
MOVS(Y,X,Dmaitre,100,0,100)        '1 tronçon
MOVS(Y,X,Dmaitre,100,20,0)         '2 tronçons
MOVS(Y,X,Dmaitre,100,0,20)         '2 tronçons
MOVS(Y,X,Dmaitre,100,20,20)'3 tronçons
```

- Le mouvement n'est stocké dans la zone de 10 tronçons que lorque tous les tronçons élémentaires peuvent y être stockés.
- Le numéro de mouvement suit le mouvement (et les tronçons) jusqu'à l'exécution complète de ce mouvement.

Différentes instructions permettent de connaître l'évolution des éléments à l'intérieur du buffer :

- BUFMOV_S indique le nombre d'instructions en attente dans le buffer.
- ORDER permet d'affecter un numéro à chaque instruction envoyée dans le buffer.
- ORDER_S permet de savoir quel numéro de mouvement est en cours d'exécution.
- MOVE_S est égale à 0 si le buffer est vide et s'il n'y a plus de mouvement en cours d'exécution.

☏ Attention :

- Les instructions STOP et SSTOP arrêtent le mouvement en cours d'exécution et vide le buffer de l'axe.
- Si l'axe est en boucle ouverte (AXIS(Axe)=Off), toute instruction de mouvement rencontrée dans une tâche passe par le buffer et est directement consommée par la carte d'axe sans exécution du mouvement.

5-3- Mode asservi / non asservi

5-3-1- Passage en mode non asservi

Un axe passe en mode non asservi (boucle ouverte) :

- A chaque redémarrage de la MCS.
- A chaque exécution de l'instruction AXIS(Axe)=Off à partir d'une tâche.
- A chaque exécution de l'instruction CONS à partir d'une tâche.
- Sur erreur de poursuite de n'importe quel axe servo (sauf si l'instruction SECURITY a été affectée).
- Sur un forçage à partir des menus de debug (bouton asservi / débrayé ...), du menu communication (arrêt des tâches, redémarrage des tâches, Envoyer les tâches).

L'instruction AXIS_S(Axe) permet de lire l'état dans lequel se trouve la carte d'axe.

Si une instruction de mouvement est envoyée à une carte paramétrée en boucle ouverte, elle sera consommée par celle-ci mais le mouvement ne sera pas effectué. Par exemple :

Tâche Process

```
PROG
...
...             ' la MCS a détecté une erreur de poursuit sur un
...             ' axe => tous les axes passent en mode non asservi
MOVA(X=1000)    ' le mouvement est consommé par la carte d'axe
                ' mais non effectué
OUT(S1)=1       ' sortie S1=1
MOVA(X=2000)    ' le mouvement est consommé par la carte d'axe
                'mais non effectué
OUT(S1)=0       ' sortie S1=0
...             ' la sortie S1 est passée fugitivement à 1 car
...             ' l'instruction Mova(X=2000)a pris peu de temps au système
END PROG
```

5-3-2- Passage en mode asservi

Pour qu'un axe servo puisse piloter et contrôler les mouvements, il est nécessaire de le passer en mode asservi.

Un axe passe en mode asservi (boucle fermée) :

- A chaque exécution de l'instruction AXIS(Axe)=On à partir d'une tâche.
- Sur un forçage à partir des menus de debug (bouton asservi).

L'instruction AXIS_S(Axe) permet de lire l'état dans lequel se trouve la carte d'axe et CONS_S(Axe) de lire sa tension.

Pour des raisons de sécurité, il est nécessaire d'affecter une sortie logique de la MCS à la gestion de la fonction « Enable / Disable » des drives reliés aux cartes d'axes.

Cette sortie devra être reliée sur l'entrée « Enable » du drive. La gestion de la sortie sera insérer dans une tâche de surveillance non bloquante. Par exemple :

Tâche Defauts

```
PROG
....
OUT(EnableDrive)=AXIS_S(Axe)        ' si axe en mode asservi
....                                ' => OUT(EnableDrive)=1 sinon =0
END PROG
```

5-4- Paramétrage d'un axe

5-4-1- Paramétrage d'un axe

Un axe doit être paramétré avant de pouvoir l'utiliser.

L'accès aux paramètres se fait à partir de l'écran de configuration du MCB après avoir sélecté la carte ou à partir d'une ligne basic de la forme « NomParametre_P » à l'intérieur d'une tâche.

La méthode de paramétrage la plus classique consiste à rentrer les paramètres de chaque carte à partir de l'écran de configuration du MCB, de les transférer dans la MCS grâce au menu Communication « Envoyer la configuration ». Si par la suite on souhaite régler un paramètre en temps réel sur la MCS, on peut passer en mode debug sur le MCB et agir dessus.

5-4-2- Unité utilisateur

En fonction de l'application, de la mécanique (axe linéaire, rotatif), il est d'intéressant de pouvoir affecter à chaque axe une unité utilisateur représentative : mm, point (point codeur * 4), degrés, radian, pouce, tour, unité quelconque...

En fait, l'unité est utilisée uniquement sur les écrans du MCB afin d'y apporter un confort d'utilisation et de compréhension. C'est pour cela qu'on ne la retrouve pas sous la forme de « NomParamètre_P ».

Par exemple, si le choix de l'unité est « mm », dans l'écran de paramétrage « Profil de vitesse » du MCB, la vitesse sera exprimée en mm/s, les accélérations et décélérations en mm/s²...

Par contre la MCS, au niveau interne, ne connaît que des points ou des incréments (point codeur * 4).

5-4-3- Codeur

Chaque carte servo possède une entrée codeur pour gérer le retour de la boucle de position. Elle peut être de type incrémental (sur carte SRV 15, SRV 85...) ou absolu (sur carte SSI 15).

A) Type de codeur

↳ Pour un codeur incrémental :

Rentrez dans le champ « Résolution » du MCB (paramètre associé ENCODER_P), le nombre de points * 4 du codeur utilisé.

↳ Pour un codeur SSI :

Rentrez dans le champ « Nombre de points » du MCB, le nombre de points total du codeur utilisé (si codeur 360 points monotour => 360, si codeur 360 points 10 tours => 3600).

Rentrez dans le champ « Echelle / Points » (paramètre associé ENCODER_P), le nombre de points pour 1 tour codeur.

Rentrez dans le champ « Nombre de bits », le nombre de bits total du protocole SSI géré par le codeur, dans « Fréquence », sa fréquence.

B) Unité par tour codeur

↳ Pour un codeur incrémental :

Rentrez dans le champ « Unité par tour » du MCB (paramètre associé UNITREV_P), l'avance en unité utilisateur de la mécanique pour 1 tour codeur. Par exemple pour une vis à billes au pas de 5mm et codeur en bout de vis, rentrez la valeur 5.

↳ Pour un codeur SSI :

Rentrez dans le champ « Echelle / pour » (paramètre associé UNITREV_P), l'avance en unité utilisateur de la mécanique pour 1 tour codeur.

Toutes les valeurs de paramètres, de position, de mouvement ... exprimées en unité utilisateur sont traduites à l'intérieur de la MCS en incrément avec la formule :

ValeurIncrémentAxe = (ValeurUnitéAxe * ENCODER_P(Axe)) / UNITREV_P(Axe)

C) Inversion du sens codeur (pour codeur incrémental seulement)

Cochez la case « Inversion du sens codeur » du MCB (paramètre associé ENCINV_P) si vous souhaitez inverser le sens de comptage du codeur en évitant des modifications de câblage.

D) Axe modulo (pour codeur incrémental seulement)

Un axe infini (sans limites de course mécanique) doit être déclaré en axe modulo car sinon il pourrait atteindre les limites de comptage des impulsions codeur (+/- 2^24 incréments) et provoquer un dysfonctionnement de la MCS.

Cochez la case « Axe modulo » du MCB (paramètre associé MODULO_P) pour déclarer l'axe en modulo et rentrez sa valeur dans le champ « Valeur » (paramètre associé MODVAL_P).

Par exemple, la valeur du modulo d'un axe rotatif exprimé en degré peut être de 360°.

5-4-4- Profil de vitesse

Une trajectoire en positionnement intègre les phases d'accélération, de vitesse plateau, de décélération.

Les champs contenus dans la configuration de la carte à partir du MCB permettent de donner des valeurs par défaut à ces différentes phases. Les valeurs sont prises en compte à chaque démarrage de la MCS, elles sont également utilisées par le mode debug de la carte et par les instructions ACC%, DEC%, VEL%.

Rentrez dans le champ « Vitesse » (paramètre associé VEL_P), la vitesse par défaut en unité/s (plage de 5 10^-5 à 1.5 10^6 incréments/s).

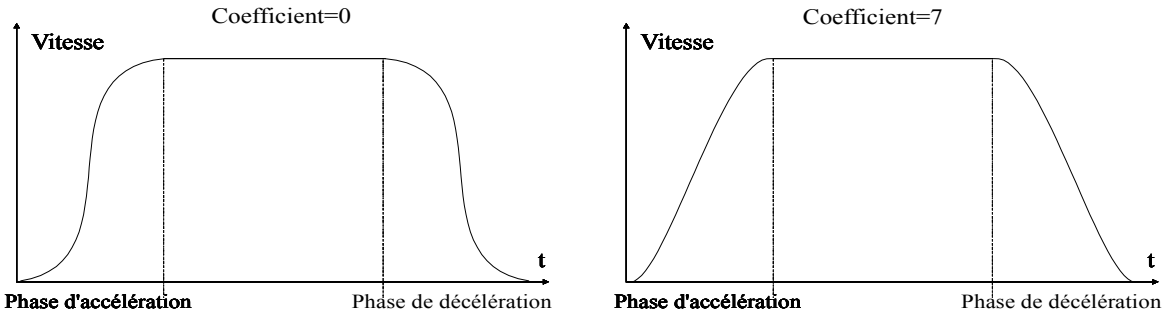
Rentrez dans les champs « Accélération » (paramètre associé ACC_P) et « Décélération » (paramètre associé DEC_P), l'accélération et la décélération par défaut en unité/s^2 (plage de 5 10^-5 à 1.5 10^6 incréments/s^2).

Si l'on désire modifier ces phases à l'intérieur d'une tâche basic, utilisez les instructions ACC, DEC, VEL, ACC%, DEC%, VEL%. L'instruction VEL_S permet de lire la vitesse de l'axe.

Elles sont effectives à tout moment : axe en mouvement ou à l'arrêt.

Les phases d'accélération/décélération peuvent être linéaires ou sinus (paramètre associé SIN_P).

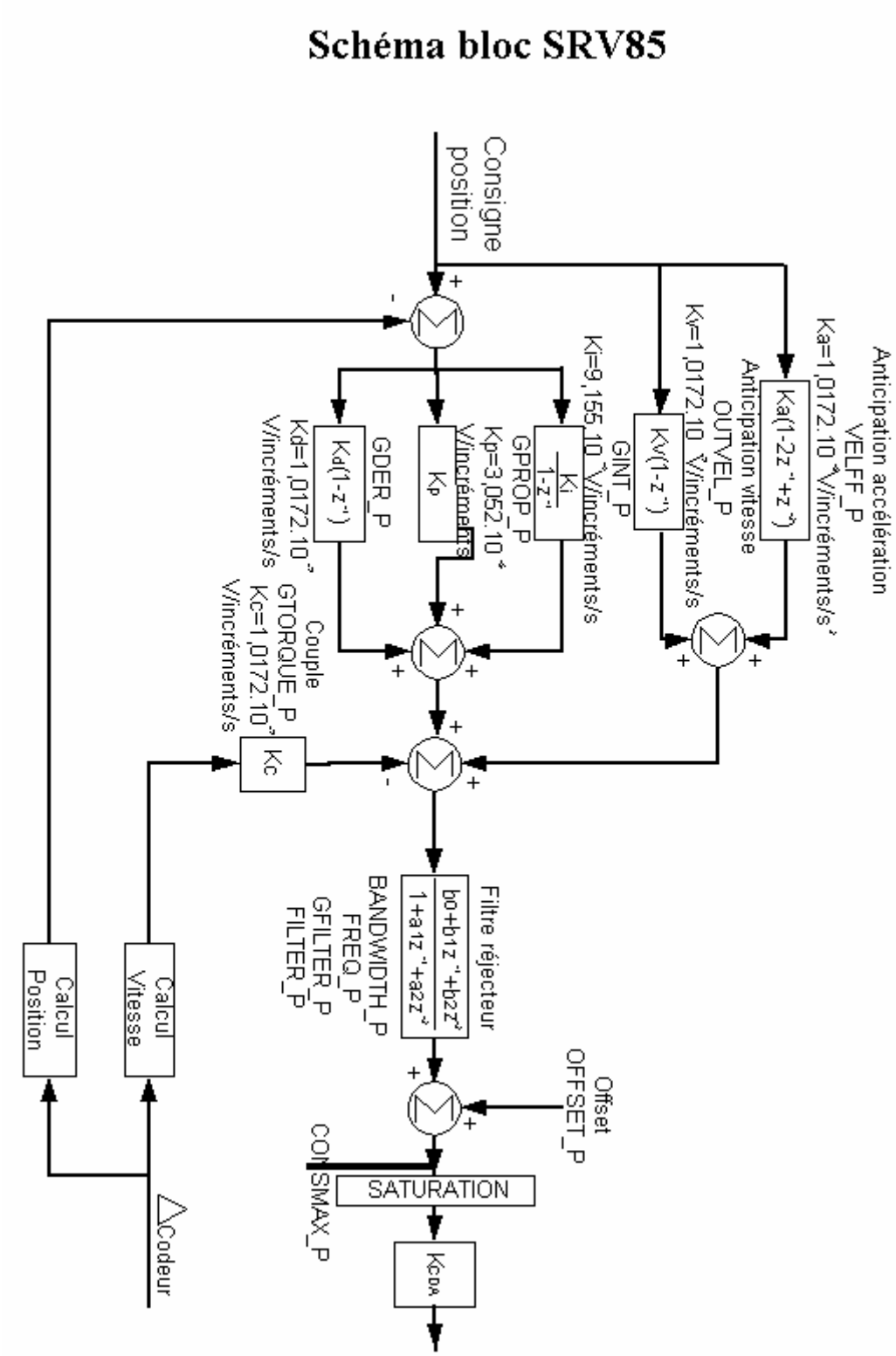
Dans le cas où le sinus a été choisi, un coefficient (paramètre associé SINVAL_P) permet d'intégrer une portion linéaire dans le sinus. Par exemple :



5-4-5- Régulation

L'asservissement d'un axe est réglable à partir des paramètres du régulateur. Il peut être programmé en mode « vitesse » ou en mode « couple » (seulement sur SRV 85) ; par défaut, en mode vitesse.

A) Schéma bloc



B) Gains

Réglage du régulateur sur un axe infini :

↳ Mettez l'erreur de poursuite maxi à une valeur importante (ex : 2 ou 3 tours moteur)

↳ Mettez le gain proportionnel (paramètre associé GPROP_P) à une valeur faible et tous les autres gains à 0.

↳ Mettez une valeur dans le gain de couple (paramètre associé GTORQUE_P) dans le cas d'un asservissement en mode couple.

↳ Lancez un mouvement infini.

↳ Augmentez le gain anticipation de vitesse (paramètre associé OUTVEL_P) jusqu'à obtenir une erreur de poursuite proche de 0.

↳ Relancez des mouvements pour augmenter le gain anticipation d'accélération (paramètre associé VELFF_P) qui intervient dans les phases transitoires jusqu'à obtenir une erreur de poursuite proche de 0.

↳ Augmentez le gain proportionnel pour apporter de la raideur et de la précision à l'asservissement tant que le système est stable (pas d'oscillations observées).

↳ Augmentez le gain de couple dans le cas d'un asservissement en mode couple pour apporter de la raideur et de la précision.

↳ Si nécessaire, augmentez le gain dérivée (paramètre associé GDER_P) pour apporter encore plus de raideur, le gain intégrale (paramètre associé GINT_P) pour vaincre l'erreur statique.

↳ Mettez l'erreur de poursuite maxi (paramètre associé FEMAX_P) au double de l'erreur de poursuite observée.

☏ Attention : Si l'asservissement n'est pas utilisé en mode couple, le gain de couple (GTORQUE_P) doit être forcé à 0.

C) Consigne analogique

↳ Cochez la case « Inversion de la consigne » du MCB (paramètre associé CONSINV_P) si vous souhaitez inverser le signe de la consigne analogique en évitant des modifications de câblage.

↳ Rentrez dans le champ «Offset» (paramètre associé OFFSET_P) la valeur de l'offset en volt si le moteur dérive lorsque l'axe est en mode non asservi. (exemple : offset = 0.04 v).

↳ Rentrez dans le champ «Limite» (paramètre associé CONSMAX_P) la valeur de la consigne analogique maxi en volt. Celui-ci peut être utilisé en mode couple pour limiter le couple : en règle générale, le laisser à sa valeur par défaut 10v.

D) Erreur de poursuite maxi

Dès qu'un axe passe en mode asservi, il est contrôlé à tout moment : à l'arrêt, en mouvement.

Si la différence entre sa position théorique calculée et sa position réelle donnée par le retour codeur est supérieure à l'erreur de poursuite maxi, le système passe tous les axes servo en mode non asservi et ouvre le contact de chien de garde (sauf si utilisation de l'instruction SECURITY).

Le réglage de cette valeur est très importante : une valeur trop petite entraîne des arrêts intempestifs sur les axes, une valeur trop grande influe sur la sécurité des organes électriques et mécaniques.

Rentrez dans le champ «Erreur de poursuite maxi » du MCB (paramètre associé FEMAX_P) la valeur adéquate.

E) Fenêtre de position

Lorsque l'on envoie un axe à une position, la MCS considère que le mouvement est terminé quand le profil théorique de la trajectoire est exécuté et que la position réelle est comprise entre +/- la fenêtre de position. Par exemple, sur une machine de perçage où l'on recherche une précision de +/- 0.1mm, on réglera la fenêtre à cette valeur.

Rentrez dans le champ «Fenêtre de position » du MCB (paramètre associé POSMIN_P) la valeur de précision recherchée.

5-4-6- Filtre réjecteur (carte SRV 85 seulement)

Un filtre de réjection de fréquence peut être intégré dans la boucle d'asservissement. Il permet d'atténuer ou de supprimer une bande de fréquence sur la consigne analogique de la carte.

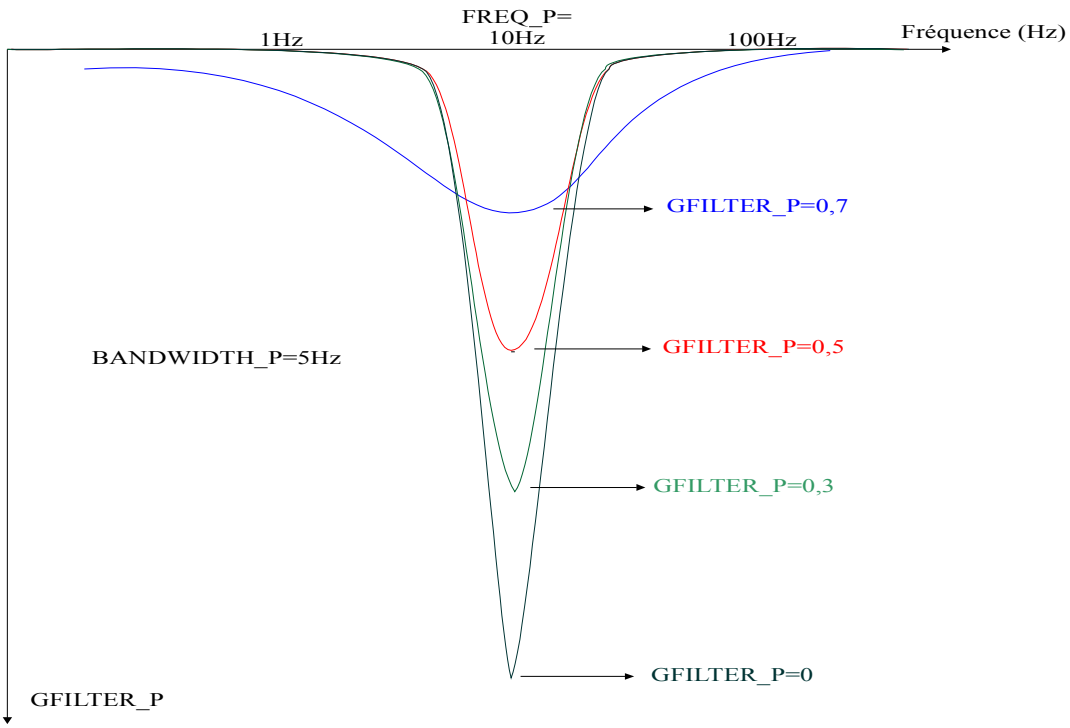
Il est composé d'une fréquence centrale, d'une largeur de bande, d'un gain atténuateur.

Cochez la case « Filtre » du MCB (paramètre associé FILTER_P) si vous souhaitez utilisez le filtre.

Rentrez dans le champ « Fréquence centrale » (paramètre associé FREQ_P), la fréquence en Hz.

Rentrez dans le champ « Bande » (paramètre associé BANDWIDTH_P), la largeur de bande à –3db en Hz.

Rentrez dans le champ « Gain » (paramètre associé GFILTER_P), la valeur de l'atténuation entre 0 et 0.7.



Exemple :

La consigne analogique a une composante continue de 4v et une fréquence d'oscillation de 30 Hz avec une amplitude de +/- 0.4v.

Si FREQ_P=30 et GFILTER_P=0.7 l'amplitude de l'oscillation sera de +/- 0.28v.

Si FREQ_P=30 et GFILTER_P=0.5 l'amplitude de l'oscillation sera de +/- 0.2v.

Si FREQ_P=30 et GFILTER_P=0 il n'y aura plus d'oscillation.

5-4-7- Prise d'origine

Un cycle de prise d'origine est nécessaire sur une carte servo après chaque redémarrage de la MCS si la carte est munie d'une entrée codeur incrémental et qu'elle pilote un axe fini.

Ce cycle a pour but de remettre à 0 le compteur impulsions codeur de la carte d'axe lorsque certaines conditions sont rencontrées.

L'instruction HOME (Axe, n° type) lance le cycle. cette instruction est bloquante pour la tâche pendant toute la durée du cycle.

L'instruction HOME_S indique si le cycle d'origine a été effectué. Celle-ci est forcée à 0 par le système en début de cycle et est mise à 1 en fin de cycle.

Les instructions ZERO_S et SENSOR_S donnent respectivement l'état logique du signal zéro codeur et de l'entrée home/capture de la carte

A) Types

La carte supporte une dizaine de types : soit remise à 0 immédiate, sur capteur relié à la carte servo ou à une carte entrées, sur capteur et top zéro ...

Pour remettre à 0 de façon immédiate le compteur d'impulsions, utilisez HOME(Axe,0) ou CLEAR(Axe).

B) Vitesse

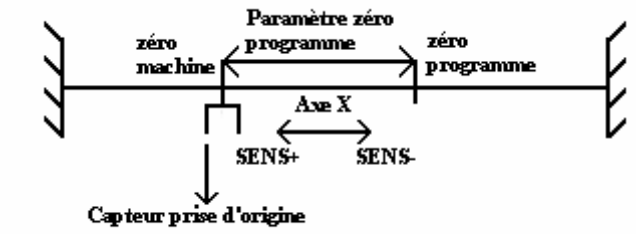
Rentrez dans le champ « Vitesse home » (paramètre associé VELHOME_P), la vitesse souhaitée en unité/s. Dans le cas d'un type avec recherche rapide, celle-ci s'effectuera d'abord à vitesse courante puis passera en vitesse home.

C) Dégagement

Rentrez dans le champ « Dégagement » (paramètre associé « DISHOME_P »), une valeur en unité, si l'on désire qu'après avoir atteint sa position d'origine, l'axe se dégage d'une distance égale à la valeur.

D) Zéro programme

Lorsque l'axe a rencontré sa position d'origine, il remet son compteur à 0 et s'arrête en se repositionnant sur 0. Cette position correspond au zéro machine. Si l'on souhaite travailler dans un autre repère à partir du programme, il suffit de rentrez une valeur dans le champ « Zéro programme » du MCB (paramètre associé ZERO_P). Par exemple :



Zéro programme = zéro machine - Paramètre zéro programme

5-4-8- Butées logicielles

Sur un axe à course limité, il est possible de déclarer des butées logicielles mini et maxi. Si le système détecte que l'axe est en dehors de celles-ci, il monte à 1 le flag LIM_S(Axe). Ce flag peut ensuite être traité dans une tâche basic ou ladder de surveillance.

☏ Attention : le système affecte seulement l'état de LIM_S, LIMMIN_S ou LIMMAX_S mais n'agit en aucun cas sur le pilotage de l'axe.

Cochez la case « Activation » du MCB (paramètre associé LIM_P) si vous souhaitez utilisez les butées.

Rentrez dans le champ « Mini » (paramètre associé LIMMIN_P), la valeur de la butée mini.

Rentrez dans le champ « Maxi » (paramètre associé LIMMAX_P), la valeur de la butée maxi.

La valeur mini doit être inférieure à la valeur maxi.

5-5- Déclaration d'un axe en mode virtuel

5-5-1- Déclaration d'un axe en mode virtuel

A partir d'une tâche basic, il est possible de faire passer une carte servo en mode virtuel grâce à l'instruction LOOP(Axe)=On. Dans ce mode, la carte simulera les impulsions codeur de façon interne et ainsi toute commande envoyée sera exécutée virtuellement.

☏ Attention : La sortie analogique sera tout de même gérée.

Ce mode est intéressant lors de la phase développement du programme : on peut tester la globalité de l'application sans avoir les variateurs et moteurs connectés.

Il peut aussi être utilisé pour des applications avec des fonctions de synchro qui nécessitent un maître parfaitement stable. Par exemple :

Soit 2 axes servo X, Y à relier en arbre électrique avec un rapport ½ : X maître, Y esclave.

Problème : X provoque des petites vibrations qui pourraient être amplifiées par Y

Solution : ajouter une carte servo travaillant en mode virtuel et étant le maître de X et Y

Tâche Process1

```
PROG
....
AXIS(X)=On
AXIS(Y)=On
LOOP(Maître)=On
AXIS(Maître)=On
.....
GEARBOX (X,Maître,1,1,0)
GEARBOX (Y,Maître,1,2,0)
STTI (Maître)=+
.....
END PROG
```

5-6- Positionnement

5-6-1- Mouvements absolus

A) Départ de mouvement : STTA

Pour lancer un mouvement vers une position absolue et ne pas attendre sa fin pour poursuivre l'exécution de la tâche, on doit utiliser STTA. Cette instruction est très utile si la vitesse ou la position à atteindre doit changer en cours de mouvement. Avec cette fonction, l'erreur absolue est minimale.

Cette instruction est non bloquante pour la tâche (excepté si le buffer de mouvements est plein).

Elle utilise les valeurs courantes d'accélération, de décélération et de vitesse. La syntaxe est :

STTA (<Axe1>=<Position1> [{, <Axe2>=<Position2>}])

Par exemple :

```
VEL%(X)=100                ' Vitesse normale
```

```
STTA(X=2000)                ' Start absolu en position 2000
WAIT POS_S(X) >200          ' Attente position 200
OUT (A1)=1                  ' Activation d'une sortie
WAIT POS_S(X) >700          ' Attente position 700
OUT (A1)=0                  ' Désactivation de la sortie
WAIT  NOT  MOVE_S(X)        ' Attente fin de mouvement
```

Dans cet exemple, pendant le mouvement, on peut changer des sorties car l'exécution n'est pas interrompue.

Si l'instruction MERGE est activée et que l'on charge plusieurs STTA dans la carte d'axe, les mouvements seront exécutés les uns après les autres sans passer par une vitesse nulle.

Si l'axe est modulo, un lancement à une position sera effectué dans le sens positif si la valeur demandée est positive, sens négatif dans le sens contraire. Par exemple :

Axe modulo 360°

Axe en position initiale à 90°

```
STTA(X=-10) 'déplacement dans le sens – d'une distance de 80°
WAIT  NOT  MOVE_S(X)
STTA(X=350) 'déplacement dans le sens + d'une distance de 340°
WAIT  NOT  MOVE_S(X)
STTA(X=350) 'déplacement dans le sens + d'une distance de 340°
WAIT  NOT  MOVE_S(X)
STTA(X=20) 'déplacement dans le sens + d'une distance de 30°
WAIT  NOT  MOVE_S(X)
```

B) Mouvement : MOVA

La fonction MOVA envoie un ou plusieurs axes à une position absolue. Elle utilise les valeurs courantes d'accélération, de décélération et de vitesse. La syntaxe est :

MOVA (<Axe1>=<Position1> [{, <Axe2>=<Position2>}])

Cette fonction envoie <Axe1> à la position absolue dont la valeur est <Position1>. Le programme attend la fin du mouvement avant de continuer. L'erreur de positionnement absolue est minimale.

Par exemple :

```
MOVA(X=100)
CALL Percage
MOVA(X=0)
```

L'instruction MOVA est bloquante pour la tâche tant que le mouvement n'est pas terminé (condition MOVE_S(Axe)=0).

MOVA(X=100) est équivalent à STTA (X=100)

WAIT NOT MOVE_S(X)

Pour obtenir une précision relative accrue on peut demander un mouvement absolu utilisant la position courante : MOVA(X=POS_S(X)+100) ' Déplacement relatif avec précision relative

Si l'axe est modulo, le mouvement sera effectué dans le sens positif si la valeur demandée est positive, sens négatif dans le sens contraire. Par exemple :

Axe modulo 360°

Axe en position initiale à 90°

```
MOVA(X=-10) 'déplacement dans le sens – d'une distance de 80°
MOVA(X=350) 'déplacement dans le sens+ d'une distance de 340°
MOVA(X=350) 'déplacement dans le sens+ d'une distance de 340°
MOVA(X=20) 'déplacement dans le sens+ d'une distance de 30°
```

C) Mouvement déclenché sur position : MOVAP

Cette instruction est équivalente à MOVA mais en plus, elle intègre une condition de déclenchement donnée par la position d'un autre axe.

D) Mouvement déclenché sur entrée capture : MOVAC (seulement sur SRV 85)

Cette instruction est équivalente à MOVA mais en plus, elle intègre une condition de déclenchement donnée par une entrée rapide « capture ».

E) Trajectoire : TRAJ

La fonction Trajectoire est conçue pour simplifier la définition de mouvements complexes.

Pour faire un mouvement à une vitesse spécifique, une accélération ou une décélération donnée, ces paramètres doivent être précisés avant de lancer le mouvement. Avec la fonction TRAJ, on peut lancer un mouvement absolu sur un ou plusieurs axes.

Syntaxe de la fonction TRAJ :

TRAJ (<Paramètre>=<Valeur> {, <Paramètre>=<Valeur>})

Exemple :

```
TRAJ( POS(X)=500, VEL(X)=2000, POS(Y)=POS_S(Y)+200, ACC(Y)=500)
```

Cet exemple est équivalent à :

```
VEL(X)=2000
STTA(X=500)
ACC(Y)=500
STTA(Y=POS_S(Y)+200)
WAIT (NOT MOVE_S(X)) AND (NOT MOVE_S(Y))
```

L'instruction TRAJ est bloquante pour la tâche tant que tous les mouvements ne sont pas terminés.

Si l'instruction MERGE est activée et que l'on charge plusieurs TRAJ dans la carte d'axe, les mouvements seront exécutés les uns après les autres sans passer par une vitesse nulle. Par exemple :

```
MERGE(X)=On
TRAJ(POS(X)=500,VEL(X)=2000)
TRAJ(POS(X)=1000,VEL(X)=50)  'passage en petite vitesse à la position 500
```

5-6-2- Mouvements relatifs

A) Départ de mouvement : STTR

Pour lancer un mouvement vers une position relative et ne pas attendre sa fin pour poursuivre l'exécution de la tâche, on doit utiliser STTR. Cette instruction est très utile si la vitesse ou la position à atteindre doit changer en cours de mouvement. Avec cette fonction, l'erreur absolue est minimale.

Cette instruction est non bloquante pour la tâche (excepté si le buffer de mouvements est plein).

Elle utilise les valeurs courantes d'accélération, de décélération et de vitesse. La syntaxe est :

STTR (<Axe1>=<Distance1> [{, <Axe2>=<Distance2>}])

Par exemple :

```
VEL%(X)=100                 ' Vitesse rapide
P!=POS_S(X)
STTR(X=2000)                ' Start relatif en position 2000
WAIT  (POS_S(X)-P!) >100    ' Attente position +100
VEL(X)%=10                  ' Vitesse lente
WAIT  NOT  MOVE_S(X)        ' Attente fin de mouvement
```

Dans cet exemple, pendant un mouvement, la vitesse peut être modifiée car l'exécution du programme n'est pas arrêtée.

Si l'instruction MERGE est activée et que l'on charge plusieurs STTR dans la carte d'axe, les mouvements seront exécutés les uns après les autres sans passer par une vitesse nulle.

B) Mouvement : MOVR

La fonction MOVR envoie un ou plusieurs axes à une position relative. Elle utilise les valeurs courantes d'accélération, de décélération et de vitesse. La syntaxe est :

MOVR (<Axe1>=<Distance1> [{, <Axe2>=<Distance2>}])

Cette fonction envoie <Axe1> à une valeur relative <Distance1>. Le programme attend la fin du mouvement avant de continuer. L'erreur de positionnement absolue est minimale.

Par exemple :

```
FOR I#=1 To 10
  MOVR(X=100)
  CALL Percage
NEXT I#
```

L'instruction MOVR est bloquante pour la tâche tant que le mouvement n'est pas terminé (condition MOVE_S(Axe)=0).

MOVR(X=100) est équivalent à STTR (X=100)

WAIT NOT MOVE_S(X)

5-6-3- Mouvements infinis

Pour lancer un mouvement continu, il faut utiliser l'instruction STTI. Les axes concernés se déplacent à leur vitesse courante.

Cette instruction est non bloquante pour la tâche (excepté si le buffer de mouvements est plein).

L'instruction STOP ou SSTOP est nécessaire pour arrêter un mouvement continu. Le sens de déplacement est défini par le caractère "+" ou "-"

Syntaxe :

STTI (<Axe1>=<Sens1> {,<Axe2>=<Sens2>...})

Exemple :

```
WAIT  INP(BoutonPlus)=On
STTI(X=+)
WAIT  INP(BoutonPlus)=Off
STOP(X)
```

5-6-4- Arrêt d'un mouvement

Pour arrêter un mouvement, il faut utiliser les instructions STOP ou SSTOP. Elles arrêtent les axes spécifiés via leur décélération programmée et elles vident le buffer de mouvement.

L'instruction STOP est bloquante pour la tâche tant que le mouvement n'est pas terminé (condition MOVE_S(Axe)=0) alors que SSTOP n'est pas bloquante.

Syntaxe : STOP (<Axe1> {,<Axe2>...})

Exemple : déplacement jusqu'à un capteur.

```
STTI(X=+)
WAIT  INP(Capteur)=On
STOP(X)
```

L'instruction AXIS(Axe)=Off arrête aussi le mouvement mais sans aucun contrôle car l'asservissement est inhibé.

5-7- Synchronisation

5-7-1- Arbre électrique

A) Arbre électrique : GEARBOX

La fonction GEARBOX permet de réaliser un arbre électrique entre un axe maître et un axe esclave.

Syntaxe :

GEARBOX(<Esclave>, <Maître> , <Numérateur>, <Dénominateur>, <Réversible>,

[<Accélération>])

Dans une fonction de synchronisation de type arbre électrique, l'axe <Maître> peut être un axe servo ou un axe codeur. L'axe <Esclave> doit être un axe servo.

<Numérateur> / <Dénominateur> définit le rapport entre l'axe maître et l'axe esclave. Le rapport peut être positif ou négatif. Le signe de <Numérateur> indique le sens normal de l'esclave et <Dénominateur> le sens normal du maître.

Si la synchronisation doit être réversible, <Réversible> doit être mis à 1.

↳ mode non réversible : si le maître se déplace à l'inverse de son sens normal, l'esclave s'arrête ; il repartira lorsque le maître reprendra son sens normal et atteindra la position maître à laquelle l'esclave s'était arrêté.

↳ mode réversible : l'esclave suit le maître dans les deux sens.

<Accélération> est un paramètre optionnel. Il représente une distance maître pendant laquelle l'esclave sera en phase d'accélération avant d'atteindre le rapport de vitesse souhaité.

↳ S'il est laissé à 0, lors de l'exécution de l'instruction GEARBOX, le maître devra être à l'arrêt.

↳ S'il est utilisé, le maître pourra déjà être en mouvement lors de l'exécution de GEARBOX.

Si la synchronisation doit être réversible, <Réversible> doit être vraie.

Le paramètre <Accélération> est optionnel est permet la définition ou non d'une accélération lors de la réalisation du mouvement.

Cette instruction est non bloquante pour la tâche (excepté si le buffer de mouvements est plein).Tant que la liaison entre le maître et l'esclave ne sera pas coupée, l'instruction MOVE_S(Esclave) sera égale à 1 (même si l'esclave est arrêté).

Attention :

↳ Si le mode réversible à été sélecté et une valeur d'<accélération> programmée, si le maître change de sens alors que l'esclave est dans la phase <accélération>, l'esclave changera également de sens jusqu'à atteindre sa position initiale de départ. Là, l'esclave s'arrête ; il repartira lorsque le maître reprendra son sens initial et atteindra la position maître à laquelle l'esclave s'était arrêté.

↳ Le paramètre <Accélération> ne peut être utilisé que sur une carte SRV 85

↳ La réversibilité est infini sur une carte SRV 85. Elle est limitée à quelques tours moteur sur les autres cartes servo (SRV 15, SRV 1524, SSI 15)

Exemple : L'axe esclave tournera 2 fois plus vite que le maître et en sens inverse.

```
GEARBOX (Esclave, Maître, -2, 1, True)
...
STOP(Esclave)
```

B) Arbre électrique multi-maître : GEARBOXM

Cette instruction est équivalente à GEARBOX mais en plus, le maître correspond à la résultante de 2 sources maître. Ceci peut être utilisé pour gérer des applications intégrant des fonctions de différentielle.

C) Modification du rapport de réduction d'un arbre électrique :GEARBOXRATIO

Cette instruction permet de modifier le rapport de réduction d'une liaison arbre électrique. Sa syntaxe est la suivante : **GEARBOXRATIO**(<AxeEsclave>,<Ratio>).

<AxeEsclave> : doit être un axe servo esclave d'une liaison arbre électrique. Le rapport de l'arbre est défini par <Ratio>×<Numérateur>/<Dénominateur>. <Numérateur> et <Dénominateur> sont les paramètres de l'instruction GEARBOX.

L'instruction est non-bloquante et permet de changer de ratio sans arrêter l'arbre électrique. Le ratio peut être positif ou négatif.

```
Exemple  : GEARBOX(Esclave, Maître, 1, 2, 1)  'Rapport nominal  : ratio 0.5
           GEARBOXRATIO(Esclave,1.2)          '20% d'augmentation
                                              'Ratio  : (1.2×1/2)=0.6
           GEARBOXRATIO(Esclave,0.8)          '20% de réduction
                                              'Ratio  : (0.8×1/2)=0.4
```

5-7-2- Mouvements synchronisés

A) Mouvement : MOVS

L'instruction MOVS permet d'effectuer une synchronisation entre un axe esclave et un maître.

Cette instruction est non bloquante pour la tâche (excepté si le buffer de mouvements est plein).

Exemple :

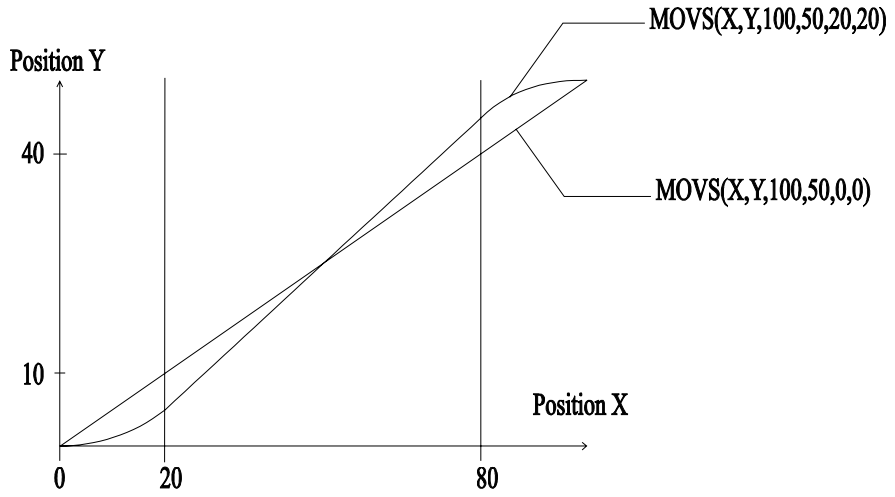
```
MOVS(Y, X, 10, 20, 0, 0 )    'pour un déplacement relatif  de 10 unités
                             'sur X, Y se déplace de 20
```

Syntaxe : MOVS(<Esclave>, <Maître>, <Dist .maître>, <Dist. esclave>, <Dist. d'accél.>,

<Dist. de décél.>)

Elle est utilisée pour synchroniser l'axe esclave avec l'axe maître pendant une distance précise de l'axe maître, avec séparément des phases variables d'accélération et de décélération sur l'axe esclave. L'axe maître peut être un axe servo ou un axe codeur. L'axe esclave doit être un axe servo.

Par exemple :



Cet exemple représente deux mouvements synchronisés avec et sans les phases d'accélération et de décélération. Quand il n'y a pas de phases d'accélération et décélération, l'axe maître et l'axe esclave doivent avoir la même vitesse pour réduire les phases transitoires. Si les vitesses sont très différentes, les phases d'accélération et de décélération doivent être spécifiées afin d'assurer un comportement mécanique correct.

Les vitesses ne sont pas nécessairement les mêmes et dépendent des phases d'accélération et de décélération. Ceci est causé par la nécessité du respect des distances.

Les fonctions LOADS et STARTS sont également utilisées pour réaliser des mouvements synchronisés. Les phases de calcul du mouvement et de lancement de la synchro sont alors dissociées de façon à permettre le démarrage simultané de plusieurs axes en synchro sur le même axe maître.

Exemple LOADS et STARTS :

```
LOADS(X, Y, 10, 20, 0, 0 )   'Chargement de la synchro pour y
LOADS(X, Z, 10, 40, 0, 0 )   'Chargement de la synchro pour z
STARTS(Y,Z)                  'Lancement simultané des deux synchros
```

B) Mouvement déclenché sur position : MOVSP

Cette instruction est équivalente à MOVS mais en plus, elle intègre une condition de déclenchement donnée par la position d'un autre axe.

C) Mouvement multi-maîtres MOVSM

Cette instruction est équivalente à MOVS mais en plus, le maître correspond à la résultante de 2 sources maître. Ceci peut être utilisé pour gérer des applications intégrant des fonctions de différentielle.

D) Mouvement déclenché sur entrée capture : MOVSC (seulement sur SRV 85)

Cette instruction est équivalente à MOVS mais en plus, elle intègre une condition de déclenchement donnée par une entrée rapide « capture ».

Tant que la liaison entre le maître et l'esclave ne sera pas coupée, l'instruction MOVE_S(Esclave) sera égale à 1 (même si l'esclave est arrêté).

5-7-3- Came

A) Introduction

a) Définition de la came

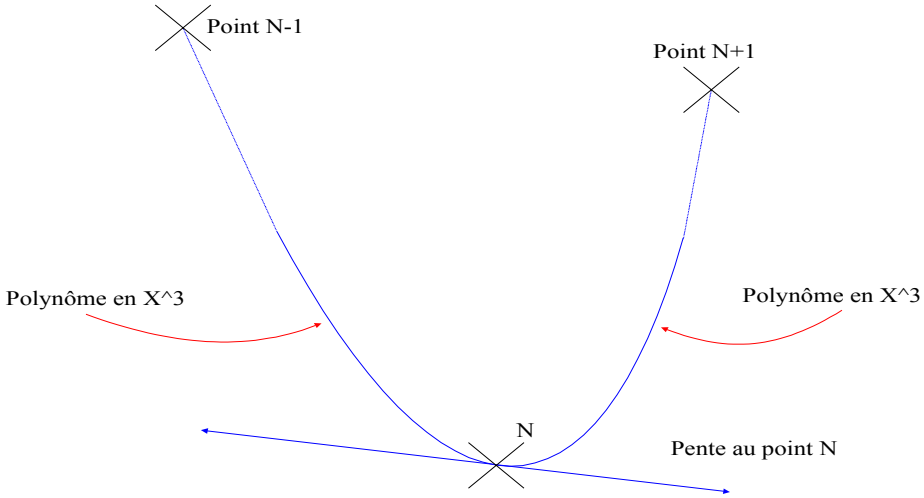
La fonction came permet de réaliser un profil de came sur un axe esclave lié à un axe maître. Ce profil est défini par un tableau de points. Chaque carte servo peut stocker jusqu'à 5 cames et 310 points pour l'ensemble des 5 cames.

Chaque point est représenté par une position de l'axe maître et une position de l'axe esclave.

La MCS exécutera entre deux points consécutifs une trajectoire calculée à partir d'un polynôme du 3ème degré.

Les valeurs données aux positions de l'axe maître à l'intérieur de la table doivent être des valeurs croissantes.

Pour que la MCS puisse connaître les pentes d'entrée et de sortie de la came, on doit fournir au tableau deux points supplémentaires : le point d'entrée, le point de sortie. Ils peuvent être directement calculés par le logiciel MCB suivant le niveau de programmation choisi.



La pente au point N est la pente de la droite qui relie les points N-1 et N+1.

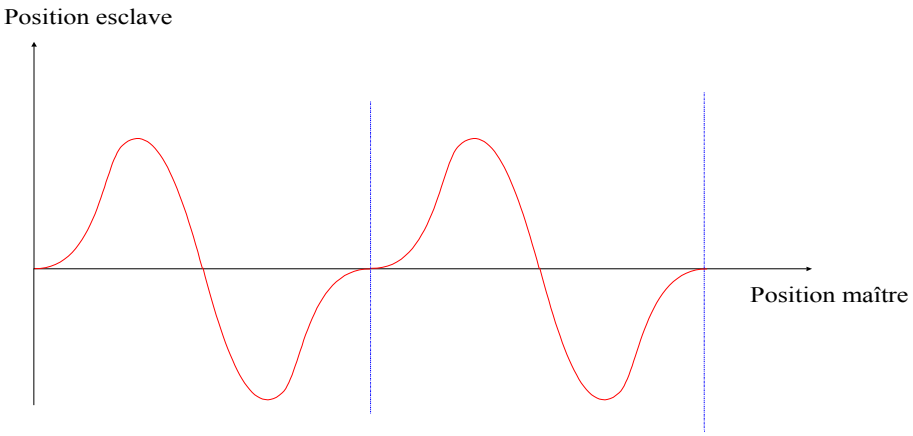
b) Came finie ou came infinie

Une came mécanique correspond à une came électronique finie. Dans le tableau de points, la 1ère et la dernière valeur de la position de l'esclave sont confondues. Le mouvement de l'esclave sera un mouvement linéaire d'amplitude finie.

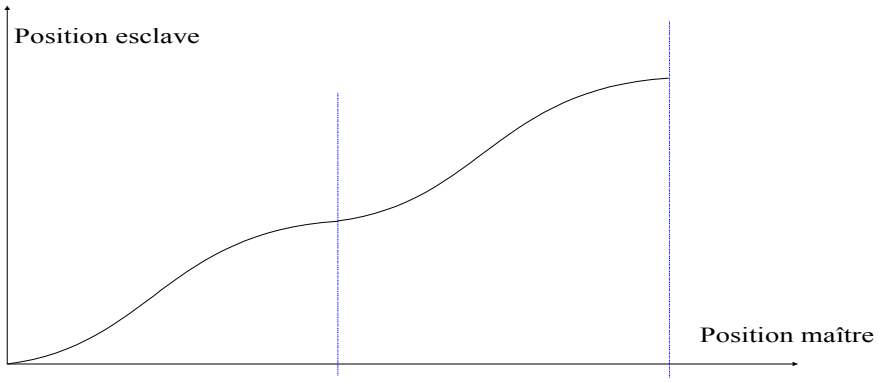
La came électronique permet également de créer un mouvement sur l'axe esclave de type rotatif infini : la position absolue de l'esclave augmente à chaque cycle maître.

Attention : Si l'axe maître ou l'axe esclave est un axe infini, il doit être déclaré en axe modulo à partir de l'onglet configuration du logiciel MCB.

Came finie



Came infinie



c) Principe de lancement d'une came

Les valeurs des coordonnées des différents points sont rentrées sous forme de valeurs absolues. Cela permet dans certains cas, de faire directement l'étalonnage entre la came mécanique et la came électronique.

Lors du lancement de l'exécution de la came, la MCS considère qu'elle se trouve déjà sur le 1er point du tableau et elle exécute les points suivants en relatif. Par exemple :

Table de came

Point	Maître	Esclave
1	0°	180°
2	20°	230°
3	50°	250°
....	....	....

Avant l'exécution de la came , l'axe maître est à la position absolue 5° et l'esclave à la position absolue 10°. Les deux axes sont à l'arrêt.

- On lance l'exécution de la came : l'axe esclave reste à l'arrêt
- On avance l'axe maître en relatif de 20 – 0 = 20° (position absolue = 5 + 20 = 25°)
- L'axe esclave avance de 230 - 180 = 50° (position absolue = 10 + 50 = 60°)
- On avance l'axe maître en relatif de 50 – 20 = 30° (position absolue = 5 + 20 + 30 = 55°)
- L'axe esclave avance de 250 - 230 = 20° (position absolue = 10 + 50 + 20 = 80°)

B) 1er Niveau : programmation simple

Le niveau 1 permet de lancer un profil de came sur un axe de façon très simple. On utilise pour cela l'instruction CAM

Ce profil est réalisé à partir d'un tableau de points créé à l'aide de l'éditeur de données accessible à partir de l'onglet « variables globales » du logiciel MCB (variable de type « tableau de came »).

Chaque point est représenté par une position de l'axe maître et une position de l'axe esclave.

Après avoir rentré tous les points, les pentes d'entrée et de sortie de la came sont calculées automatiquement par un simple appui sur le bouton « Calcul » ; les points d'entrée et de sortie du tableau prennent alors les valeurs calculées en fonction des critères came mono-coup ou non, came finie ou infinie.

La syntaxe de la fonction CAM est la suivante :

CAM(<Esclave>, <Maître>, <Tableau>, <Mono-coup>, <Réversible>,

<DirectionPositive>,<Gain>)

<Esclave> : Nom de l'axe esclave où est effectuée la came (carte servo : SRV15, SRV85...)

<Maître> : Nom de l'axe maître (carte servo ou codeur : SCD22, SRV15, SRV85...)

<Tableau> : Nom du tableau de came rentré à partir de l'éditeur de données de l'onglet variables globales du logiciel MCB (variable de type « tableau de came »).

Les valeurs données aux positions de l'axe maître à l'intérieur de la table doivent être des valeurs croissantes.

Pour que la MCS puisse connaître les pentes d'entrée et de sortie de la came, on doit intégrer dans le tableau deux points supplémentaires : le point d'entrée, le point de sortie. A partir de l'éditeur de données, indiquez si la came est mono-coup ou non, finie ou infinie et appuyez sur le bouton « Calcul des points d'entrée et sortie ».

<Mono-coup> : Définit le rebouclage automatique de la came.

Rentrez la valeur 0 pour une came qui va se reboucler sur son profil jusqu'à ce qu'un arrêt soit demandé, 1 pour une came qui va exécuter son profil une seule fois.

<Réversible> : Indique si l' <Esclave> doit suivre le <Maitre> dans les deux sens.

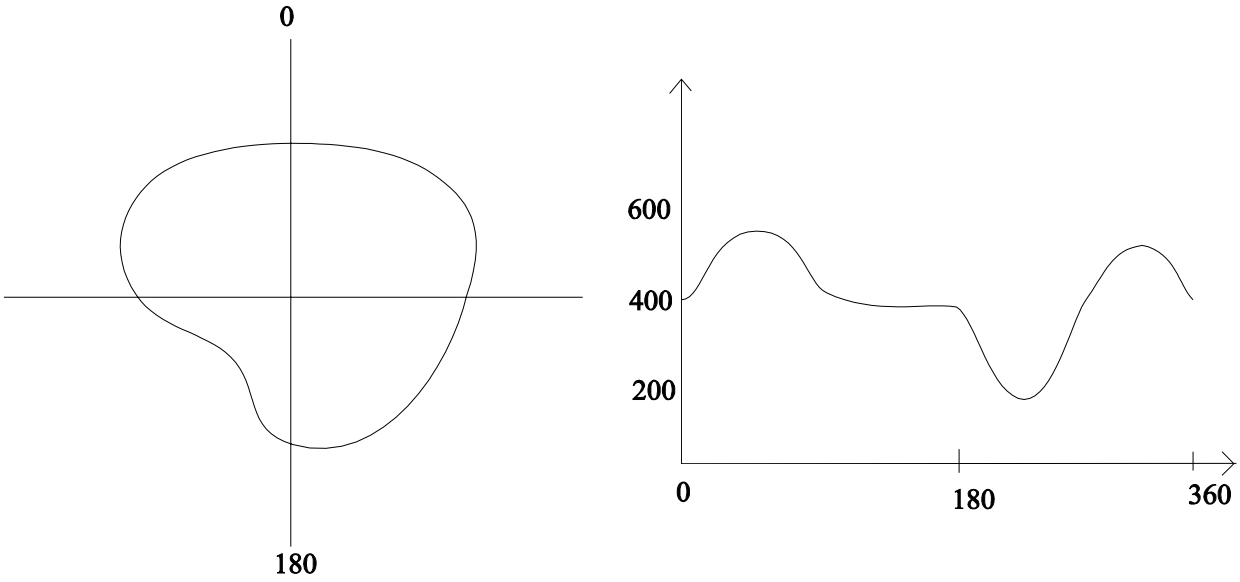
↳ Rentrez la valeur 0 pour une came non réversible : si le maître se déplace à l'inverse de son sens normal donné par <DirectionPositive>, l'esclave s'arrête ; il repartira lorsque le maître reprendra son sens normal et atteindra la position maître à laquelle l'esclave s'était arrêté.

↳ Rentrez la valeur 1 pour une came réversible : l'esclave suit son profil de came quel que soit le sens d'avance du maître.

<DirectionPositive> : Si la came n'est pas réversible, le sens normal de l'avance du maître doit être indiqué. Rentrez la valeur 0 pour un sens négatif, 1 pour un sens positif.

<Gain> : Le <Gain> est utilisé pour multiplier toutes les positions sur l'axe <Esclave> définies dans le tableau par un facteur.

Exemple :



Dans cet exemple, le tableau de came définit le profil de came représenté. Une simple ligne dans le programme permet de lancer la came.

```
.....
CAM (Esclave,Maître,TableCame, 0, 1, 0, 1)      ' non mono-coup
...                                             ' (rebouclée), réversible
WAIT INP(InfoStop)                              ‘ Attente demande d’arrêt
ENDCAM(Esclave)                                 ' Demande d’arrêt en fin
                                                ' du profil courant
```

Index	Angle axe maître	Position axe esclave
1	0	400
2	40	550
3	80	410
4	120	400
5	160	400
6	180	400
7	240	200
8	280	400
9	320	550
10	360	400

C) 2ème Niveau : programmation avancée

Le niveau 2 permet de lancer un profil de came dont le tableau de points a pu être rentré ou modifié à partir d'une tâche basic de la MCS (par exemple, tableau saisi à l'aide du terminal Dialog 640 ...).

On peut également enchaîner des profils de cames différents.

a) Création d'un tableau de réels

Dans le projet MCB, il est nécessaire de déclarer un tableau de réels qui aura la structure suivante :

élément 1 = Point d'entrée du maître

élément 2 = Point d'entrée de l'esclave

élément 3 = Premier point du maître

élément 4 = Premier point de l'esclave

....

élément w = Dernier point du maître

élément x = Dernier point de l'esclave

élément y = Point de sortie du maître

élément z = Point de sortie de l'esclave

Le tableau de réels contient (NombrePointsCame + 2) *2 éléments. Lors de la déclaration du tableau à partir du MCB, il est judicieux de rentrer un nombre plus important afin de pouvoir rajouter des points si nécessaire.

Par exemple, si on déclare un tableau de came de 10 éléments à l'adresse 100, il sera constitué de (10 + 2) * 2 = 24 variables réelles d'adresse 100 à 123 inclus.

adresse 100 : point d'entrée maître	adresse 101 : point d'entrée esclave
adresse 102 : point 1 maître	adresse 103 : point 1 esclave
.....	.....
adresse 121 : point 10 maître	adresse 122 : point 10 esclave
adresse 122 : point de sortie maître	adresse 123 : point de sortie esclave

b) Principe de calcul des points d'entrée et de sortie

Dans le niveau de programmation 2, c'est au programmeur de calculer les points d'entrée et de sortie de la came.

↳ Came courante Cc enchaînée sur elle-même ou sur une came précédente Cp ou suivante Cs :

PointEntréeMaîtreCc = PremierPointMaîtreCc – (DernierPointMaîtreCp – AvantDernierPointMaîtreCp)

PointEntréeEsclavCc = PremierPointEsclavCc – (DernierPointEsclavCp – AvantDernierPointEsclavCp)

PointSortieMaîtreCc = DernierPointMaîtreCc + (DeuxièmePointMaîtreCs – PremierPointMaîtreCs)

PointSortieEsclavCc = DernierPointEsclavCc + (DeuxièmePointEsclavCs – PremierPointEsclavCs)

↳ Came courante Cc avec départ arrêté :

PointEntréeMaîtreCc = PremierPointMaîtreCc – (DernierPointMaîtreCc – AvantDernierPointMaîtreCc)

PointEntréeEsclavCc = DeuxièmePointEsclavCc

↳ Came courante Cc avec arrêt en fin de profil :

PointSortieMaîtreCc = DernierPointMaîtreCc + (DeuxièmePointMaîtreCc – PremierPointMaîtreCc)

PointSortieEsclavCc = AvantDernierPointEsclavCc

Par exemple :

Table de came

Point	Maître	Esclave
Entrée	Me	Ee
1	M1	E1
2	M2	E2
3	M3	E3
4	M4	E4
5	M5	E5
6	M6	E6
Sortie	Ms	Es

Attention : On doit avoir Me < M1 < M2 ... < Ms

Dans le cas d'une came seule, mono-coup (exécution 1 seule fois du profil ; départ arrêté ; arrêt en fin de profil) :

Me = M1 - (M6 - M5) <=> 1er point – (dernier point – avant dernier point)

Ms = M6 + (M2 – M1) <=> dernier point + (2ème point – 1er point)

Ee = E2 <=> 2ème point

Es = E5 <=> avant dernier point

Dans le cas d'une came seule, non mono-coup (exécution n fois du même profil) :

Me = M1 - (M6 - M5) <=> 1er point – (dernier point – avant dernier point)

Ms = M6 + (M2 – M1) <=> dernier point + (2ème point – 1er point)

Ee = E1 – (E6 – E5) <=> 1er point – (dernier point – avant dernier point)

Es = E6 + (E2 – E1) <=> dernier point + (2ème point – 1er point)

c) Chargement d'une came

Pour charger une came dans la carte servo, utilisez l'instruction LOADCAMEX.

Sa syntaxe est la suivante :

LOADCAMEX(<Esclave>,<Maître>,<N°came>,<Tableau>,<Nombre>,
<PremierPolynôme>,<Mono-coup>, <Réversible>,<DirectionPositive>,
<N°came suivante>,<N°came précédente>)

<Esclave> : Nom de l'axe esclave où est effectuée la came (carte servo : SRV 15, SRV 85 ...)

<Maître> : Nom de l'axe maître (carte servo ou codeur : SCD 22, SRV 15, SRV 85 ...)

<N°came> : numéro de la came (de 1 à 5)

<Tableau> : Nom du tableau de came déclaré à partir de l'onglet variables globales du logiciel MCB (variable de type « réel »).

<Nombre> : nombre d'éléments du tableau pour définir la came.

Nombre = (NombrePointsCame + 2) *2

<PremierPolynôme> : Une carte servo possède une table globale de 310 polynômes pour l'ensemble des 5 cames. Rentrez une valeur entre 1 et 310 pour indiquer où sera stocké le 1er polynôme de la came dans la table globale de la carte.

Du tableau de came est extrait un tableau de (NombrePointsCame-1) polynômes.

☏ Attention : <PremierPolynôme> + <NombrePointsCame-1> doit être inférieur à 310.

<Mono-coup> : Définit le rebouclage automatique de la came.

Rentrez la valeur 0 pour une came qui va se reboucler sur son profil jusqu'à ce qu'un

arrêt soit demandé, 1 pour une came qui va exécuter son profil une seule fois.

<Réversible> : Indique si l' <Esclave> doit suivre le <Maitre> dans les deux sens.

↳ Rentrez la valeur 0 pour une came non réversible : si le maître se déplace à l'inverse de son sens normal donné par <DirectionPositive>, l'esclave s'arrête ; il repartira lorsque le maître reprendra son sens normal et atteindra la position maître à laquelle l'esclave s'était arrêté.

↳ Rentrez la valeur 1 pour une came réversible : l'esclave suit son profil de came quel que soit le sens d'avance du maître.

<DirectionPositive> : Si la came n'est pas réversible, le sens normal de l'avance du maître doit être indiqué.Rentrez la valeur 0 pour un sens négatif, 1 pour un sens positif.

<N°came suivante> : Mettez 0 si la came ne doit pas être enchaînée sur une autre came. Dans le cas contraire, rentrez le numéro de la came suivante compris entre 1 et 5.

<N°came précédente> : Mettez 0 si la came n'enchaînera pas sur une came précédente. Dans le cas contraire, rentrez le numéro de la came précédente compris entre 1 et 5.

d) Lancement d'une came

Pour lancer l'exécution d'une came, utilisez l'instruction STARTCAM.

Sa syntaxe est la suivante : STARTCAM(<Esclave>,<Maître>,<N°came>)

<Esclave> : Nom de l'axe esclave où est effectuée la came (carte servo : SRV15, SRV85...)

<Maître> : Nom de l'axe maître (carte servo ou codeur : SCD22, SRV15, SRV85...)

<N°came> : numéro de la came (de 1 à 5) de la carte servo <Esclave>.

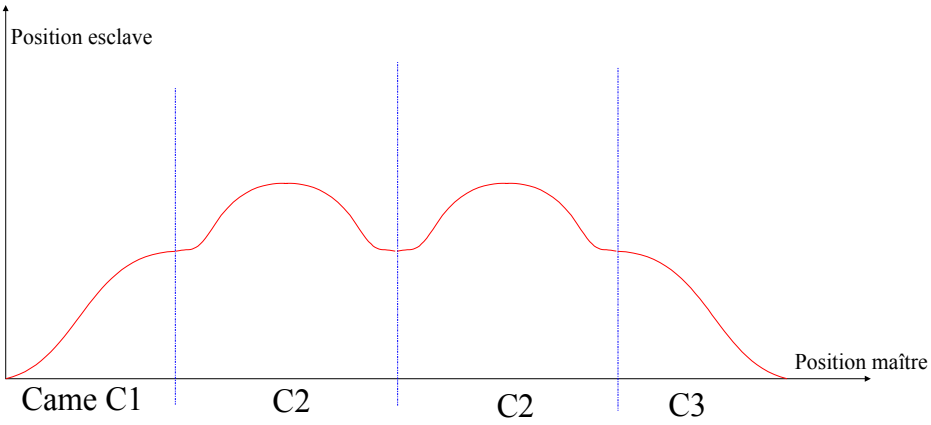
e) Exemple 1 : came de 6 points, infinie, non mono-coup.

```
PROG
.....
' Définition du tableau dans le tableau de réel TabCam
TabCam[3]=0           'point 1 maître
TabCam[4]=0           'point 1 esclave
TabCam[5]=30          'point 2 maître
TabCam[6]=20          'point 2 esclave
TabCam[7]=90          'point 3 maître
TabCam[8]=76          'point 3 esclave
TabCam[9]=270         'point 4 maître
TabCam[10]=283        'point 4 esclave
TabCam[11]=330        'point 5 maître
TabCam[12]=338        'point 5 esclave
TabCam[13]=360        'point 6 maître
TabCam[14]=360        'point 7 esclave
TabCam[1]=TabCam[3]-(TabCam[13]-TabCam[11])      'point d'entrée maître
TabCam[2]=TabCam[4]-(TabCam[14]-TabCam[12])      'point d'entrée esclave
TabCam[15]=TabCam[13]+TabCam[5]-TabCam[3]        'point de sortie maître
TabCam[16]=TabCam[14]+TabCam[6]-TabCam[4]        'point de sortie esclave
' Chargement de la came n°1
LOADCAMEX(Esclave,Maître,1,TabCam,16,1,0,1,0,0,0)
' Lancement de la came n°1
STARTCAM(Esclave,Maître,1)
.....
WAIT INP(InfoStop)                          ' Attente demande d'arrêt
ENDCAM(Esclave)                             ' Demande d'arrêt en fin
.....                                       ' du profil courant
END PROG
```

f) Exemple 2 : enchaînement de cames

Soit un cycle composé de 3 cames : une came C1 de « profil d'entrée » mono-coup, une came C2 de « profil répétitif »non mono-coup, une came C3 de « profil de sortie » mono-coup.

La came C1 est enchaînée à C2 et C2 est enchaînée à C3.




```
PROG
.....
' Chargement de la came n°1  : 10 points, mono-coup, enchaînée sur came n°2
LOADCAMEX(Esclave,Maître,1,TabCam1,24,1,1,1,0,2,0)
' Chargement de la came n°2  : 36 points, non mono-coup, enchaînée sur came
' n°3
LOADCAMEX(Esclave,Maître,2,TabCam2,76,100,0,1,0,3,1)
' Chargement de la came n°3  : 6 points, mono-coup
LOADCAMEX(Esclave,Maître,3,TabCam3,16,200,1,1,0,0,2)
' Lancement de la came n°1 => exécution de la came1 puis de la came2
STARTCAM(Esclave,Maître,1)
WAIT CAMNUM_S(Esclave)=2          ' Attente exécution de la came2
.....
WAIT INP(InfoStop)               ' Attente demande d'arrêt
ENDCAM(Esclave)                  ' Arrêt came2 en fin profil
                                 ' et enchaînement sur came3
WAIT NOT CAM_S(Esclave)          ' Attente came 3 terminée
.....
END PROG
```

D) Etat de la came

Trois fonctions permettent de connaître l'état courant d'une carte servo pilotant une came.

↳ Instruction CAM_S : permet de savoir si une came de la carte est en cours d'exécution.

Exemple :

```
IF NOT CAM_S(Esclave) THEN PRINT «  Came arrêtée  »
IF CAM_S(Esclave) THEN PRINT «  Came Lancée  »
```

↳ Instruction CAMNUM_S : permet de savoir quel numéro de came est en cours d'exécution. La valeur retournée est significative que si CAM_S est à 1.

Exemple :

```
IF CAMNUM_S(Esclave)=1 THEN PRINT «  Came 1 en cours  »
IF CAMNUM_S(Esclave)=2 THEN PRINT «  Came 2 en cours  »
```

↳ Instruction CAMSEG_S : permet de savoir quelle numéro d'équation de la came est en cours d'exécution. La valeur retournée est significative que si CAM_S est à 1.

Exemple :

```
IF CAMSEG_S(Esclave)=1 THEN PRINT «  Came entre le point 1 et le point 2  »
IF CAMSEG_S(Esclave)=2 THEN PRINT «  Came entre le point 2 et le point 3  »
```

E) Arrêt d'une came

La fonction ENDCAM permet d'arrêter le mouvement de l'esclave à la fin du profil de la came tandis que la fonction STOP met fin au mouvement immédiatement. La syntaxe de l'instruction ENDCAM est la suivante : ENDCAM (<Axe>)

Attention : Si ENDCAM s'applique à une came qui a été déclarée en mode non mono-coup et enchaînée avec une autre, la came termine son profil et enchaîne sur la suivante.

F) Mise en garde

↳ Les valeurs données aux positions de l'axe maître à l'intérieur de la table doivent être des valeurs croissantes.

↳ L'écart entre deux points successifs du maître ou de l'esclave ne doit pas dépasser +/- 2^21 incréments.

↳ Cet écart ne doit pas être trop petit. Il est préférable que le système passe par tous les points successifs, même à vitesse maximale (période d'échantillonnage de 330µs).

G) Mouvement déclenché sur entrée capture : CAMC (seulement sur SRV 85)

Cette instruction est équivalente à CAM mais en plus, elle intègre une condition de déclenchement donnée par une entrée rapide « capture ».

H) Modification de points d'une came : LOADPOINT

Cette instruction permet de changer deux polynômes à partir d'un point d'une came dans la carte servo. Sa sintaxe est la suivante :

LOADPOINT(<Esclave>,<Maître>,<Tableau>,<Nombre>,<PremierPolynôme>,

<IndexVariable)

<Esclave> : Nom de l'axe esclave où est effectuée la came (carte servo : SRV15, SRV85...)

<Maître> : Nom de l'axe maître (carte servo ou codeur : SRV15, SRV85, SCD22...)

<Tableau> : Nom du tableau de came déclaré à partir de l'onglet variables globales du logiciel MCB (variable de type « réel »).

<Nombre> : nombre d'éléments du tableau pour définir la came.

Nombre = (NombrePointsCame + 2) × 2

<PremierPolynôme> : Une carte servo possède une table globale de 310 polynômes pour l'ensemble de 5 cames. Rentrez une valeur entre 1 et 310 pour indiquer où sera stocker le premier polynôme de la came dans la table globale de la carte.

<IndexVariable> : Index de la variable dans le <Tableau> où se situe la modification à prendre en compte. La modification des polynômes tient compte des positions du maître et de l'esclave. Il n'est pas nécessaire d'utiliser LOADPOINT deux fois pour modifier ces deux informations.

5-7-4- Fonction de compensation / décompensation (carte SRV 85 seulement)

Les fonctions de compensation / décompensation permettent d'effectuer un déphasage dynamique sur un axe esclave lié à un axe maître. La liaison entre les deux axes doit être de type arbre électrique (GEARBOX, GEARBOXM), synchronisation (MOVS, MOVSM, MOVSP, MOVSC) ou came (CAM, CAMC).

A) Fonction de compensation immédiate : ICORRECTION (SRV 85)

Cette fonction permet d'appliquer un mouvement de correction sur un axe esclave pendant une distance de l'axe maître.

L'esclave devra au préalable être lié à un maître par une fonction d'arbre électrique ou de mouvement synchronisé.

Au mouvement de synchronisation normal de l'esclave, on superpose le mouvement suivant :

Pendant que le maître parcourt une « distance maître », on ajoute un déplacement « distance esclave » avec une accélération et une décélération sur une distance maître de « distance d'accél ».

Syntaxe :

ICORRECTION(<Esclave>,<Maître>,<Dist.maître>,<Dist.esclave>,<Dist. d'accél>)

<Esclave> : Nom de l'axe esclave où est effectuée la compensation (SRV 85 seulement)

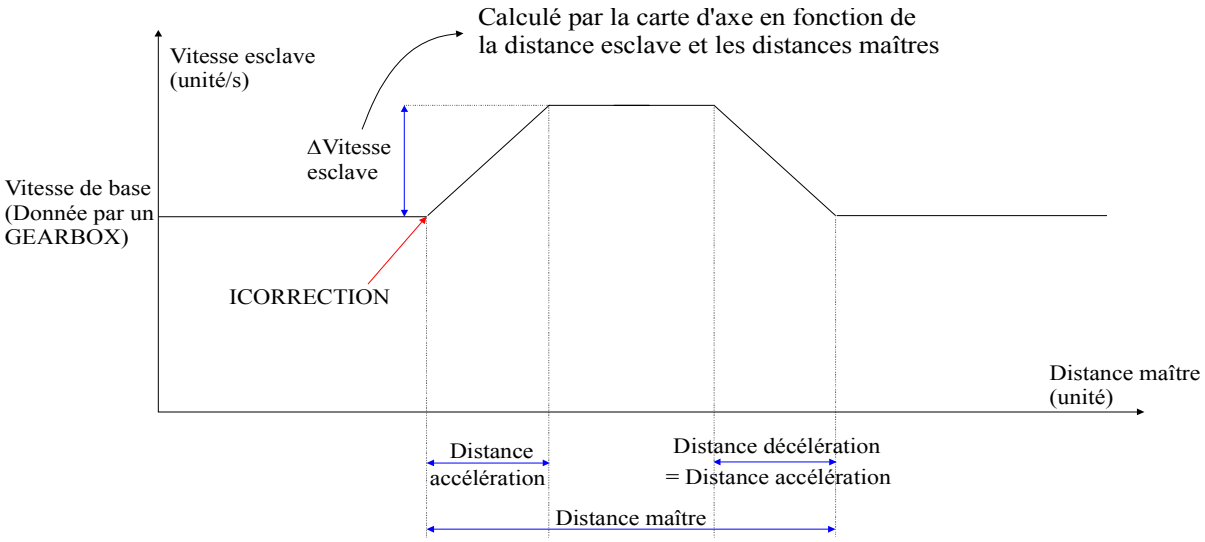
<Maître> : Nom de l'axe maître (carte servo ou codeur : SCD 22, SRV 15, SRV 85 ...)

<Dist.maître>: Intervalle de correction (distance de déplacement du maître pendant la correction).

Si la distance est positive, la correction sera appliquée si le maître avance en sens positif. Si elle est négative, la correction sera appliquée si le maître avance en sens négatif.

<Dist.esclave> : Correction (distance de correction apportée à l'esclave). Si le signe de la correction est positif, on effectuera une compensation, s'il est négatif on effectuera une décompensation.

<Dist.d'accél> : Accélération / décélération (portion d'accélération ou de décélération de l'esclave sur une distance du maître).

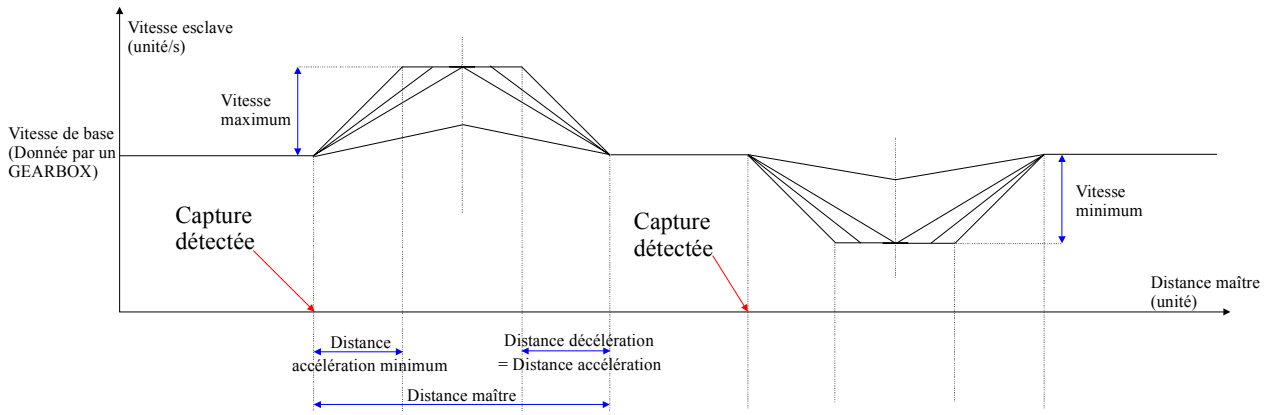



```
Exemple  :
.....
GEARBOX(Esclave,Maître,1,1,1)
STTI(Maître=+)
.....
```

```
CAPTURE2(Maître,24,Off,0,0,Off)     ' Capture position axe maître sur front
                                    ' descendant de l'entrée C2 de l'axe
                                    ' maître
WAIT REG2_S(Maître)=1               '  Attente détection capture
CorrectionValue!=Phase-REGPOS2_S(X)
ICORRECTION(Esclave,Maître,Intervalle!,CorrectionValue!,Dacc  !)
.....
```

B) Fonction de compensation : CORRECTION (SRV 85)

Cette fonction permet d'appliquer un mouvement de correction sur un axe esclave pendant une distance de l'axe maître sur un événement de capture de position. Comme tous les paramètres sont stockés dans la carte de l'axe esclave, dès que l'événement est détecté, la compensation est traitée instantanément.



a) Lancement de la compensation

La compensation peut être déclenchée sur 2 modes de capture :

↳ capture de la position de l'axe maître à partir d'une entrée capture de l'axe esclave (précision capture : 330µs).

↳ capture effectuée physiquement sur l'axe maître et utilisation de la capture diffusée sur l'axe esclave (précision capture : 0.1µs).

Gabarit de compensation :

Syntaxe :

CORRECTION(<Esclave>,<Maître>,<Mode>,<Capture>,<Configuration>,<Fenêtre>,<Mini>, <Maxi>,<Intérieur>,<Phase>,<Dist.maître>,<Vit.maxi esclave>,<Vit.mini esclave> ,<Dist.accél.mini>)

<Esclave> : Nom de l'axe esclave où est appliquée la compensation (SRV 85 seulement)

<Maître> : Nom de l'axe maître (carte servo ou codeur : SCD 22, SRV 15, SRV 85 ...)

<Mode> : 0 pour mono-coup (une seule compensation par exécution d'instruction CORRECTION), 1 pour permanent (une compensation appliquée à chaque capture tant que l'instruction STOPCORRECTION n'a pas été rencontrée).

<Capture> : 0 pour capture de la position maître à partir du registre1 de l'axe maître (précision 0.1µs), 1 pour capture de la position maître à partir du registre1 de l'axe esclave (précision 330µs).

<Configuration> : Octet de configuration hardware de la capture

b0 : non utilisé

b1 : capture sur top Z

b2 : capture sur entrée n°1 C1

b3 : capture sur entrée n°2 C2

b4 : choix du front : 0 pour front montant, 1 pour front descendant

b5..7 : non utilisés

<Fenêtre> : activation d'une fenêtre bornée par les positions <Mini> et <Maxi> de <Maître>.

<Interieur> permet de définir si le test s'effectue à l'intérieur ou à l'extérieur des bornes. <Mini> doit toujours être inférieur à <Maxi>.

<Phase> : phase théorique Si la valeur de position capturée est égale à la phase, aucune compensation ne sera appliquée.

<Dist.maître> : Intervalle de correction (distance de déplacement du maître pendant la correction)

<Vit.maxi esc.>: Vitesse maxi ajoutée à la vitesse courante de l'esclave pour appliquer la compensation. Elle est exprimée en unité/s (exemple : Vcourante=1m/s, Vmaxi=0.5m/s => Vcorrection maxi = 1.5m/s).

<Vit.mini esc.>: Vitesse mini retranchée à la vitesse courante de l'esclave pour appliquer la compensation. Elle est exprimée en unité/s (exemple : Vcourante=1m/s, Vmini=0.8m/s => Vcorrection mini = 0.2m/s).

<Dist.acc.mini> :Accélération / décélération mini (portion d'accélération ou de décélération de l'esclave sur une distance du maître).

Attention :

↳ La compensation précédente doit être complètement terminée avant de pouvoir en traiter une autre.

↳ Si le maître tourne en sens opposé à son sens normal (donné par le signe de « distance maître »), la compensation n'est pas appliquée et elle est perdue.

↳ L'esclave devra au préalable être lié à un maître par une fonction d'arbre électrique ou de mouvement synchronisé avant de lancer une compensation.

↳ Si la correction force à passer en dehors des limites définies par l'utilisateur, l'erreur est signalée et la correction maximale autorisée est appliquée.

b) Arrêt de la compensation

Pour arrêter la fonction de compensation déclarée en mode permanent, il faut utiliser l'instruction STOPCORRECTION.

Syntaxe : STOPCORRECTION (<Esclave>)

<Esclave>: Nom de l'axe esclave où est appliquée la compensation (SRV 85 seulement)

c) Etat de la compensation

Pour connaître l'état du cycle de compensation, utilisez l'instruction CORRECTION_S.

Syntaxe : <Variable>=CORRECTION_S(<Esclave>)

La valeur retournée est de type octet.

b0 : à 1, compensation demandée et en attente de l'événement capture.

b1 : à 1, compensation en cours.

b2 : à 1, erreur détectée : compensation hors limites (la compensation limitée à quand même été appliquée).

⇘ En mode rebouclé, la valeur retournée par CORRECTION_S varie entre « compensation en attente » et « compensation en cours » (| b0=1 b1=0 | b0=0 b1=1 | b0=1 b1=0 |...).

⇘ En mode mono-coup, elle passe par : | b0=1 b1=0 | b0=0 b1=1 | b0=0 b1=0 |.

Le bit d'erreur b2 est automatiquement remis à 0 après chaque lecture de CORRECTION_S ou au départ de la compensation suivante.

d) Exemple :

```
.....
GEARBOX(Esclave,Maître,1,1,1)
STTI(Maître=+)
.....                                ' Compensation monocoup
.....                                ' Condition de compensation  : front
.....                                ' montant sur entrée C1 de
.....                                ' l'axe esclave
CORRECTION(Esclave,Maître,0,1,4,0,0,0,0,Phase,
          Intervalle!,0.5,0.5,Intervalle!/4)
REPEAT                               ' Attente fin compensation
                                     ' ou erreur compensation
C#=CORRECTION_S
UNTIL (C#=0) OR (C#=4)
IF C#=0 THEN GOTO ERROR
.....
```

5-7-5- Fonction de superposition de mouvements (carte SRV 85 seulement)

La fonction ADDMOV permet de superposer sur un axe esclave ses propres mouvements avec ceux effectuer par un axe maître. La liaison entre les deux axes doit être de type arbre électrique (GEARBOX, GEARBOXM), synchronisation (MOVS, MOVSM, MOVSP, MOVSC) ou came (CAM, CAMC).

Syntaxe : ADDMOV (<Esclave>, <Maître>, <Coefficient>)

<Esclave> : Nom de l'axe esclave où est effectuée la superposition (SRV 85 seulement)

<Maître> : Nom de l'axe maître (carte servo ou codeur : SCD 22, SRV 15, SRV 85 ...)

<Coefficient> : Mise à l'échelle du maître et de l'esclave en incrément

```
Exemple :
⇘ axe maître : axe linéaire avec ENCODER_P=4000, UNITREV_P=5mm
⇘ axe esclave : axe linéaire avec ENCODER_P=4000, UNITREV_P=10mm
⇘ Coefficient=(ENCODER_P(Esc)*UNITREV_P(Maître))/(ENCODER_P(Maître)
              *UNITREV_P(Esc))
⇨ Coefficient=5/10=0.5
```

Dès que la fonction ADDMOV a été exécutée, le lien entre les 2 axes est établi jusqu'à rencontre de l'instruction ADDSTOP.

Syntaxe : ADDSTOP (<AxeEsclave>)

Si l'axe esclave passe en mode non asservi (AXIS(Esclave)=Off), le lien entre le maître et l'esclave est également coupé.

Le maître et l'esclave peuvent exécuter n'importe quel type de mouvement : positionnement, synchronisation, interpolation.

```
Exemple :
.....
MOVA(X=0,Y=0)
ADDMOV(Y,X,1)          ' activation du lien de superposition
STTI(X=+)              ' lancement axe X => axe Y se déplace également
WAIT INP(StartCycle)
MOVR(Y=StepCycle)      ' superposition d'un mouvement de distance StepCycle
.....
STOP(X)
ADDSTOP(Y)             ' arrêt du lien de superposition
.....
```

5-7-6- Arrêt d'une liaison maître / esclave

Pour arrêter une liaison maître / esclave, il faut utiliser les instructions STOP ou SSTOP sur l'axe esclave. Ces instructions arrêtent également l'axe spécifié via la décélération courante et elles vident son buffer de mouvement.

L'arrêt de la liaison peut se faire pendant le mouvement ou non de l'esclave.

L'instruction STOP est bloquante pour la tâche tant que le mouvement n'est pas terminé (condition MOVE_S(Axe)=0) alors que SSTOP n'est pas bloquante.

Syntaxe : STOP (<Axe1> {,<Axe2>...})

Exemple :

```
GEARBOX (Esclave, Maître, 2, 1, 1) 'activation du lien maître / esclave
STTI(Maître=+)                     ' départ du maître => l'esclave suit
...
STOP(Maître)                       ' arrêt du maître => l'esclave s'arrête
...
STOP(Esclave)                      'arrêt de la liaison maître / esclave
```

L'instruction AXIS(Esclave)=Off arrête également la liaison et le mouvement mais sans aucun contrôle car l'asservissement est inhibé.

5-8- Interpolation

5-8-1- Interpolation linéaire

La fonction MOVL permet d'effectuer une interpolation linéaire sur deux axes.

Syntaxe : MOVL(<AxeX>=<DestinationX>, <AxeY>=<DestinationY>[,<Vitesse>])

Les paramètres <DestinationX> et <DestinationY> s'expriment en relatif.

Cette instruction est non bloquante pour la tâche (excepté si le buffer de mouvements est plein).

Si l'instruction MERGE est activée et que l'on charge plusieurs MOVL dans le buffer de la carte d'axe, les mouvements seront enchaînés les uns aux autres sans passer par une vitesse nulle.

5-8-2- Interpolation circulaire

La fonction MOVC permet d'effectuer une interpolation circulaire sur deux axes. Les dimensions du cercle ou de l'arc sont définies par les quatre paramètres suivants:

MOVC(<AxeX>=<DestinationX>, <AxeY>=<DestinationY>, <CentreX>, <CentreY>, <SensHoraire> [,<Vitesse>])

Les paramètres <DestinationX> et <DestinationY> sont exprimé en relatif. Les paramètres <CentreX> et <CentreY> définissent la position du centre du cercle par rapport à la position courante. Le <SensHoraire> défini le sens de déplacement.

Cette instruction est non bloquante pour la tâche (excepté si le buffer de mouvements est plein).

Si l'instruction MERGE est activée et que l'on charge plusieurs MOVC dans le buffer de la carte d'axe, les mouvements seront enchaînés les uns aux autres sans passer par une vitesse nulle.

5-8-3- Accélération / Vitesse résultante

Comme en positionnement, un mouvement interpolé possède une phase d'accélération, vitesse plateau, décélération. La phase de décélération sera égale à celle d'accélération. Ces valeurs correspondent à des accélération et vitesse résultantes des deux axes.

L'instruction ACC(<AxeX>, <AxeY>)=Valeur permet d'affecter une valeur d'accélération et de décélération résultante. Elle est prise en compte si les axes sont à l'arrêt.

L'instruction VEL(<AxeX>, <AxeY>)=Valeur permet d'affecter une valeur de vitesse résultante. Elle est prise en compte à tout moment.

Si l'instruction MERGE(<AxeX>, <AxeY>) est activée et que l'on charge plusieurs MOVL, MOVC dans le buffer de la carte d'axe, la phase d'accélération sera effective jusqu'à ce que la vitesse soit atteinte et ensuite les mouvements étant enchaînés, on passera d'un mouvement à l'autre à la même vitesse. La phase de décélération s'effectuera sur le ou les derniers mouvements.

☏ Attention : sur des pièces qui possèdent des arcs de cercle et sur lesquelles on travaille en mouvements enchaînés (MERGE(X,Y)=On), l'accélération réelle peut devenir très importante :

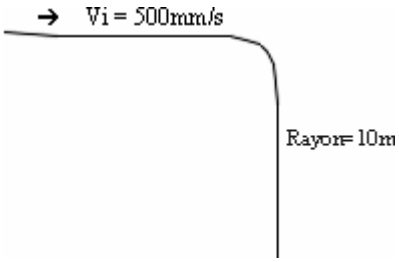
Accél (t) = $-(Vi^2 / r) * \cos(\text{Téta } i + ((Vi / r) * t))$

Accél. maxi = $-(Vi^2 / r)$

Avec Vi = vitesse initiale avant l'arc de cercle en m / s

r = rayon de l'arc en m

Accél en m / s²



Accél maxi = $-(0.5^2 / 0.01) = -25$ m/s²

5-8-4- Arrêt d'un mouvement

Pour arrêter un mouvement interpolé en cours de trajectoire, il faut utiliser l' instruction STOPI(<AxeX>, <AxeY>). Les axes s'arrêtent en suivant la trajectoire interpolée via la pente de décélération donnée par l'instruction ACC(<AxeX>,<AxeY>). Le buffer de mouvement est vidé : on ne pourra donc pas faire de reprise de trajectoire.

L'instruction est bloquante pour la tâche tant que le mouvement n'est pas terminé (condition MOVE_S(AxeX)=0 et MOVE_S(AxeY)=0). Par exemple :

```
ACC(X,Y)=1000
VEL(X,Y)=100
MERGE(X,Y)=On
MOVL(X=10,Y=5)
MOVL(X=20,Y=0)
...
STOPI(X,Y)
```

Si l'on souhaite faire une pause sur une trajectoire il faut mettre à 0 la vitesse résultante. Par exemple :

```
ACC(X  ,Y)=1000
VEL(X,Y)=100
MERGE(X,Y)=On
```

```
MOVL(X=10,Y=5)
MOVL(X=20,Y=0)
.....
...
IF FlagStop Then
    FlagStop=0
    VEL(X,Y)=0
END IF
IF FlagStart Then
    FlagStart=0
    VEL(X,Y)=100
END IF
...
```

5-8-5- Outils

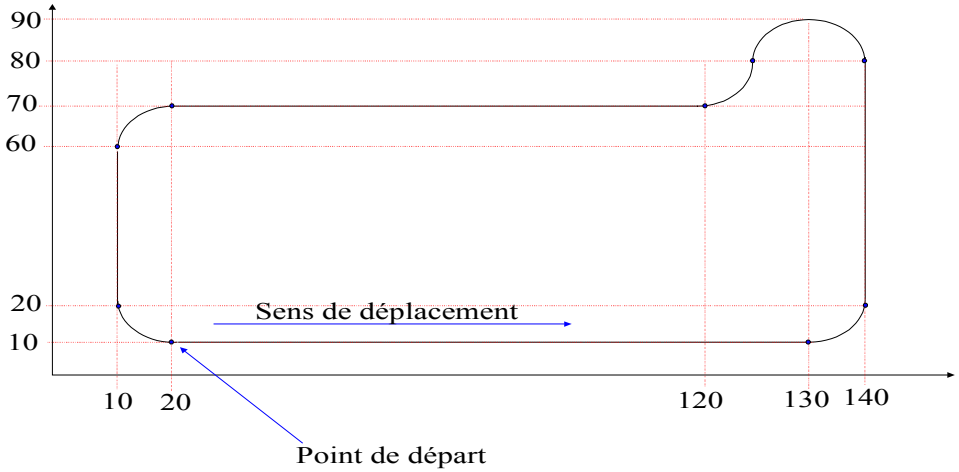
A partir de l'éditeur de tâche, il est possible de rapatrier un fichier de type .DXF qui sera converti en instructions MOVL et MOVC. Le fichier DXF devra comporter une seule forme bouclée.

Un interpréteur ISO est également disponible (option). Il reconnaît les codes G00, G01, G02, G03, G04, G17, G18, G19, G40, G41, G42, G61, G62, G90, G91, G92, M01, P, Q , K, N, E, S.

Consultez votre revendeur pour obtenir les informations complètes.

5-8-6- Exemples

A) Dépose de colle



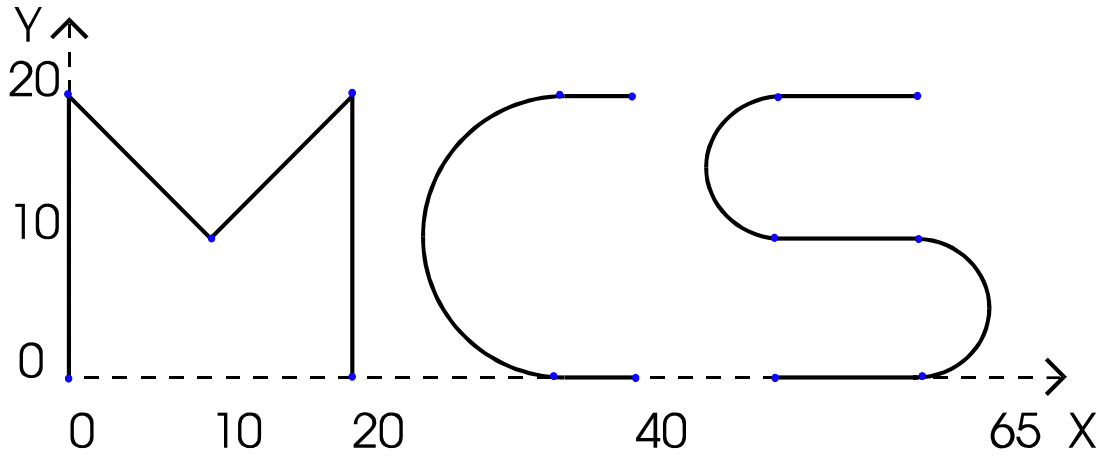
Programme :

```
AXIS(X)=On
AXIS(Y)=On
VEL(X)=1000                 'Vitesse en mode positionnement
VEL(Y)=1000
ACC(X)=1500                 'Accélération et décélération
DEC(X)=1500                 'en mode positionnement
ACC(Y)=1500
DEC(Y)=1500
VEL(X,Y)=300                'Vitesse interpolée
ACC(X,Y)=2000               'Accélération interpolée
MERGE(X,Y)=On               'Mouvements enchaînés
...
MOVA(X = 20, Y = 10)        'Déplacement de positionnement au 1er point
OUT(Dépose)=ON              'Activation
DELAY MarcheDépose          'Tempo
MOVL(X=110, Y=0)            'Déplacement interpolé en 130,10
MOVC(X=10, Y=10, 0, 10, 0)  'Déplacement en 140,20 sens anti-horaire
MOVL(X=0, Y=60)             'Déplacement en 140,80
MOVC(X=-20, Y=0, -10, 0, 0) 'Déplacement en 120,80 sens anti-horaire
```

```
MOVC(X=-10, Y=-10, -10, 0, 1)      'Déplacement en 110,70 sens horaire
MOVL(X=-90, Y=0)                   'Déplacement en 20,70
MOVC(X=-10, Y=-10, 0, -10, 0)      'Déplacement en 10,60 sens anti-horaire
MOVL(X=0, Y=-40)                   'Déplacement en 10,20
MOVC(X=10, Y=-10, 10, 0, 0)        'Déplacement en 20,10 sens anti-horaire
WAIT (MOVE_S(X)=Off) And (MOVE_S(Y)=Off) 'Attente de la fin des
                                          'déplacements
OUT(Dépose)=OFF              'Désactivation
DELAY  ArretDépose           'Tempo
MOVA(X = 0, Y = 0)           'Retour à l'origine
```

Dans cet exemple, les axes X and Y sont utilisés pour déposer un cordon de colle. "Dépose" est la sortie qui commande la dépose de colle. Après un mouvement de l'origine vers la première position (20,10), la dépose commence en décrivant la figure. Quand le mouvement rejoint le point de départ, la sortie est désactivée et les axes se replacent à l'origine.

B) Gravure ou découpe



Les points bleus sont les points de passage

Programme :

```
AXIS(X)=On
AXIS(Y)=On
VEL(X)=1000          'Vitesse en mode positionnement
VEL(Y)=1000
ACC(X)=1500          'Accélération et décélération en mode positionnement
DEC(X)=1500
ACC(Y)=1500
DEC(Y)=1500
VEL(X,Y)=300         'Vitesse interpolée
ACC(X,Y)=2000        'Accélération interpolée
MERGE(X,Y)=Off       'Mouvements non enchaînés
...
MOVA(X = 0, Y = 10)  'Déplacement de positionnement au 1er point
OUT(Graveuse)=ON     'Activation
DELAY MarcheGraveuse 'Tempo
MOVL(X=0, Y=20)      'Déplacement interpolé en 0,20
MOVL(X=10, Y=-10)    'Déplacement en 10,10 - Gravure du M
MOVL(X=10, Y=10)     'Déplacement en 20,20
MOVL(X=0, Y=-20)     'Déplacement en 20,0
WAIT (MOVE_S(X)=Off) And (MOVE_S(Y)=Off) 'Attente de l'arrêt des axes
OUT(Graveuse)=OFF    'Arrêt de la graveuse
DELAY ArretGraveuse  'Tempo
MOVA(X=40, Y=0)      'Déplacement en 40,0
OUT(Graveuse)=ON     'Activation
DELAY MarcheGraveuse 'Tempo
MOVL(X=-5, Y=0)      'Déplacement en 35,0
MOVC(X=0, Y=20, 0, 10, 1)   'Déplacement en 35,20 - Gravure du C
MOVL(X=5, Y=0)       'Déplacement en 40,20
WAIT (MOVE_S(X)=Off) And (MOVE_S(Y)=Off) 'Attente de l'arrêt des axes
OUT(Graveuse)=OFF    'Arrêt de la graveuse
```

```
DELAY ArretGraveuse   'Tempo
MOVA(X=60, Y=20)      'Déplacement en 60,20
OUT(Graveuse)=ON      'Activation
DELAY MarcheGraveuse  'Tempo
MOVL(X=-10, Y=0)      'Déplacement en 50,20
MOVC(X=0, Y=-10, 0, -5, 0)  'Déplacement en 50,10
MOVL(X=10, Y=0)       'Déplacement en 60,10 - Gravure du S
MOVC(X=0, Y-10, 0, -5, 1)          'Déplacement en 60,0
MOVL(X=-10, Y=0)      'Déplacement en 50,0
WAIT (MOVE_S(X)=Off) And (MOVE_S(Y)=Off) 'Attente de l'arrêt des axes
OUT(Graveuse)=OFF     'Arrêt de la graveuse
DELAY ArretGraveuse   'Tempo
MOVA(X = 0, Y = 0)    'Retour à l'origine
```

5-9- Capture

5-9-1- Capture

A) CAPTURE (sur carte servo SRV 15 ou SRV 85)

L'instruction CAPTURE est utilisée pour enregistrer la position courante d'un axe. Une entrée spécifique à la capture (également utilisée pour la prise d'origine) est présente sur chaque carte servo et codeur.

Syntaxe : CAPTURE(<Axe>,<Inversion>,<Fenêtre>,<Mini>,<Maxi>,<Intérieur>)

Avec cette instruction la MCS attend un front sur l'entrée capture. Quand le front est détecté, la position est stockée dans la variable REGPOS_S. Le flag REG_S est alors positionné à vrai.

<Inversion> permet de traiter le front descendant au lieu du front montant. Si <Fenêtre> est vraie alors l'entrée n'est testée que lorsque <Axe> est entre les positions <Mini> et <Maxi>.

<Interieur> permet de définir si le test s'effectue à l'intérieur ou à l'extérieur des bornes <Mini> et <Maxi>

<Mini> doit toujours être inférieur à <Maxi>.

```
Par exemple :
CAPTURE(X,Off,On,10,20,On)   'Capture position sur front montant
                             'lorsque X est entre 10 et 20
WAIT REG_S(X)=True
MOVA(Y=500+REGPOS_S(X))
```

La position capturée sur l'axe X est utilisée afin de déterminer la position cible du prochain mouvement sur l'axeY.

B) CAPTURE1 (sur carte servo SRV 85 seulement)

L'instruction CAPTURE1 est une extension de l'instruction CAPTURE.

Elle permet d'enregistrer la position de n'importe quelle carte servo SRV 85 à partir d'un front de l'une de ces 2 entrées de capture.

Syntaxe :

CAPTURE1(<Axe1>,<Axe2>,<Configuration1>,<Fenêtre2>,<Mini2>,<Maxi2>,
<Intérieur2>)

<Axe1> : Axe où est situé l'entrée de capture (SRV 85 seulement)

<Axe2> : Axe sur lequel on souhaite capturer la position (SRV 85 seulement)

<Configuration1> : Octet de configuration hardware de <Axe1> :

b0 : non utilisé

b1 : capture sur top Z

b2 : capture sur entrée n°1 C1

b3 : capture sur entrée n°2 C2

b4 : choix du front (0 : front montant, 1 : front descendant)

b5..7 : non utilisés

<Fenêtre2> : activation d'une fenêtre bornée par les positions <Mini2> et <Maxi2> de <Axe2> dans laquelle l'entrée capture est traitée.

<Interieur2> permet de définir si le test s'effectue à l'intérieur ou à l'extérieur des bornes.

<Mini2> doit toujours être inférieur à <Maxi2>.

Avec cette instruction la MCS attend un front sur l'information capture. Quand le front est détecté, la position est stockée dans la variable REGPOS1_S. Le flag REG1_S est alors positionné à vrai.

```
Par exemple :
CAPTURE1(Y,X,4,On,10,20,On)  ' Capture position axe X sur front montant
                             ' de l'entrée C1 de l'axe Y lorsque
                             ' la position de l'axe X est entre 10 et 20
WAIT REG1_S(X)=True
Correction!=Phase-REGPOS1_S(X)
...
```

C) CAPTURE2 (sur carte servo SRV 85 seulement)

L'instruction CAPTURE2 est équivalente à l'instruction CAPTURE mais elle intègre en plus le choix de capture sur l'entrée C1 ou C2 ou le top Z du codeur.

Syntaxe :

CAPTURE2(<Axe>,<Configuration>,<Fenêtre>,<Mini>,<Maxi>,<Intérieur>)

<Axe> : Nom de l'axe (SRV 85 seulement)

<Configuration> : Octet de configuration hardware de <Axe>

b0 : non utilisé

b1 : capture sur top Z

b2 : capture sur entrée n°1 C1

b3 : capture sur entrée n°2 C2

b4 : choix du front (0 : front montant, 1 : front descendant)

b5..7 : non utilisés

<Fenêtre> : activation d'une fenêtre bornée par les positions <Mini> et <Maxi> de <Axe> dans laquelle l'entrée capture est traitée.

<Interieur> : permet de définir si le test s'effectue à l'intérieur ou à l'extérieur des bornes.

<Mini> doit toujours être inférieur à <Maxi>.

Avec cette instruction la MCS attend un front sur l'information capture. Quand le front est détecté, la position est stockée dans la variable REGPOS2_S. Le flag REG2_S est alors positionné à vrai.

```
Par exemple :
CAPTURE2(X,24,Off,0,0,Off)          ' Capture position axe X sur front
                                    ' descendant de l'entrée C2 de l'axe X
WAIT REG2_S(X)=True
Correction!=Phase-REGPOS2_S(X)
...
```

D) Informations complémentaires

↳ La carte servo SRV 15 contient un seul registre géré par l'instruction CAPTURE.

↳ La carte servo SRV 85 possède deux registres : registre1 géré par CAPTURE1, registre2 géré par CAPTURE2.

↳ Dans le cas d'une SRV 85, les instructions CAPTURE et CAPTURE2 utilisent le même registre interne. Il est donc préférable d'utiliser sur une même carte, soit CAPTURE, soit CAPTURE2, mais pas les deux à la fois.

↳ Il ne faut pas affecter à une même entrée d'une carte SRV 85 (entrée C1 ou C2), des fonctions de comptage (SETUPCOUNTER) et des fonctions de capture (CAPTURE, CAPTURE1 ou CAPTURE2).

5-10- Codeurs temporels

5-10-1- Codeurs temporels

Le système contient deux codeurs temporels, qui peuvent être utilisés comme axe maître dans une fonction de synchro ou une boîte à cames. Ces codeurs sont nommés TIMEBASE1 et TIMEBASE2

Les fonctions communes aux axes et codeurs temporels sont :

↳ Lecture de la position : POS_S

↳ Remise à zéro : CLEAR

↳ Arrêt : STOP

↳ Definition du modulo : MODVAL_P

Une fonction spécifique permet de démarrer un codeur temporel : STARTTIME

Syntaxe :

STARTTIME(TIMEBASE1) ou STARTTIME(TIMEBASE2)

La position d'un codeur temporel est incrémentée toutes 1.0005ms s'il est utilisé sur une boîte à cames (ex : CAMBOX).

Si un codeur temporel est le maître d'une fonction de synchro, sa position est incrémentée toutes les 0.3335ms. (ex : MOVS, CAM).

Ces codeurs sont très limités car on ne peut pas modifier la base de temps. Il faut sinon utiliser une carte servo déclarée en mode virtuel (instruction LOOP).

5-11- Défaut sur un axe

5-11-1- Défaut sur un axe

Dès qu'un axe passe en mode asservi, il est contrôlé à tout moment : à l'arrêt, en mouvement.

Si la différence entre sa position théorique calculée et sa position réelle donnée par le retour codeur est supérieure au paramètre « erreur de poursuite maxi », le système réagit de la façon suivante :

↳ Mise à 1 du flag FEMAX_S

↳ Passage de tous les axes servo en mode non asservi et ouverture du contact de chien de garde si l'instruction SECURITY n'a pas été utilisée.

↳ Passage de tous les axes servo en mode non asservi si l'instruction SECURITY(On,Off) a été exécutée.

↳ Ouverture du contact de chien de garde si l'instruction SECURITY(On,Off) a été exécutée.

↳ Forçage à 0 de la consigne analogique et vidage du buffer de tous les axes passés en mode non asservi.

↳ Forçage à 0 des flags MOVE_S de tous les axes passés en mode non asservis. Si des tâches continuent à envoyer des instructions de mouvement sur ces axes, les mouvements seront consommés mais non exécutés.

↳ Arrêt des liaisons maître / esclave sur les axes esclaves passés en mode non asservis.

Le flag FEMAX_S repassera à 0 dès que l'on remettra l'axe en mode asservi (AXIS(Axe)=On).

☏ Attention : l'instruction SECURITY est automatiquement forcée à SECURITY(On,On) si toutes les tâches de la MCS sont à l'arrêt.

6- PROGRAMMATION DE L'AUTOMATE

6-1- Tâche pseudo-basic

6-1-1- 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
                      'd'une carte SIH24
B2# = INPB(Bloc2)     'lecture du deuxième bloc de 8 entrées
                      'd'une carte SIH24
B3# = INPB(Bloc3)     'lecture du troisième bloc de 8 entrées
                      'd'une carte SIH24
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
```

6-1-2- Entrées/Sorties analogiques

A) Lecture d'une entrée

La fonction ADC est utilisée pour lire une entrée analogique. Sa syntaxe est : ADC(<EntréeAna>)

<EntréeAna> doit représenter l'identificateur d'une entrée analogique. Cet identificateur peut être soit un nom symbolique utilisé dans le module de configuration ou le nom par défaut de cette même entrée. Les données retournées par la fonction sont toujours de type réel et comprises entre -10 et +10.

Par exemple:

```
A! = ADC(Temperature)       'Lecture d'une entrée analogique
```

B) Ecriture d'une sortie

La fonction DAC est utilisée pour écrire sur une sortie analogique.

La syntaxe est : DAC(<SortieAnalogique>)=<Expression réelle>

<SortieAna> doit représenter l'identificateur d'une sortie analogique. Cet identificateur peut être soit un nom symbolique utilisé dans le module de configuration ou le nom par défaut de cette même sortie. Les données utilisées par l'instruction sont toujours de type réel et comprises entre -10 et +10.

Par exemple :

```
DAC(Consigne)=5.0     'Ecriture d'une valeur de consigne de 5 V
```

C) Lecture d'une sortie

Toutes les sorties peuvent également être lues. La valeur lue correspond à la dernière valeur écrite.

Cette fonctionnalité est très utile si plusieurs tâches utilisent la même sortie.

Par exemple :

```
DAC(Consigne)=DAC(Consigne)*2      'Multiplication par 2 de la consigne
                                   'de sortie
```

6-1-3- 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

6-1-4- 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
```

6-1-5- 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.
```

6-1-6- 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.

6-1-7- 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.

6-1-8- 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.

6-1-9- Compteurs

A) Compteurs

La carte SRV 85 possède 2 compteurs 16 bits. Chaque entrée de la carte peut être affectée à un compteur.

☏ Attention :

- Une même entrée ne peut utiliser à la fois la fonction de comptage et de capture de position.

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

6-1-10- Configuration

L'instruction SETUPCOUNTER permet de configurer le compteur.

Syntaxe : SETUPCOUNTER(<Axe>,<Entrée>,<Inversion>,<Filtre>)

<Axe> : Nom de la carte servo

<Entrée> : Numéro de l'entrée (1 pour l'entrée C1, 2 pour l'entrée C2)

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

<Filtre> : Validation du filtre : 0 pour sans filtrage, 1 pour filtre de 2 ms.

Si le filtre n'est pas activé, la fréquence maxi est de 1.5 KHz sinon de 200 Hz.

6-1-11- Remise à zéro

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

Syntaxe : CLEARCOUNTER(<Axe>,<Entrée>)

<Axe> : Nom de la carte servo

<Entrée> : Numéro de l'entrée (1 pour l'entrée C1, 2 pour l'entrée C2)

6-1-12- Lecture

L'instruction COUNTER_S permet de lire le compteur.

Syntaxe : <Variable>=COUNTER_S(<Axe>,<Entrée>)

<Variable> : entier compris entre 0 et 65535

<Axe> : Nom de la carte servo

<Entrée> : Numéro de l'entrée (1 pour l'entrée C1, 2 pour l'entrée C2)

6-1-13- Compléments

L'état de chacune des entrées C1 et C2 d'une carte SRV 85 peut être lu grâce aux instructions SENSOR1_S et SENSOR2_S.

```
Exemple  :
OUT(Lamp1)=SENSOR1_S(X)
```

6-1-14- Boîte à cames

A) Boîtes à cames

La boîte à cames permet de piloter des sorties TOR suivant des positions angulaires, linéaires ou temporelles par des instructions optimisées. MCB propose une boîte à cames élaborée à 3 niveaux de complexité.

MCB accepte jusqu'à 8 boîtes à cames avec jusqu'à 16 segments par boîte. Chaque boîte commande une carte de sorties (8 ou 16 sorties). Par exemple, sur une carte 16 sorties, les sorties 3, 4 et 12 peuvent être affectées à la boîte et les autres sorties peuvent être utilisées ailleurs.

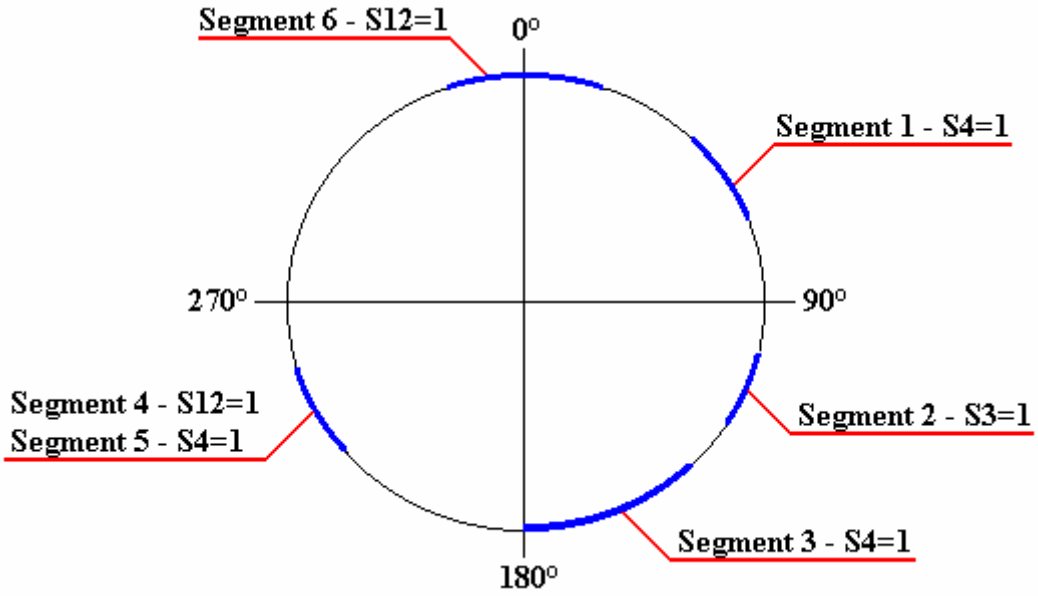
Si une seule boîte a cames est déclarée, les sorties de la boîte seront remise à jour toutes les 1ms (2 boîtes à cames ⇨ toutes les 2ms,…, 8 boîtes à cames ⇨ toutes les 8ms).

Les fonctions disponibles sont :

CAMBOX, CAMBOXSEG, CAMBOXDELAY, STARTCAMBOX et STOPCAMBOX

Lors de la déclaration d'un segment, la valeur de début peut-être supérieure à celle de fin. Le zéro programme est pris en compte à chaque définition de segment.

6-1-15- Boîte à cames simple



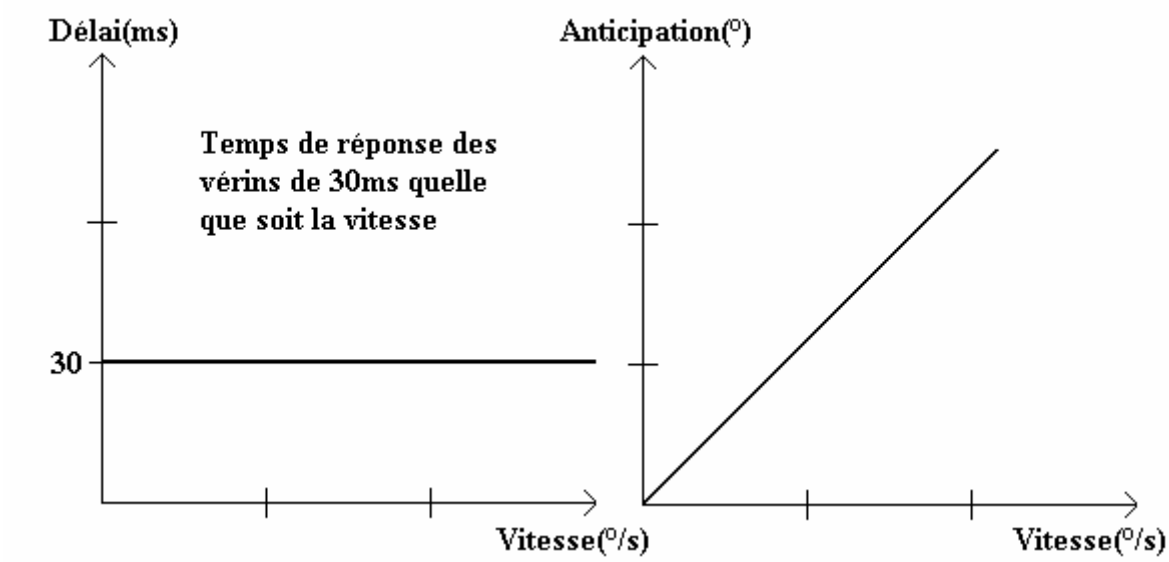
Dans cet exemple, l'axe <Maître> est un axe modulo 360 et la carte de sortie à pour nom A. La boite à came s'écrit donc de la façon suivant :

```
CAMBOX (1,Maître,A, 6, 0)     'La boîte à cames n°1 a 6 segments
CAMBOXSEG(1,1,4,20,60)        'Le segment 1 met la sortie 4 à 1
                              'entre 20° et 60°
CAMBOXSEG(1,2,3,100,135)      'Le segment 2 met la sortie 3 à 1
                              'entre 100° et 135°
CAMBOXSEG(1,3,4,135,180)      'Le segment 3 met la sortie 4 à 1
                              'entre 135° et 180°
CAMBOXSEG(1,4,12,200,240)     'Le segment 4 met la sortie 12 à 1
                              'entre 200° et 240°
CAMBOXSEG(1,5,4,200,240)      'Le segment 5 met la sortie 4 à 1
                              'entre 200° et 240°
CAMBOXSEG(1,6,12,350,10)      'Le segment 6 met la sortie 12 à 1
                              'entre 350° et 10°
STARTCAMBOX (1)               'Lancement de la boîte
...
STOPCAMBOX (1)                'Arrêt de la boîte
```

6-1-16- Compensation des temps de réponse

La boîte à cames peut également compenser les temps de réponse des actionneurs. Cela se traduit par une anticipation de la commande des actionneurs proportionnelle à la vitesse.

Dans l'exemple précédent, on désire compenser le temps de réponse de 30ms des vérins pneumatiques pilotés par les sorties 3, 4 et 12. Cette anticipation s'exprime de la façon suivante :




```
CAMBOX (1,Maître,A, 6, 1)     'La boîte à cames n°1 a 6 segments
                              'et 1 délai d'anticipation
CAMBOXSEG(1,1,4,20,60)        'Le segment 1 met la sortie 4 à 1
                              'entre 20° et 60°
CAMBOXSEG(1,2,3,100,135)      'Le segment 2 met la sortie 3 à 1
                              'entre 100° et 135°
CAMBOXSEG(1,3,4,135,180)      'Le segment 3 met la sortie 4 à 1
                              'entre 135° et 180°
CAMBOXSEG(1,4,12,200,240)     'Le segment 4 met la sortie 12 à 1
                              'entre 200° et 240°
CAMBOXSEG(1,5,4,200,240)      'Le segment 5 met la sortie 4 à 1
                              'entre 200° et 240°
CAMBOXSEG(1,6,12,350,10)      'Le segment 6 met la sortie 12 à 1
                              'entre 350° et 10°
CAMBOXDELAY(1,1,0,30)         'Une anticipation de 30ms est ajoutée
                              'quelle que soit la vitesse
STARTCAMBOX (1)               'Lancement de la boîte n°1
```

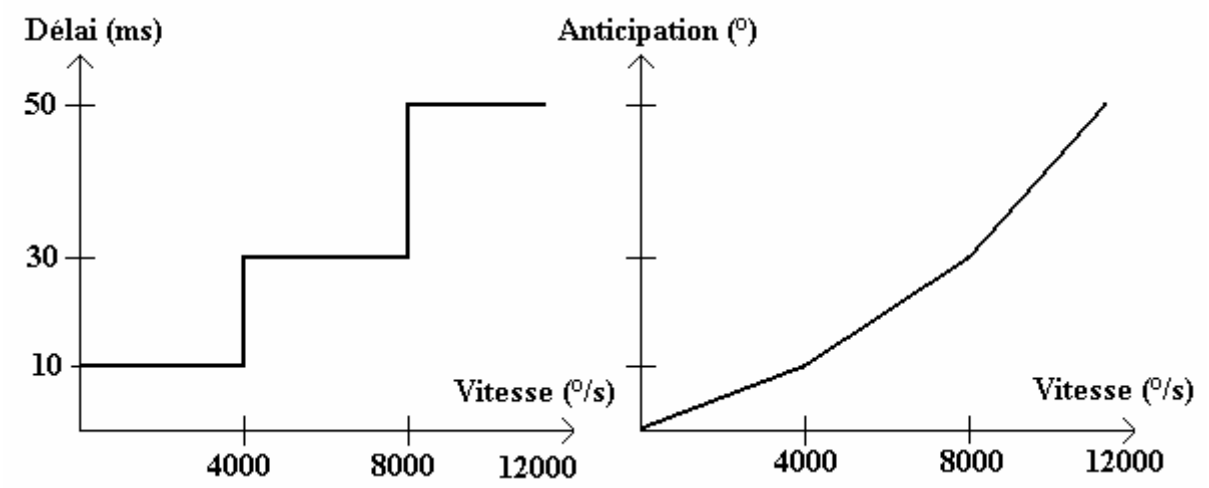
```
...
STOPCAMBOX (1)              'Arrêt de la boîte n°1
```

6-1-17- Compensation des temps de réponse et du comportement des produits

Pour des applications plus pointues, il est possible d'utiliser plusieurs paliers d'anticipation pour compenser les actionneurs et le comportement du produit.

Dans l'exemple précédent, on désire avoir une compensation de 10 ms jusqu'à 4000°/s, 30 ms de 4000°/s à 8000°/s et 50ms de 8000°/s et au-delà.

Cette anticipation s'exprime de la façon suivante :




```
CAMBOX (1,Maître,A, 6, 3)    'La boîte à cames n°1 a 6 segments
                             'et 3 délais d'anticipation
CAMBOXSEG(1,1,4,20,60)       'Le segment 1 met la sortie 4 à 1
                             'entre 20° et 60°
CAMBOXSEG(1,2,3,100,135)     'Le segment 2 met la sortie 3 à 1
                             'entre 100° et 135°
CAMBOXSEG(1,3,4,135,180)     'Le segment 3 met la sortie 4 à 1
                             'entre 135° et 180°
CAMBOXSEG(1,4,12,200,240)    'Le segment 4 met la sortie 12 à 1
                             'entre 200°et 240°
CAMBOXSEG(1,5,4,200,240)     'Le segment 5 met la sortie 4 à 1
                             'entre 200° et 240°
CAMBOXSEG(1,6,12,350,10)     'Le segment 6 met la sortie 12 à 1
                             'entre 350° et 10°
CAMBOXDELAY(1,1,4000,10)     'Une anticipation de 10ms est ajoutée
                             'jusqu'à 4000° / s
CAMBOXDELAY(1,2,8000,30)     'Une anticipation de 30ms est ajoutée
                             'jusqu'à 8000° / s
CAMBOXDELAY(1,3,12000,50)    'Une anticipation de 50ms est ajoutée
                             'jusqu'à 12000° / s
STARTCAMBOX (1)              'Lancement de la boîte n°1
...
STOPCAMBOX (1)               'Arrêt de la boîte n°1
```

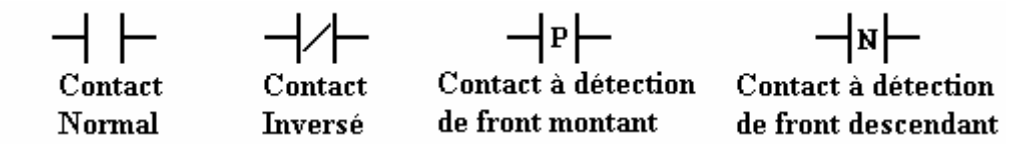
6-2- Tâche ladder

6-2-1- 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.

6-2-2- Contacts, Bobine, Blocs

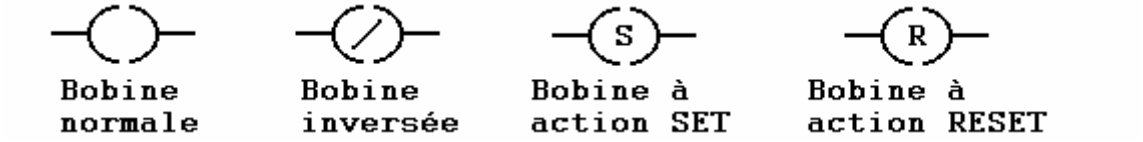
6-2-3- Contacts



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.

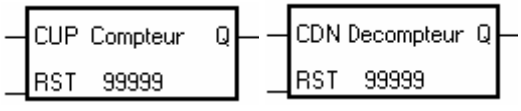
6-2-4- Bobines



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".

6-2-5- 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&
```

6-2-6- 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.

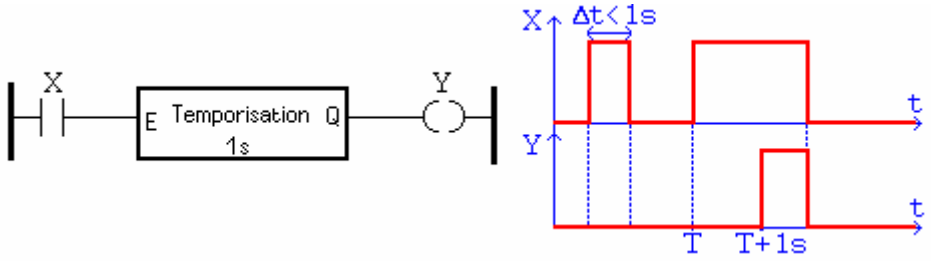
La variable associée à la temporisation est accessible dans la tâche sous la forme : <Nom du bloc> + <TVAL!>. Cette variable indique la durée écoulée depuis la mise en fonction du bloc.

```
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.

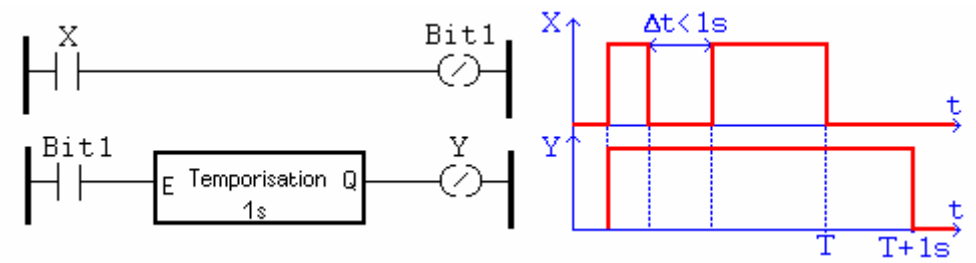
Exemple :



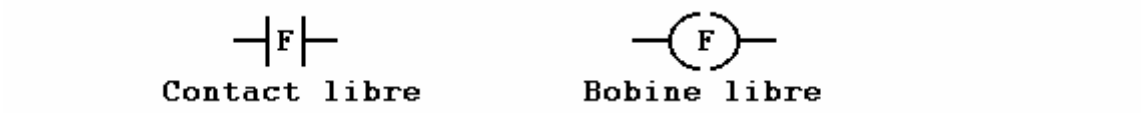
↳ Réalisation d'une temporisation de type retard au déclenchement :

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 :



6-2-7- 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.

6-2-8- Bit systèmes

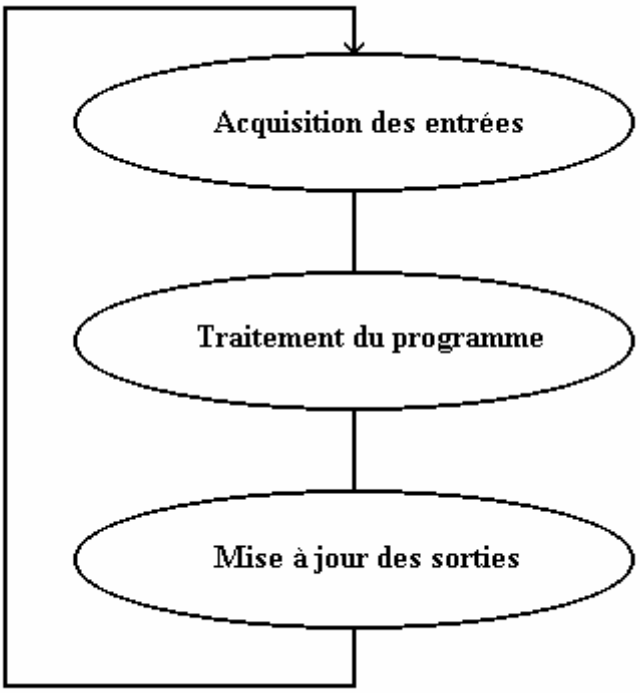
↳ Bit d'initialisation (initialisation) : Ce bit est 1 lors de la première scrutation de la tâche ladder.

↳ 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.

6-2-9- 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.

7- PROGRAMMATION DES PORTS DE COMMUNICATION SERIAL1 / SERIAL2

7-1- Introduction

La MCS 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 la MCS.

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.

7-2- 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 MCB (accessible avec un double click sur le SUBD de la Serial).

Exemple :

```
OPEN «SERIAL2  :9600,8,N,1" AS #1  ' ouverture du port SERIAL2
                                   ' pour gérer un terminal Dialog 640
```

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.

7-3- 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.

7-4- 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.

7-5- Fermeture d'un port

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

CLOSE #<CommNumber>

7-6- Spécificités du traitement RS 485

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

Pour émettre un message en RS 485, la MCS 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 la MCS tant que TX485 est non nul (fonction écho).

Exemple :

```
.....
Message$=  «  Motion Control System  »
TXD485(#1)=Len(Message$)
PRINT #1,Message$  ;         ' prise de la ligne RS485 pendant tout
                             ' l'envoi de Message$
CLEARIN #1                   ' vide les caractères écho
.....
```

7-7- 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 la mcs32
       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
```

8- PROGRAMMATION DU TERMINAL OPERATEUR

8-1- Présentation Dialog 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

Performances

- Processeur 16 bits
- 512Ko de mémoire flash
- 128Ko de mémoire ram sauvegardée
- Un port de communication série RS232
- Un port de communication série optionnel RS422 ou RS485
- Bus de terrain optionnel CANBUS

Caractéristiques techniques

- Alimentation 24Vdc
- Consommation 4W
- Température de service 0 à 45°C
- Température de stockage –20 à 70°C
- Indice de protection face avant IP65

8-2- Présentation Dialog 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

Performances

- Processeur 16 bits
- 512Ko de mémoire flash
- 128Ko de mémoire ram sauvegardée
- Un port de communication série RS232
- Un port de communication série optionnel RS422 ou RS485
- Bus de terrain optionnel CANBUS

Caractéristiques techniques

- Alimentation 24Vdc
- Consommation 6W
- Température de service 0 à 45°C
- Température de stockage –20 à 70°C
- Indice de protection face avant IP65

8-3- Fonctions pupitre

8-3-1- Ouverture liaison

Toutes les fonctions spécifiques aux terminaux opérateurs utilisent le port de communication déclaré sur le canal #1. Si toutefois le port de communication physique utilisé est "SERIAL2" il est nécessaire de réaffecter le port de communication #1 en utilisant la commande OPEN ou en utilisant la commande d'ouverture de liaison dans la barre d'outil de l'éditeur de tâche.

```
Open "Serial2:9600,8,N,1" As #1    ' Terminal connecté sur SERIAL2
ou
Open "Serial1:9600,8,N,1" As #1    ' Terminal connecté sur SERIAL1
```

8-3-2- Affichage

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

↳ La fonction CLS permet d'effacer l'écran partiellement (seulement Dialog80 et Dialog160) ou dans sa totalité. Pour ce faire, on doit préciser le numéro de la ligne à effacer. Si aucun numéro n'est précisé la totalité de l'écran est effacée.

La syntaxe de cette fonction est la suivante : CLS [<Numéro de ligne>].

Cette instruction possède d'autres extensions spécifiques au Dialog 640 :

⇨ 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
```

Pour le Dialog 640, il existe 5 autres types de fonctions :

↳ 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.

8-3-3- 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 ...
...
```

8-3-4- Editeur

Les terminaux DIALOG 80, 160 et 640 autorisent, 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, les pupitres Dialog 80 et 640 sont munies 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
```

8-3-5- Buzzer

Deux possibilités sont offertes pour utiliser le buzzer des pupitres DIALOG 80, 160 et 640 :

- 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
```

8-3-6- Backlight

Le pupitre Dialog 640 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.

8-3-7- Leds

Le pilotage des leds des pupitres Dialog 80 et 640 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 Dialog640 sont pilotées par l'instruction LED(@F1)…LED(@F6).

8-4- Correspondance des touches

8-4-1- Dialog 80

F1	à F6	@F1 à @F6	.	@POINT

0 à 9	@0 à @9	Shift	@SHIFT
Help	@HELP	▲	@UP
Alarm	@ALARM	▼	@DOWN
Esc	@ESC	▶	@RIGHT
Mod	@MOD	◀	@LEFT
+/-	@SIGN	Enter	@RETURN

8-4-2- Dialog 160

F1 à F6	@F1 à @F6	+/-	@SIGN
0 à 9	@0 à @9	.	@POINT
D	@D	Shift	@SHIFT
A	@A	▲	@UP
Esc	@ESC	▼	@DOWN
Mod	@MOD	▶	@RIGHT
Ins	@INS	◀	@LEFT
Del	@DEL	Enter	@RETURN

8-4-3- Dialog 640

F1 à F12	@F1 à @F12	.	@POINT
0 à 9	@0 à @9	▲	@UP
Help	@HELP	▼	@DOWN
Alarm	@ALARM	▶	@RIGHT
Esc	@ESC	◀	@LEFT
Mod	@MOD	Enter	@RETURN
+/-	@SIGN		

8-5- Menus internes

8-5-1- Généralités

Ces menus internes ne sont exploitables que sur les pupitres Dialog 160 et 640. Ils permettent de :

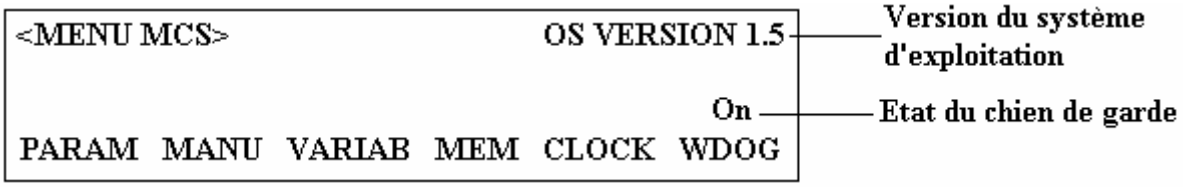
- Modifier les paramètres
- Tester les axes et entrées / sorties en manuel
- Lire et écrire les variables globales sauvegardées
- Gérer la sauvegarde et la restauration des données en flash
- Régler l'heure et la date
- Changer l'état du chien de garde

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>.

8-5-2- 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

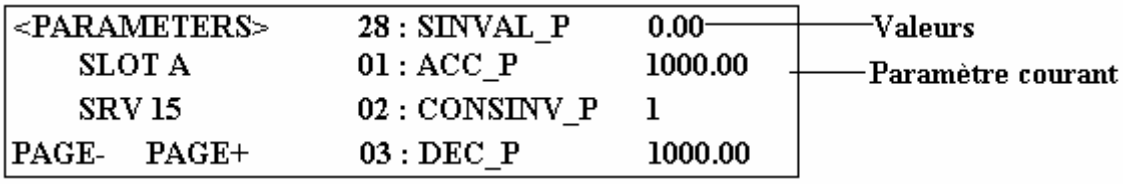
F5 : sous-menu horloge F6 : modifie l'état du chien de garde

ESC : sortie du menu

8-5-3- Sous-menu paramètre

Sous -menu paramètre : PARAMMCS

Syntaxe : CALL _PARAMMCS



Touches :

F1 : page précédente F2 : page suivante

← : slot précédent → : slot suivant

↑ : Paramètre précédent ↓ : paramètre suivant

ESC : sortie du menu ou retour au menu général MOD : modification d'un paramètre

Les paramètres ACC_P, DEC_P et VEL_P ne seront pris en compte que par les prochaines instructions ACC%, DEC% et VEL%.

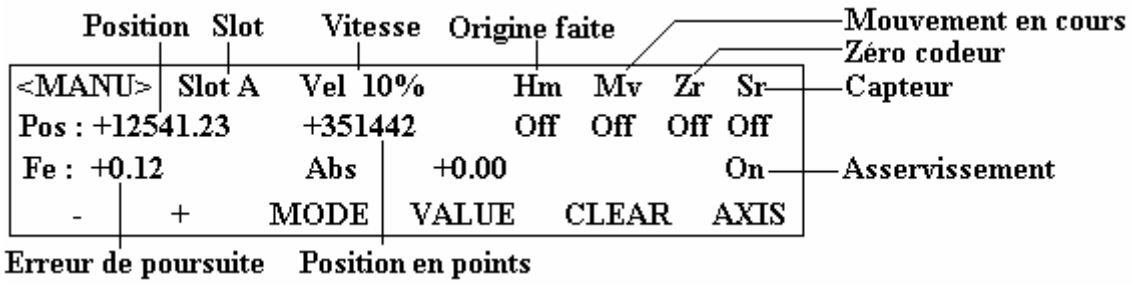
Affectation des paramètres pour la carte SRV85 :

01 : ACC_P	Accélération par défaut	17 : HOME_P	Type de prise d'origine
02 : BANDWIDTH	Bande passante du Filtre	19 : LIMMAX_P	Butée max
03 : GFILTER_P	Gain du Filtre réjécteur	20 : LIMMIN_P	Butée min
04 : CONSINV_P	Inversion de la consigne	21 : LIM_P	Activation des butées
05 : CONSMAX_P	Saturation du couple	22 : MODULO_P	Activation du modulo
06 : DEC_P	Décélération par défaut	23 : MODVAL_P	Valeur du modulo
07 : DISHOME_P	Dégagement	24 : OFFSET_P	Offset consigne analogique
08 : ENCINV_P	Inversion du codeur	25 : OUTVEL_P	Anticipation de vitesse
09 : ENCODER_P	Résolution du codeur	26 : POSMIN_P	Fenêtre de position
10 : FEMAX_P	Erreur de poursuite max	27 : SINVAL_P	Valeur du sinus
11 : FILTER_P	Activation du filtre réjécteur	28 : SIN_P	Accélération sinus
12 : FREQ_P	Fréquence centrale du filtre	29 : UNITREV_P	Unité par tour
13 : GDER_P	Gain dérivé	30 : VELFF_P	Anticipation d'accélération
14 : GINT_P	Gain intégral	31 : VELHOME_P	Vitesse de prise d'origine
15 : GPROP_P	Gain proportionnel	32 : VEL_P	Vitesse par défaut
16 : GTORQUE	Constante de couple	33 : ZERO_P	Zéro programme

8-5-4- Sous-menu manuel pour un axe

Sous-menu manuel : MANUMCS pour un axe

Syntaxe : CALL _MANUMCS



Touches :

F1 : mouvement en sens négatif F2 : mouvement en sens positif

F3 : mode de déplacement F4 : valeur ou type d'home

F5 : remise à zéro de la position F6 : asservissement

F7 : départ d'un mouvement F8 : arrêt d'un mouvement

← : slot précédent → : slot suivant

↑ : incrémente la vitesse ↓ : diminue la vitesse F12 : passe de 5 à 80%

ESC : sortie du menu ou retour au menu général

Modes : Abs : Mouvement absolu à la position absolue définie par VALUE (F4)

Rel : Mouvement relatif de la distance définie par VALUE (F4)

Inf+ : Mouvement infini en sens positif

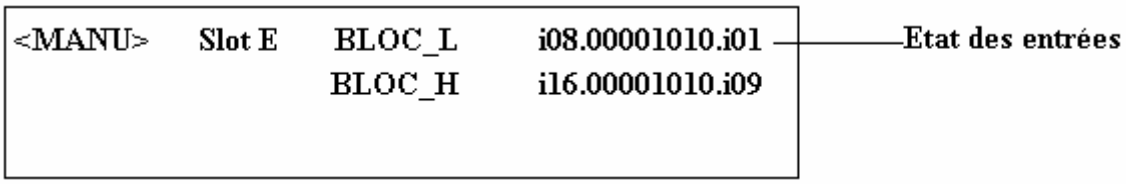
Inf- : Mouvement infini en sens négatif

Home : Prise d'origine suivant le numéro indiqué par VALUE

8-5-5- Sous-menu manuel pour les entrées TOR

Menu général : MANUMCS

Syntaxe : CALL _MANUMCS



Touches :

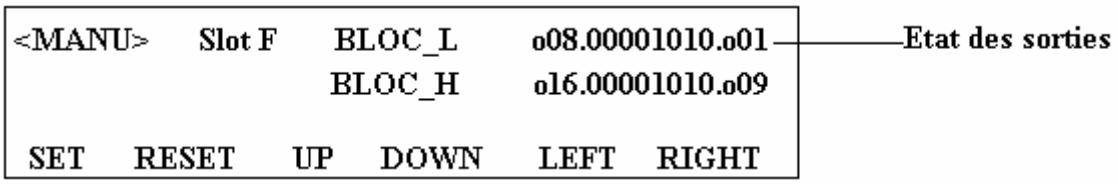
← : slot précédent → : slot suivant

ESC : sortie du menu ou retour au menu général

8-5-6- Sous-menu manuel pour les sorties TOR

Menu général : MANUMCS

Syntaxe : CALL _MANUMCS



Touches :

← : slot précédent → : slot suivant

F1 : mise à 1 F2 : mise à 0

F3 : bloc précédent F4 : bloc suivant

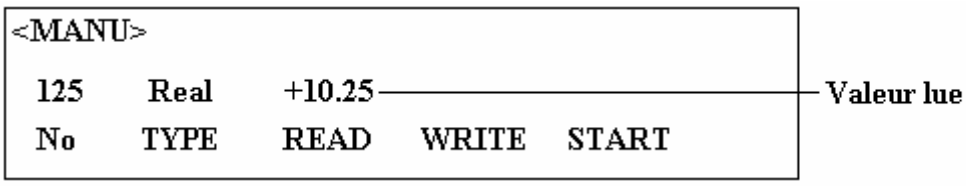
F5 : bit suivant F6 : bit précédent

ESC : sortie du menu ou retour au menu général

8-5-7- 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

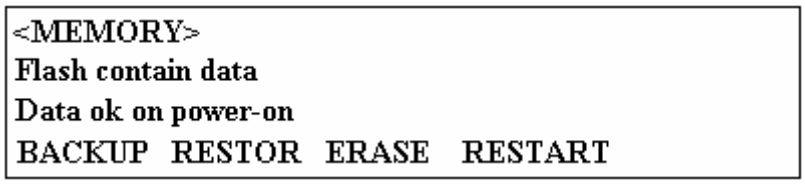
↑ : variable suivante ↓ : variable précédente

ESC : sortie du menu ou retour au menu général

8-5-8- Sous-menu mémoire

Menu général : MEMMCS

Syntaxe : CALL _MEMMCS



Touches :

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

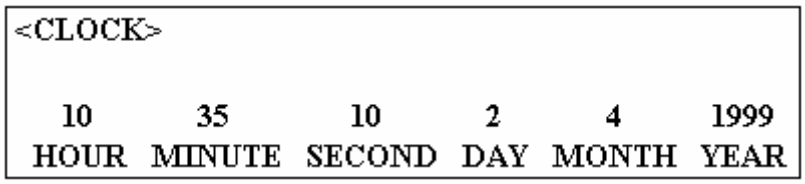
F3 : efface les données en flash F4 : redémarre la MCS32

ESC : sortie du menu ou retour au menu général

8-5-9- 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

9- LISTE DES PARAMETRES

9-1- Axe

ZERO_P	Zéro programme

9-2- Codeur

ENCODER_P	Nombre de points codeur ×4
UNITREV_P	Nombre d'unités par tour codeur
ENCINV_P	Inversion du sens de comptage
MODULO_P	Activation du modulo
MODVAL_P	Définition du modulo

9-3- Régulation

BANDWIDTH_P	Bande passante du filtre réjecteur (SRV85)
CONSINV_P	Inversion du sens de la consigne
CONSMAX_P	Limitation du couple (SRV85)
FEMAX_P	Erreur de poursuite maxi
FILTER_P	Active le filtre réjecteur (SRV85)
FREQ_P	Fréquence centrale du filtre réjecteur (SRV85)
GDER_P	Gain dérivé
GFILTER_P	Gain du filtre réjecteur (SRV85)
GINT_P	Gain intégral
GPROP_P	Gain proportionnel
GTORQUE_P	Constante du couple (SRV85)
OFFSET_P	Offset consigne analogique
OUTVEL_P	Anticipation de vitesse
POSMIN_P	Fenêtre positionnement mini
VELFF_P	Anticipation d'accélération

9-4- Profil de vitesse

ACC_P	Accélération par défaut
DEC_P	Décélération par défaut
VEL_P	Vitesse par défaut
SINACC_P	Activation de l'accélération sinus
SINVAL_P	Définition de l'accélération sinus

9-5- Prise d'origine

DISHOME_P	Distance de prise d'origine
INPHOME_P	Entrée de prise d'origine
VELHOME_P	Vitesse de prise d'origine

9-6- Butées logicielles

LIMMIN_P	Butée minimale
LIMMAX_P	Butée maximale
LIM_P	Activation des butées

9-7- Correspondance entre les limites exprimées en incréments et les limites en unité utilisateur

Les limites en unité utilisateur s'expriment au travers de l'expression suivante :

⇨ Limiteunité utilisateur= (Limiteincrément × Paramètre UNITREV_P)/(Paramètre ENCODER_P)

L'unité utilisateur correspond à l'unité sélectée dans le menu Axe de la configuration de la carte d'axe (Ex : mm, degré, tour…).

Exemple :

↳ Axe avec un codeur de 1000 impulsions ⇨ ENCODER_P(Axe)=4×1000=4000

↳ Codeur en bout de vis à billes au pas de 5mm ⇨ UNITREV_P(Axe)=5

↳ La valeur du paramètre POSMIN_P est de 0 à 32767 incréments, c'est à dire 0 à 40,95875 mm.

9-8- Liste alphabétique

9-8-1- ACC_P – Accélération par défaut

Syntaxe : **ACC_P**(<Axe>) = <Expression>

Unité : Unité utilisateur par s² (Ex : mm/s², degré/s², …)

Limites : 5.10^-5 à 1,5.10^6 incrément/s²

Types acceptés: <Expression> : réel

Description : Ce paramètre spécifie l'accélération chargée dans la carte d'axe à chaque démarrage de la MCS.

Remarques : Ce paramètre est utilisé par l'instruction ACC%

Exemple :

```
ACC_P(X)=5000
ACC%(X)=50 ' Accélération  = 2500
MOVA ...
```

Voir aussi : DEC_P

9-8-2- BANDWIDTH_P – Bande passante du filtre réjecteur (SRV85)

Syntaxe : **BANDWIDTH_P**(<Axe>) = <Expression>

Limites : <Expression> : de –32767 à +32767 incréments

Types acceptés : <Expression> : réel

Description : Ce paramètre spécifie la bande passante du filtre réjecteur de la boucle de régulation de la carte servo SRV85.

Voir aussi : GFILTER_P, FREQ_P, FILTER_P, GTORQUE_P

9-8-3- CONSINV_P– Inversion du sens de la consigne

Syntaxe : **CONSINV_P**(<Axe>) = ON / OFF

Description : Ce paramètre permet d'inverser la consigne analogique.

Remarques : Ce paramètre est utilisé avec ENCINV_P, pour changer le sens de déplacement d'un axe. Il peut également être utilisé pour palier une inversion dans le câblage de la consigne.

Exemple :
```
CONSINV_P(X)=On    'Inversion de la consigne
```

Voir aussi : ENCINV_P

9-8-4- CONSMAX_P – Limitation du couple (SRV85)

Syntaxe : **CONSMAX_P**(<Axe>) = <Expression>

Unités : <Expression> : Volt

Types acceptés : <Expression> : réel

Limites : <Expression> : de -10 à +10 Volt

Description : Ce paramètre spécifie la valeur de saturation du couple pour une carte servo SRV85 .

Voir aussi : GTORQUE_P

9-8-5- DEC_P – Décélération par défaut

Syntaxe : **DEC_P**(<Axe>) = <Expression>

Unité : unité utilisateur par s² (Ex : mm/s², degré/s²,…)

Limites : de 5.10^-5 à 1,5.10^6 incréments/s²

Types acceptés : <Expression> : réel

Description : Ce paramètre spécifie la décélération chargée dans la carte d'axe à chaque démarrage de la MCS.

Remarques : Ce paramètre est utilisé par l'instruction DEC%.

Exemple :
```
DEC_P(X)=5000
DEC%(X)=50        'La décélération est fixée à 50%
                  'donc 2500 unités /s²
```

Voir aussi : ACC_P

9-8-6- DISHOME_P – Distance de prise d'origine

Syntaxe : **DISHOME_P**(<Axe>) = <Expression>

Unité : <Expression> : unité utilisateur (Ex : mm, degré ; …)

Limites : +/-2^124 incréments

Types acceptés : <Expression> : réel

Description : Ce paramètre spécifie la distance de dégagement lors d'une prise d'origine.

Exemple :
```
DISHOME_P(X)=500
HOME(X) ' Position finale  = 500
```

Voir aussi : INPHOME_P

9-8-7- ENCINV_P – Inversion du sens de comptage

Syntaxe : **ENCINV_P**(<Axe>) = ON /OFF

Description : Ce paramètre permet d'inverser le sens de comptage.

Remarques : Ce paramètre est utilisé avec CONSINV_P, pour changer le sens de déplacement d'un axe. Il peut également être utilisé pour palier une inversion dans le câblage du codeur.

Exemple : `ENCINV_P(X)=On  'Inversion du sens de comptage`

Voir aussi : CONSINV_P

9-8-8- ENCODER_P – Nombre de points codeur

Syntaxe : **ENCODER_P**(<Axe>) = <Expression>

Unité : <Expression> : incréments

Limites : <Expression> : de 1 à 65535 incréments

Types acceptés : <Expression> : entier

Description : Ce paramètre spécifie le nombre de points codeur x 4 relié sur la carte d'axe.

Exemple : `ENCODER_P(<Axe>)=2000  'Codeur 500 points`

Voir aussi : UNITREV_P

9-8-9- FEMAX_P – Erreur de poursuite maxi

Syntaxe : **FEMAX_P**(<Axe>) = <Expression>

Unité : <Expression> : unité utilisateur (Ex : mm, degré,…)

Limites : <Expression> : de –32767 à +32767 incréments

Types acceptés : <Expression> : réel

Description : Ce paramètre spécifie l'erreur de poursuite maximale admissible.

Remarques : Ce paramètre est utilisé pour modifier la valeur de l'erreur de poursuite au-delà de laquelle le système passe en défaut : passage de tous les axes en boucle ouverte, ouverture du chien de garde et vidage du buffer de mouvements.

Exemple :
```
FEMAX_P(X)=5.0 'Erreur max 5.0 unités
MOVA...
```

Voir aussi : POSMIN_P, SECURITY

9-8-10- FILTER_P – Active le filtre réjecteur (SRV85)

Syntaxe : **FILTER_P**(<Axe>) = <Expression>

Types acceptés : <Expression> : bit

Description : Ce paramètre permet l'activation (1) ou non (0) du filtre réjecteur de la boucle de régulation de la carte servo SRV85.

Voir aussi : BANDWIDTH_P, GFILTER_P, FREQ_P, GTORQUE_P

9-8-11- FREQ_P – Fréquence centrale du filtre réjecteur (SRV85)

Syntaxe : **GFILTER_P**(<Axe>) = <Expression>

Limites : <Expression> : de –32767 à +32767 incréments

Types acceptés : <Expression> : réel

Description : Ce paramètre spécifie la fréquence centrale du filtre réjecteur de la boucle de régulation de la carte servo SRV85.

Voir aussi : BANDWIDTH_P, GFILTER_P, FILTER_P, GTORQUE_P

9-8-12- GDER_P – Gain dérivé

Syntaxe : **GDER_P**(<Axe>) = <Expression>

Limites : <Expression> : de 0 à 32000

Types : <Expression> : Réel

Description : Ce paramètre spécifie le gain dérivé du régulateur PID.

Remarques : Ce paramètre est utilisé pour modifier le gain dérivé à tout moment.

Exemple : `GDER_P(X)=100`

Voir aussi : GINT_P, GPROP_P, OUTVEL_P, VELFF_P

9-8-13- GFILTER_P – Gain du filtre réjecteur (SRV85)

Syntaxe : **GFILTER_P**(<Axe>) = <Expression>

Limites : <Expression> : de –32767 à +32767 incréments

Types acceptés : <Expression> : réel

Description : Ce paramètre spécifie le gain du filtre réjecteur de la boucle de régulation de la carte servo SRV85.

Voir aussi : BANDWIDTH_P, FREQ_P, FILTER_P, GTORQUE_P

9-8-14- GINT_P – Gain intégral

Syntaxe : **GINT_P**(<Axe>) = <Expression>

Limites : <Expression> : de 0 à 32 000 000 pour une carte SRV85
et de 0 à 32000 pour les autres cartes servo

Types acceptés : <Expression> : Réel

Description : Ce paramètre spécifie le gain intégral du régulateur PID.

Remarques : Ce paramètre est utilisé pour modifier le gain intégral à tout moment.

Exemple : `GINT_P(X)=50`

Voir aussi : GDER_P, GPROP_P, OUTVEL_P, VELFF_P

9-8-15- GPROP_P – Gain proportionnel

Syntaxe : **GPROP_P**(<Axe>) = <Expression>

Limites : <Expression> : de 0 à 32000

Types acceptés : <Expression> : Réel

Description : Ce paramètre spécifie le gain proportionnel du régulateur PID.

Remarques : Ce paramètre est utilisé pour modifier le gain proportionnel à tout moment.

Exemple : `GPROP_P(X)=1000`

Voir aussi : GINT_P, GDER_P, OUTVEL_P, VELFF_P

9-8-16- GTORQUE_P – Constante de couple (SRV85)

Syntaxe : **GTORQUE_P**(<Axe>) = <Expression>

Limites : <Expression> : de 0 à 32000

Types acceptés : <Expression> : Réel

Description : Ce paramètre spécifie la constante de couple de la boucle de régulation de la carte servo SRV85.

Voir aussi : CONSMAX_P

9-8-17- HOME_P – type de prise d'origine (SRV85)

Syntaxe 1 : **HOME_P**(<Axe>)=<Expression>

Syntaxe 2 : <Variable>=**HOME_P**(<Axe>)

Limites : <Expression>, <Variable> : de 1 à 10

Types acceptés : <Expression>, <Variable> : Octet

Description : Ce paramètre spécifie le type de prise d'origine de <Axe>. Les valeurs de 1 à 10 correspondent à une prise d'origine définit dans la fenêtre de configuration de la carte d'axe.

9-8-18- INPHOME_P – Entée de prise d'origine

Syntaxe : **INPHOME_P**(<Axe>) = <Entrée>

Description : Ce paramètre spécifie l'entrée utilisée comme capteur de prise d'origine.

Remarques : <Entrée> doit être le nom d'une entrée TOR ou le nom de la carte d'axe. Si <Axe> est utilisée comme nom d'entrée, l'entrée de la carte d'axe sera utilisée.

Exemple :
```
INPHOME_P(X)=X    ' On utilise l'entrée home de la carte d'axe
INPHOME_P(X)=CapteurHome       ' On utilise une entrée TOR
                                ' nommée CapteurHome
```

Voir aussi : DISHOME_P

9-8-19- LIMMAX_P – Butée maximale

Syntaxe : **LIMMAX_P**(<Axe>) = <Expression>

Unité : <Expression> : unité utilisateur (Ex : mm, degré,…)

Types acceptés : <Expression> : réel

Description : Ce paramètre spécifie la butée logicielle maximale.

Remarques : Si la position dépasse LIMMAX_P alors LIM_S et LIMMAX_S deviennent vrais.

Exemple :
```
LIMMAX_P(X)=100.0 ' Si position > 100 alors erreur
IF LIM_S(X)=True Then Goto Erreur
```

Voir aussi : LIMMIN_P, LIM_P

9-8-20- LIMMIN_P – Butée minimale

Syntaxe : **LIMMIN_P**(<Axe>) = <Expression>

Unité : <Expression> : unité utilisateur (Ex : mm, degré,…)

Limites : <Expression> : réel

Description : Ce paramètre spécifie la butée logicielle minimale.

Remarques : Si la position dépasse LIMMIN_P alors LIM_S et LIMMIN_S deviennent vrais.

Exemple :
```
LIMMIN_P(X)=100.0 ' Si position > 100 alors erreur
IF LIM_S(X)=True Then Goto Erreur
```

Voir aussi : LIMMAX_P, LIM_P

9-8-21- LIM_P – Activation des butées

Syntaxe : **LIM_P**(<Axe>)= ON / OFF

Description : Ce paramètre définit l'activation des butées.

Remarques : Ce paramètre est utilisé pour activer ou désactivater les butées logicielles. Quand ce paramètre est faux LIMMAX_P et LIMMIN_P n'ont pas d'effets.

Exemple : LIM_P(X)=ON

Voir aussi : LIMMAX_P, LIMMIN_P

9-8-22- MODULO_P – Activation du modulo

Syntaxe : **MODULO_P**(<Axe>) = ON / OFF

Description : Ce paramètre permet de déclarer un axe modulo. La position de l'axe évoluera entre 0 et la valeur du modulo

Remarques : Un axe infini doit être déclaré en axe modulo

Exemple :
```
MODVAL_P(X)=360   'Axe modulo 360 unités
MODULO_P(X)=On    'Activation du modulo
MOVA(X=100)       'Déplacement de l'axe dans le sens positif
                  'à la position 100°
MOVA(X=-20)       'Déplacement de l'axe dans le sens négatif
                  'à la position 20°
```

Voir aussi : MODVAL_P

9-8-23- MODVAL_P – Définition du modulo

Syntaxe : **MODVAL_P**(<Axe>) = <Expression>

Unité : <Expression> : unité utilisateur (Ex : mm, degré,…)

Limites : <Expression> : +/-2^124 incréments

Types acceptés : <Expression> : réel

Description : Ce paramètre permet de déclaré la valeur du modulo.

Remarques : Si le modulo est activé la position évoluera entre 0 et la valeur.

Exemple :
```
MODVAL_P(X)=360   'Axe modulo 360 unités
MODULO_P(X)=On
```

Voir aussi : MODULO_P

9-8-24- OFFSET_P – offset de la consigne analogique (SRV85)

Syntaxe : **CONSMAX_P**(<Axe>) = <Expression>

Unités : <Expression> : Volt

Types acceptés : <Expression> : réel

Limites : <Expression> : de -10 à +10 Volt

Description : Ce paramètre spécifie l'offset de la consigne analogique d'une carte servo SRV85.

9-8-25- OUTVEL_P – Anticipation de vitesse

Syntaxe : **OUTVEL_P**(<Axe>) = <Expression>

Limites : <Expression> : de 0 à 32000

Types acceptés : <Expression> : Réel

Description : Ce paramètre spécifie le coefficient d'anticipation de vitesse. Il permet de diminuer l'erreur de poursuite dynamique.

Remarques : Ce paramètre doit être utilisé si l'axe gère des fonctions de synchronisation ou d'interpolation. Les valeurs usuelles se situent entre 0 et 2000.

Exemple : OUTVEL_P(X)=500

Voir aussi : GDER_P, GPROP_P

9-8-26- POSMIN_P – Fenêtre de position mini

Syntaxe : **POSMIN_P**(<Axis>) = <Expression>

Unité : <Expression> : Unité utilisateur (Ex : mm, degré,…)

Limites : <Expression> : de 0 à 32767 incréments

Types acceptés : <Expression> : réel

Description : Ce paramètre spécifie la fenêtre de positionnement minimale.

Remarques : Ce paramètre est utilisé pour modifier la fenêtre de positionnement minimale entre la position réelle et la position théorique. Après un déplacement, si la différence entre la position réelle et la position demandée est inférieure à POSMIN_P, le système considère que la position est atteinte.

Exemple :
```
POSMIN_P(X)=0.1
MOVA(X=100) ' la position réelle est entre 99.9 et 100.1
```

9-8-27- SINACC_P – Activation de l'accélération sinus

Syntaxe : **SINACC_P**(<Axe>) = ON / OFF

Description : Ce paramètre permet d'utiliser l'accélération sinus.

Remarques : Ce paramètre est utilisé pour spécifier le comportement durant les phases d'accélération et de décélération. L'accélération sinus permet de réduire les efforts infligés à la mécanique pendant les phases d'accélération et de décélération.

Exemple :
```
SINACC_P(X)=On
```

Voir aussi : SINVAL_P

9-8-28- SINVAL_P – Définition de l'accélération sinus

Syntaxe : **SINVAL_P**(<Axe>) = <Expression>

Limites : <Expression> : de 0 à 10

Types acceptés : <Expression> : réel

Description : Ce paramètre spécifie la valeur de plateau dans la courbe sinus.

Plateau = Expression / (Expression+2)

Exemple :
```
SINVAL_P(X)=0     ' sinus pur
SINVAL_P(X)=2     ' plateau de 50% (2/4)
```

Voir aussi : SINACC_P

9-8-29- UNITREV_P – Nombre d'unités par tour codeur

Syntaxe : **UNITREV_P**(<Axe>) = <Expression>

Unité : <Expression> : unité utilisateur (Ex : mm, degré,…)

Limites : <Expression> : de 1 à 2^116 incréments

Types acceptés : <Expression> : réel

Description : Ce paramètre spécifie la distance parcourue en un tour codeur.

Exemple :
```
UNITREV_P(<Axe>)=5      'Vis à billes de 5mm avec
                        'codeur en bout de vis
```

Voir aussi : ENCODER_P

9-8-30- VELFF_P – Anticipation d'accélération

Syntaxe : **VELFF_P**(<Axe>) = <Expression>

Limites : <Expression> : de 0 à 32000

Types acceptés : <Expression> : Réel

Description : Ce paramètre spécifie le coefficient d'anticipation d'accélération. Il permet de diminuer l'erreur de poursuite dynamique pendant les phases d'accélération et de décélération.

Remarques : Ce paramètre doit être utilisé si l'axe gère des fonctions de synchronisation ou d'interpolation. Les valeurs usuelles sont comprises entre 0 et 2000.

Exemple :
```
VELFF_P(X)=20
MOVA...
```
Voir aussi : GDER_P, GPROP_P, GINT_P, OUTVEL_P

9-8-31- VELHOME_P – Vitesse de prise d'origine

Syntaxe : **VELHOME_P**(<Axe>) = <Expression>

Unité : <Expression> : unité utilisateur par seconde (Ex : mm/s, degré/s,…)

Limites : <Expression> : de 5.10^-5 à 1,5.10^6 incréments/s

Types acceptés : <Expression> : réel

Description : Ce paramètre spécifie la vitesse de prise d'origine en unité par seconde.

Remarques : Cette vitesse doit être faible.

Exemple :
```
VELHOME_P(X)=0.2*VEL_P(X)       ' Vitesse = 20% de la vitesse par
                                'défaut
```

9-8-32- VEL_P – Vitesse par défaut

Syntaxe : **VEL_P**(<Axe>) = <Expression>

Unité : <Expression> : unité utilisateur par seconde (Ex : mm/s, degré/s,…)

Limites : <Expression> : de 5.10^-5 à 1,5.10^6 incréments/s

Types acceptés : <Expression> : réel

Description : Ce paramètre spécifie la vitesse chargée dans la carte à chaque démarrage de la MCS.

Remarques : Ce paramètre est utilisé par la fonction VEL% .

Exemple :
```
VEL_P(X)=2000
VEL%(X)=10 'Vitesse  : 200 unités / s
MOVA ...
```
Voir aussi : ACC_P, DEC_P

9-8-33- ZERO_P – Zéro programme

Syntaxe : **ZERO_P**(<Axe>) = <Expression>

Unité : <Expression> : unité utilisateur (Ex : mm, degré,…)

Limites : <Expression> : +/-2^116

Types acceptés : <Expression> : réel

Description : Ce paramètre spécifie la position du zéro programme.

Remarques : Ce paramètre est utilisé pour spécifier la différence entre le zéro machine et le zéro logiciel. Toutes les positions gérées dans les tâches sont référencées par rapport à ce paramètre.

Exemple :
```
ZERO_P(X)=10.0 '10 mm entre zéro machine et zéro programme.
```

10- LISTE DES OPERATEURS ET INSTRUCTIONS

10-1- Programme

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

10-2- Arithmétique

+	Addition
-	Soustraction
*	Multiplication
/	Division

10-3- 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

10-4- Logique

<<	Décalage à gauche
>>	Décalage à droite
AND	Opérateur ET
NOT	Opérateur complément
OR	Opérateur OU
XOR	Opérateur OU exclusif

10-5- Test

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

10-6- Boucles

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

10-7- 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

10-8- 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

10-9- Contrôle de mouvement

10-9-1- Contrôle de l'axe

ACC	Accélération
ACC%	Accélération en pourcentage
ADDMOV	Superposition de mouvements (SRV85)
ADDSTOP	Arrêt de superposition de mouvements (SRV85)
AXIS	Contrôle la boucle d'asservissement
AXIS_S	Lit l'état de la boucle d'asservissement
BUFMOV_S	Nombre d'ordre en attente
CLEAR	Met à zéro la position de l'axe
CONS	Tension de consigne
CONS_S	Etat de la tension de consigne
CORRECTION	Fonction de compensation déclenchée sur entrée Capture
CORRECTION_S	Etat de la compensation (SRV85)
DEC	Décélération
DEC%	Décélération en pourcentage
FE_S	Erreur de poursuite
FEMAX_S	Limite d'erreur de poursuite
HOME	Prise d'origine
HOME_S	Etat de la prise d'origine
ICORRECTION	Fonction de compensation (SRV85)
LIMMIN_S	Etat de la butée minimale
LIMMAX_S	Etat de la butée maximale
LIM_S	Etat des butées
LOOP	Mode virtuel
MERGE	Définit l'enchaînement
MOVE_S	Etat du mouvement
ORDER	Numéro d'ordre du mouvement
ORDER_S	Numéro d'ordre courant
POS	Position à atteindre
POS_S	Position réelle
SENSOR_S	Etat du capteur
SENSOR1_S	Etat du capteur C1 (SRV85)
SENSOR2_S	Etat du capteur C2 (SRV85)
STOPCORRECTION	Arrêt de la fonction de compensation (SRV85)
VEL	Vitesse

VEL%	Vitesse en pourcentage
VEL_S	Vitesse réelle
ZERO_S	Etat du zéro codeur

10-9-2- Positionnement

MOVA	Mouvement absolu
MOVAC	Mouvement absolu déclenché sur entrée Capture
MOVAP	Mouvement absolu déclenché
MOVR	Mouvement relatif
SSTOP	Arrêt d'un axe sans attente
STOP	Arrêt d'un axe
STTA	Lance un mouvement absolu
STTI	Lance un mouvement infini
STTR	Lance un mouvement relatif
TRAJ	Trajectoire

10-9-3- Synchronisation

CAM	Came électronique
CAMC	Came électronique déclenchée sur entrée Capture
CAMNUM_S	Numéro de la came en cours d'exécution
CAMSEG_S	Numéro d'équation de la came en cours d'exécution
CAM_S	Etat de la came
GEARBOX	Arbre électrique
GEARBOXRATIO	modifie le rapport de réduction d'un arbre électrique
LOADCAMEX	Charge une came dans la carte servo
LOADPOINT	point d'une came dans la carte servo
LOADS	Charge un mouvement synchronisé
MOVC	Mouvement circulaire
MOVL	Mouvement linéaire
MOVS / MOVSP	Mouvement synchronisé
MOVSC	Mouvement synchronisé déclenché sur entrée Capture
STARTCAM	Lance l'exécution d'une came
STARTS	Lance un mouvement synchronisé
STOPI	Stoppe une interpolation

10-9-4- Capture

CAPTURE	Lancement de capture de position
CAPTURE1	Lancement de capture de position (SRV85)
CAPTURE2	Lancement de capture de position (SRV85)
REGPOS_S	Position capturée
REGPOS1_S	Position capturée (SRV85)
REGPOS2_S	Position capturée (SRV85)
REG_S	Etat de la capture

REG1_S	Etat de la capture (SRV85)
REG2_S	Etat de la capture (SRV85)

10-10- Automate

10-10-1- Entrées / sorties TOR

CAMBOX	Boîte à came
CAMBOXDELAY	Délais d'anticipation
CAMBOXSEG	Segment de boîte à came
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
SETINP	Filtrage et inversion des entrées
SETOUT	Inversion des sorties
STARTCAMBOX	Lance une boîte à came
STOPCAMBOX	Arrête une boîte à came
WAIT	Attente d'une condition

10-10-2- Entrées / sorties analogiques

ADC	Entrées analogiques
DAC	Sorties analogiques

10-10-3- 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

10-10-4- 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

10-10-5- Compteurs

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

10-11- 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

10-12- Terminaux opérateur

10-12-1- Dialog 80, 160 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
INKEY	Lit une touche
KEY	Dernière touche
KEYDELAY	Délai avant répétition touche
KEYREPEAT	Période de répétition touche
LOCATE	Positionne le curseur
PRINT	Affiche un texte
WAIT KEY	Attente d'une touche

10-12-2- Dialog 80 et 640

EDIT$	Saisie d’une valeur alphanumérique
LED	Pilotage des leds

10-12-3- Dialog 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

10-13- 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

10-14- Flash, Sécurité, Divers

CLEARFLASH	Efface la mémoire Flash
DISPLAY	Afficheur 7 segments
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
SECURITY	Définit les actions de sécurité
WATCHDOG	Chien de garde

10-15- Correspondance entre les limites exprimées en incréments et les limites en unité utilisateur

Les limites en unité utilisateur s'expriment au travers de l'expression suivante :

⇨ Limiteunité utilisateur= (Limiteincrément × Paramètre UNITREV_P)/(Paramètre ENCODER_P)

L'unité utilisateur correspond à l'unité sélectée dans le menu Axe de la configuration de la carte d'axe (Ex : mm, degré, tour…).

Exemple :

↳ Axe avec un codeur de 1000 impulsions ⇨ ENCODER_P(Axe)=4×1000=4000

↳ Codeur en bout de vis à billes au pas de 5mm ⇨ UNITREV_P(Axe)=5

↳ La valeur du paramètre POSMIN_P est de 0 à 32767 incréments, c'est à dire 0 à 40,95875 mm.

10-16- Liste alphabétique

10-16-1- 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 `/'.

10-16-2- 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 `/'.

10-16-3- 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 `/'.

10-16-4- 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.

10-16-5- 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 : '=', '>', '>=', '<=', '<>'.

10-16-6- 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 : '=', '>', '>=', '<', '<>'.

10-16-7- 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 : '>>'.

10-16-8- 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 : '=', '>', '>=', '<', '<='

10-16-9- 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 : '>', '>=', '<', '<=', '<>'

10-16-10- 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 : '=', '>=', '<', '<=', '<>'

10-16-11- 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 : '=', '>', '<', '<=', '<>'.

10-16-12- 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 : '<<'.

10-16-13- 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
```

10-16-14- 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
```

10-16-15- ACC - Accélération

Syntaxe 1 : **ACC**(<Axe>) = <Expression>

Syntaxe 2 : <Variable> = **ACC**(<Axe>)

Syntaxe 3 : **ACC**(<AxeX>,<AxeY>) = <Expression>

Unité : Expression,Variable : unité utilisateur par s² (Ex : mm/s², degré/s², tr/s²…)

Limites : Expression,Variable : de 5.10^-5 à 1,5.10^6 incréments/s

Types acceptés : <Expression> : Octet, Entier, Entier long ou réel

<Variable> : réel

Description : Cette instruction lit ou modifie l'accélération courante.

Remarques : <Expression> doit être une expression réelle valide. L'accélération courante peut être lue et modifiée à tout moment. La syntaxe 3 est utilisée pour l'interpolation où elle représente une accélération résultante des deux axes.

Exemple :
```
ACC(X)=500
```

Voir aussi : DEC, POS et VEL

10-16-16- ACC% - Accélération en pourcentage

Syntaxe : **ACC%**(<Axe>) = <Expression>

Limites : Expression : de 1 à 100

Types acceptés : <Expression> : Octet, Entier, Entier long ou réel

Description : Cette fonction ajuste l'accélération courante en pourcentage du paramètre d'accélération (ACC_P).

Remarques : La valeur de ACC_P peut être entrée dans l'écran "Profil de vitesse" lors de la configuration de la carte.

Exemple :
```
ACC_P(X)=500
ACC%(X)=10 'L'accélération courante est de 50 unités / s²
```

Voir aussi : DEC%

10-16-17- ADC – Entrées analogiques

Syntaxe : <Variable>= **ADC** (<Entrée>)

Unité : Variable : Volt

Limite : Variable : +/- 10V

Types acceptés : <Variable> : réel

Description : Cette fonction retourne la tension d'une entrée analogique +/- 10 V.

Remarques : <Entrée> doit représenter le nom d'une entrée analogique.

Exemple :
```
a!=ADC(Temperature)
```

Voir aussi : DAC

10-16-18- ADDMOV – Superposition de mouvements

Syntaxe : **ADDMOV** (<Esclave>, <Maître>, <Coefficient>)

Unités : <Coefficient> : rapport d'incréments entre le maître et l'esclave

Types acceptés : <Coefficient> : réel

Description : Cette fonction permet de superposer sur un axe esclave ses propres mouvements avec ceux effectuer par un axe maître.(SRV85 seulement)

Remarques : <Esclave> et <Maître> doivent être des axes servo. Le maître et l'esclave peuvent exécuter n'importe quel type de mouvement : positionnement, synchronisation, interpolation. L'esclave devra au préalable être lié à un maître par une fonction d'arbre électrique (GEARBOX ou GEARBOXM), de mouvement synchronisé (MOVS, MOVSM, MOVSP, MOVSC) ou came (CAM, CAMC) avant de lancer une superposition de mouvement.

Voir aussi : ADDSTOP

10-16-19- ADDSTOP – Arrêt superposition de mouvements

Syntaxe : **ADDSTOP** (<AxeEsclave>)

Description : Cette fonction permet de détruire les liens de superposition de mouvements de l'axe esclave.(SRV85 seulement)

Remarques : <AxeEsclave> doit être un axe servo. Si l'axe esclave passe en mode non asservi (AXIS(Esclave)=Off), le lien entre le maître et l'esclave est également coupé.

Voir aussi : ADDMOV

10-16-20- 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

10-16-21- 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

10-16-22- 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

10-16-23- 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

10-16-24- 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$.

10-16-25- AXIS – Contrôle la boucle d'asservissement

Syntaxe : **AXIS** (<Axe>) = ON | OFF

Description : Cette instruction est utilisée pour ouvrir et fermer la boucle d'asservissement.

Remarques : Lorsque l'axe est en boucle fermé (AXIS=ON), toutes les instructions de mouvement sont transmises à la carte d'axe par l'intermédiaire du buffer de mouvement et sont exécutées. Si l'axe passe en boucle ouverte (AXIS=OFF), le buffer de mouvement est vidé, la consigne analogique est forcée à 0, les instructions MOVE_S et FE_S retournent la valeur 0.

☏ Attention :

↳ Toute instruction de mouvement envoyée à un axe qui est en boucle ouverte sera consommée par la carte d'axe mais pas exécutée réellement.

↳ Si un axe passe en erreur de poursuite, tous les axes pilotés par la MCS passent en boucle ouverte.

Exemple :
```
AXIS(X) = ON        'passage en boucle fermé
MOVA(X=1000)        'déplacement à la position 1000
OUT(S1)=1           'Sortie S1=1
MOVA(X=2000)        'Erreur de poursuite détectée sur 1 autre axe
                    'Même si la position 2000 n'est pas atteinte,
                    'La tâche continue et la sortie S1 passe à 0
OUT(S1)=0
```

Voir aussi : AXIS_S, SECURITY

10-16-26- AXIS_S – Lit l'état de la boucle d'asservissement

Syntaxe : **AXIS_S**(<Axe>)

Description : Cette fonction est utilisée pour lire l'état de la boucle d'asservissement et retourne une réponse ON ou OFF.

Remarques : Cette fonction peut être utilisée à tout moment pour voir si l'axe est asservi.

Exemple :
```
MOVA(X=100)
  If AXIS_S(X) = OFF THEN GOTO Error  'Erreur car l'axe est
                                      'passé en boucle ouverte
OUT(ValidVar) = Axis_s(X)       'Gestion d'une sortie validation
                                'variateur à mettre dans une
                                'tâche de surveillance
```

Voir aussi : AXIS

10-16-27- BACKLIGHT – Mise en veille du Dialog 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 Dialog 640 restera allumé si l'utilisateur n'appuie pas sur les touches du pupitre. Par défaut la durée est de 15 minutes.

Remarques : Lorsque le Dialog 640 est en veille, si l'utilisateur appuie sur une touche du Dialog 640 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 Dialog 640 se mettra en veille au bout de
                  '2 heures si l'utilisateur n'appuie pas sur
                  ' les touches du pupitre.
```

10-16-28- BEEP – Emet un son bref

Syntaxe : **BEEP**

Description : Cette instruction émet un son bref sur le pupitre opérateur dialog 80, 160 ou 640.

Remarques : Cette fonction utilise le port de communication #1.Par défaut, le port de communication SERIAL1 sera utilisé. Si un pupitre opérateur Dialog 80, 160 ou 640 est connecté à SERIAL2 veuillez utiliser la fonction OPEN pour affecter #1 au port SERIAL2.

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

Voir aussi : BUZZER

10-16-29- 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

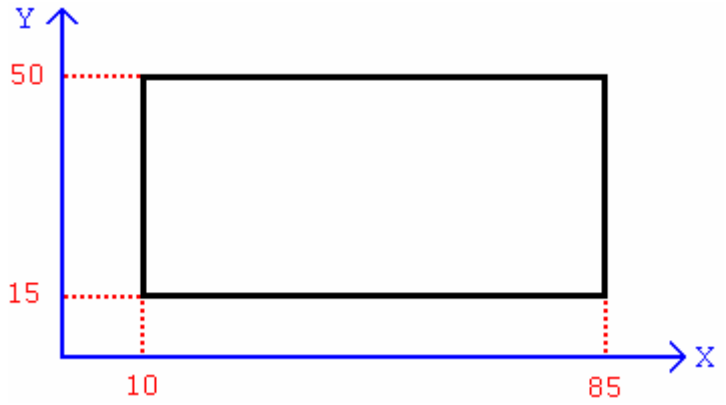
Types acceptés : X1, Y1, X2, Y2, Couleur remplissage : Octet

Couleur cadre : Bit

Description : Trace un rectangle de coordonées X1,Y1 (coin haut, gauche) et X2,Y2 (coin bas, droit) sur le pupitre Dialog 640.

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
```



10-16-30- BUFMOV_S – Nombre d'ordres en attente

Syntaxe : <Variable>=**BUFMOV_S**(<Axe>)

Types acceptés : <Variable> : Octet

Description : Cette fonction retourne le nombre de mouvements en attente dans le buffer d'une carte d'axe. Le mouvement en cours d'exécution n'est pas comptabilisé par cette fonction.

Remarques : Cette fonction peut être utilisée après avoir lancé des mouvements, pour savoir si un mouvement est fini. Dans le cas où le buffer de mouvement se trouve plein, la tâche se bloque jusqu'à ce qu'une place soit libérée.

Exemple :
```
STTR(X)=100
STTR(X)=50
STTR(X)=50
WAIT BUFMOV_S(X)<2       'Attendre la fin du premier mouvement
```

10-16-31- BUZZER – Emet un son continu

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

Description : Cette fonction active ou désactive le buzzer du pupitre opérateur dialog 80, 160 ou 640.

Remarques : Cette fonction utilise le port de communication #1.Par défaut, le port de communication SERIAL1 sera utilisé. Si un pupitre opérateur Dialog 80, 160 ou 640 est connecté à SERIAL2 veuillez utiliser la fonction OPEN pour affecter #1 au port SERIAL2.

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

Voir aussi : BEEP

10-16-32- 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

10-16-33- CAM – Came électronique

Syntaxe : **CAM** (<Esclave>,<Maître>, <Tableau>, <Mono-coup>, <Réversible>, <DirectionPositive>,<Gain>)

Description : Cette fonction lance une came électronique entre deux axes.

Remarques : <Esclave> : Nom de l'axe esclave où est effectuée la came (carte servo : SRV15, SRV85...)

<Maître> : Nom de l'axe maître (carte servo ou codeur : SCD22, SRV15, SRV85...)

<Tableau> : Nom du tableau de came rentré à partir de l'éditeur de données de l'onglet variables globales du logiciel MCB (variable de type « tableau de came »). Les valeurs données aux positions de l'axe maître à l'intérieur de la table doivent être des valeurs croissantes.

<Mono-coup> : Définit le rebouclage automatique de la came. Rentrez la valeur 0 pour une came qui va se reboucler sur son profil jusqu'à ce qu'un arrêt soit demandé, 1 pour une came qui va exécuter son profil une seule fois.

<Réversible> : Indique si l' <Esclave> doit suivre le <Maitre> dans les deux sens.

↳ Rentrez la valeur 0 pour une came non réversible : si le maître se déplace à l'inverse de son sens normal donné par <DirectionPositive>, l'esclave s'arrête ; il repartira lorsque le maître reprendra son sens normal et atteindra la position maître à laquelle l'esclave s'était arrêté.

↳ Rentrez la valeur 1 pour une came réversible : l'esclave suit son profil de came quel que soit le sens d'avance du maître.

<DirectionPositive> : Si la came n'est pas réversible, le sens normal de l'avance du maître doit être indiqué. Rentrez la valeur 0 pour un sens négatif, 1 pour un sens positif.

<Gain> : Le <Gain> est utilisé pour multiplier toutes les positions sur l'axe <Esclave> définies dans le tableau par un facteur.

Exemple :
```
CAM(Master,Slave,Table,0,1,1,2.5) ' Non mono-coup, reversible,
                                  ' sens positif, avec un
                                  ' facteur 2.5
```

Voir aussi : CAMBOX, GEARBOX,, MOVS, CAMC

10-16-34- CAMBOX – Boîte à cames

Syntaxe : **CAMBOX** (<Num boîte>,<Maître>, <CarteSorties>, <Nb seg>, <Nb délais>)

Limites : Num boîte : de 1 à 8

Nb seg : de 1 à 16

Types acceptés : Num boîte : Octet

Nb seg : Octet

Nb délais : Octet

Description : Cette fonction définit une boîte à came. Tout segment (CAMSEG) précédemment défini sur cette boîte est effacé.

Remarques : <Num boite> désigne un numéro de boîte

<Maître> peut être un axe servo, un codeur, ou un codeur temporel.

<CarteSorties> est une carte 8 ou 16 sorties.

<Nb seg> est le nombre maximal de segment dans la boîte. Si cette valeur est nulle la boite à came est détruite et doit être redéfinie si on veut la réutiliser.

<Nb délais> est le nombre de phase d'anticipation.

Exemple :
```
CAMBOX(1,Maitre,S,10,0)  'Boite à came de 10 segments sur les
                         'sorties de la carte S.
```

Voir aussi : CAMBOXSEG, CAMBOXDELAY

10-16-35- CAMBOXDELAY – Délais d'anticipation

Syntaxe : **CAMBOXDELAY** (<Num boîte>, <Num délai>,<Vitesse>, <Délai>)

Limites : Num boîte : de 1 à 8

Num délai : de 1 à 5

Unité : Vitesse : en Unités utilisateur par seconde (Ex : mm/s, degré/s…)

Délai : en millisecondes

Types acceptés : Num boîte, Num délai : Octet

Vitesse : réel

Délai : Entier Long

Description : Cette fonction définit un délai d'anticipation.

Remarques : <Num boîte> désigne la boîte.

<Num délai> est le numéro du délai.

Exemple :
```
CAMBOXDELAY(1,2,100,10)  'Un délai de 10 millisecondes sera
                         'appliqué jusqu'à 100 unités/secondes
```

Voir aussi : CAMBOX

10-16-36- CAMBOXSEG – Segment de boîte à cames

Syntaxe : **CAMBOXSEG** (<Num boîte>, <Num seg>, <Num sortie>, <Début>,<Fin>)

Limites : Num boîte : de 1 à 8

Num sortie : de 1 à 16

Unité : Début, Fin : Unité utilisateur

Types acceptés : Num boîte, Num seg, Num sortie : Octet

Début, Fin :réel

Description : Cette fonction définit un segment d'une boîte à came.

Remarques : <Num boîte> désigne la boîte. <Num seg> désigne le segment. <Num sortie> est la sortie à modifier. La sortie sera mise à 1 entre <Début> et <Fin>.

Exemple :
```
CAMBOXSEG(1,2,4,0,90)    'Le second segment de la boîte 1 met la
                         '4ème sortie à 1 entre 0 et 90°(l'unité
```

```
                                    'utilisateur définie étant le degré).
```

Voir aussi : CAMBOX

10-16-37- CAMC – Came électronique déclenchée sur entrée Capture

Syntaxe : CAMC (<Axe esclave>,<Axe maître>,<Cam table>,<Mono>,<Reverse>,
<DirectionPositive>,<Facteur>,<Configuration>
[,<AxeDéclenchement>,<Fenêtre>,<Mini>,<Maxi>,<Intérieur>])

Types acceptés : <Configuration> : Octet de configuration hardware de <Axe>
b0 : non utilisé
b1 : capture sur top Z
b2 : capture sur entrée n°1 C1
b3 : capture sur entrée n°2 C2
b4 : choix du front : 0 pour front montant, 1 pour front descendant
b5..7 : non utilisés
<DirectionPositive>, <Facteur>, <Mini>, <Maxi> : réel
<Cam table> : tableau de came
<Fenêtre>, <Intérieur> : bit

Description : Cette instruction est équivalente à CAM mais en plus, elle intègre une condition de déclenchement donnée par une entrée rapide « capture ». Ce déclenchement peut se faire dans une fenêtre de position. Les paramètres <AxeDéclenchement>, <Fenêtre>, <Mini>, <Maxi> et <Intérieur> sont optionnels.

Remarques : <AxeMaître> peut-être un axe servo, un codeur ou un codeur temporel.
<AxeEsclave> doit être un axe servo.
<AxeEsclave> doit être à l'arrêt (MOVE_S(AxeEsclave)=0) avant d'envoyer l'ordre sinon le mouvement peut-être obtenu même si la condition n'est pas valide.

Exemple :
```
CAM(Master,Slave,Table,0,1,1,2.5)  ' Non mono-coup, reversible,
                                   ' sens positif, avec un
                                   ' facteur 2.5
```

Voir aussi : CAMBOX, GEARBOX,, MOVS, CAM

10-16-38- CAMNUM_S – Numéro de la came en cours d'exécution

Syntaxe : <Variable> = **CAMNUM_S**(<Esclave>)

Types acceptés : <Variable> : Octet

Description : Cette instruction permet de savoir quel numéro de came est en cours d'exécution.

Remarques : <Esclave> : Nom de l'axe esclave où est effectuée la came (carte servo : SRV15, SRV85...)
La valeur retournée est significative que si CAM_S est à 1.

Exemple :
```
IF CAMNUM_S(Esclave)=1 THEN PRINT «  Came 1 en cours  »
IF CAMNUM_S(Esclave)=2 THEN PRINT «  Came 2 en cours  »
```

Voir aussi : CAM_S, CAMSEG_S

10-16-39- CAMSEG_S – Numéro d'équation de la came en cours d'exécution

Syntaxe : <Variable> = **CAMSEG_S**(<Esclave>)

Types acceptés : <Variable> : Octet

Description : Cette instruction permet de savoir quelle numéro d'équation de la came est en cours d'exécution.

Remarques : <Esclave> : Nom de l'axe esclave où est effectuée la came (carte servo : SRV15, SRV85...)

La valeur retournée est significative que si CAM_S est à 1.

Exemple :
```
IF CAMSEG_S(Esclave)=1 THEN PRINT «  Came entre le point 1
et le point 2  »
IF CAMSEG_S(Esclave)=2 THEN PRINT «  Came entre le point 2
et le point 3  »
```

Voir aussi : CAM_S, CAMNUM_S

10-16-40- CAM_S – Etat de la came

Syntaxe : <Variable> = **CAM_S**(<Esclave>)

Types acceptés : <Variable> : Bit

Description : Cette instruction permet de savoir si une came de la carte est en cours d'exécution.

Remarques : <Esclave> : Nom de l'axe esclave où est effectuée la came (carte servo : SRV15, SRV85...)

Exemple :
```
IF NOT CAM_S(Esclave) THEN PRINT «  Came arrêtée  »
IF CAM_S(Esclave) THEN PRINT «  Came Lancée  »
```

Voir aussi : CAMNUM_S, CAMSEG_S

10-16-41- CAPTURE – Lancement d'une capture de position

Syntaxe : **CAPTURE** (<Axe>,<Inverse>,<Fenêtre>,<Mini>,<Maxi>,<Intérieur>)

Description : Cette instruction est utilisée pour enregistrer la position courante d'un axe.

Remarques : Avec cette instruction la MCS attend un front sur l'entrée capture. Quand le front est détecté, la position est stockée dans la variable REGPOS_S. Le flag REG_S est alors positionné à vrai. <Inversion> permet de traiter le front descendant au lieu du front montant. Si <Fenêtre> est vraie alors l'entrée n'est testée que lorsque <Axe> est entre les positions <Mini> et <Maxi>.

<Interieur> permet de définir si le test s'effectue à l'intérieur ou à l'extérieur des bornes <Mini> et <Maxi>. <Mini> doit toujours être inférieur à <Maxi>.

Exemple :
```
CAPTURE(X,Off,On,10,20,On)      'Capture position sur front
                                'Montant lorsque X est
                                'entre 10 et 20
WAIT REG_S(X)=True
MOVA(Y=500+REGPOS_S(X))
```

Voir aussi : REGPOS_S, REG_S, CAPTURE1, CAPTURE2

10-16-42- CAPTURE1 – Lancement d'une capture de position

Syntaxe : CAPTURE1(<Axe1>,<Axe2>,<Configuration1>,<Fenêtre2>,<Mini2>, <Maxi2>,<Intérieur2>)

Types acceptés : <Configuration1> : Octet de configuration hardware de <Axe1>

b0 : non utilisé

b1 : capture sur top Z

b2 : capture sur entrée n°1 C1

b3 : capture sur entrée n°2 C2

b4 : choix du front : 0 pour front montant, 1 pour front descendant

b5..7 : non utilisés

<Mini2>, <Maxi2> : réel

<Fenêtre2>, <Intérieur2> : bit

Description : Elle permet d'enregistrer la position de n'importe quelle carte servo SRV 85 à partir d'un front de l'une de ces 2 entrées de capture.

Remarques : Avec cette instruction la MCS attend un front sur l'information capture. Quand le front est détecté, la position est stockée dans la variable REGPOS1_S. Le flag REG1_S est alors positionné à vrai. Si <Fenêtre2> est vraie alors l'entrée n'est testée que lorsque <Axe2> est entre les positions <Mini2> et <Maxi2>.

<Interieur2> permet de définir si le test s'effectue à l'intérieur ou à l'extérieur des bornes <Mini2> et <Maxi2>. <Mini2> doit toujours être inférieur à <Maxi2>.

Exemple :

```
CAPTURE1(Y,X,4,On,10,20,On)     'Capture position axe X sur
                                'front montant de l'entrée C1 de
                                'l'axe Y lorsque la position de
                                'l'axe X est entre 10 et 20
WAIT REG1_S(X)=True
Correction  !=Phase-REGPOS1_S(X)
...
```

Voir aussi : REGPOS_S, REG_S, CAPTURE, CAPTURE2

10-16-43- CAPTURE2 – Lancement d'une capture de position

Syntaxe : CAPTURE2(<Axe>,<Configuration>,<Fenêtre>,<Mini>,<Maxi>,<Intérieur>)

Types acceptés : <Configuration> : Octet de configuration hardware de <Axe>

b0 : non utilisé

b1 : capture sur top Z

b2 : capture sur entrée n°1 C1

b3 : capture sur entrée n°2 C2

b4 : choix du front : 0 pour front montant, 1 pour front descendant

b5..7 : non utilisés

<Mini>, <Maxi> : réel

<Fenêtre>, <Intérieur> : bit

Description : Cette instruction est équivalente à l'instruction CAPTURE mais elle intègre en plus le choix de capture sur l'entrée C1 ou C2 ou le top Z du codeur.

Remarques : Avec cette instruction la MCS attend un front sur l'information capture. Quand le front est détecté, la position est stockée dans la variable REGPOS2_S. Le flag REG2_S est alors positionné à vrai. Si <Fenêtre> est vraie alors l'entrée n'est testée que lorsque <Axe> est entre les positions <Mini> et <Maxi>.

<Interieur> permet de définir si le test s'effectue à l'intérieur ou à l'extérieur des bornes <Mini> et <Maxi>. <Mini> doit toujours être inférieur à <Maxi>.

Exemple :

```
CAPTURE2(X,24,Off,0,0,Off)      ' Capture position axe X sur
                                ' front descendant de l'entrée
                                ' C2 de l'axe X
WAIT REG2_S(X)=True
Correction!=Phase-REGPOS2_S(X)
...
```

Voir aussi : REGPOS_S, REG_S, CAPTURE, CAPTURE1

10-16-44- 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:
```

10-16-45- 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

10-16-46- 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

10-16-47- 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

10-16-48- CLEAR – Met à zéro la position d'un axe

Syntaxe : **CLEAR** (<Axe>)

Description : Cette instruction met à zéro la position d'un axe.

Remarques : <Axe> peut être une carte codeur ou servo.

Exemple :
```
CLEAR(X)
A!=POS_S(X)        'Résultat : A!=0.0
```

10-16-49- CLEARCOUNTER – remise à zéro du compteur

Syntaxe : **CLEARCOUNTER**(<Axe>,<Entrée>)

Types acceptés : <Entrée> : Octet

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

Remarques : <Axe> : Nom de la carte servo

<Entrée> : Numéro de l'entrée (1 pour l'entrée C1, 2 pour l'entrée C2)

Voir aussi : COUNTER_S, SETUPCOUNTER

10-16-50- 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

10-16-51- 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

10-16-52- 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

10-16-53- 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.

10-16-54- CLS – Efface l'écran du terminal

Syntaxe : **CLS**

CLS 1, CLS 2, CLS 3 ou CLS 4 (uniquement avec le Dialog 160 et 80)

CLS B, CLS W (uniquement avec le Dialog 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 DIALOG 160 et 80. La fonction CLS B efface l'écran du Dialog 640 avec un fond noir. La fonction CLS W efface l'écran du Dialog 640 avec un fond blanc.

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 utiliser la fonction OPEN afin d'affecter #1 au port SERIAL2.

10-16-55- CONS – Tension de consigne

Syntaxe : **CONS**(<Axe>)=<Expression>

Unité : Expression : Volt

Limites : Expression : de –10 à +10

Types acceptés : Expression : réel

Description : Cette instruction ouvre la boucle d'asservissement de l'axe et fixe la tension de consigne à la valeur <Expression>**.**

Remarques : Attention, dans ce cas l'erreur de poursuite n'est plus contrôlée. <Axe> doit être un axe servo. Cette instruction est utile pour tester le variateur et le moteur en boucle ouverte.

Exemple :
```
CONS(X)=5.0        'Consigne à 5V
```

Voir aussi : CONS_S

10-16-56- CONS_S – Etat de la tension de consigne

Syntaxe : **CONS_S**(<Axe>)

Description : Cette fonction retourne l'image de la tension de consigne en boucle ouverte ou fermée.

Remarques : <Axe> doit être un axe servo. Cette fonction est utilisée pour connaître la valeur courante de la tension de consigne.

Exemple :
```
a!=CONS_S(X)
PRINT  #1,a!
```

Voir aussi : CONS

10-16-57- 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

10-16-58- CORRECTION – Fonction de compensation

Syntaxe : **CORRECTION**(<Esclave>,<Maître>,<Mode>,<Capture>,<Configuration>,
<Fenêtre>,<Mini>,<Maxi>,<Intérieur>,<Phase>,
<Dist.maître>,<Vit.maxi esclave>,<Vit.mini esclave> ,
<Dist.accél.mini>)

Unités : <Dist.maître> : unité utilisateur

<Vit.maxi esclave>, <Vit.mini esclave> : unité utilisateur/s

<Dist.accél.mini> : unité utilisateur/s²

Types acceptés : <Mode>, <Capture>, <Fenêtre>, <Interieur> : bit

<Configuration> : Octet de configuration hardware de la capture

b0 : non utilisé

b1 : capture sur top Z

b2 : capture sur entrée n°1 C1

b3 : capture sur entrée n°2 C2

b4 : choix du front : 0 pour front montant, 1 pour front descendant

b5..7 : non utilisés

<Phase>, <Maxi>, <Mini>, <Dist.maître>, <Vit.maxi esc.>, <Vit.mini esc.>,

<Dist.acc.mini> : réel

Description : Cette fonction permet d'appliquer un mouvement de correction sur un axe esclave pendant une distance de l'axe maître sur un événement de capture de position.(SRV85 seulement)

Remarques : <Esclave> doit être un axe servo (SRV85) pour effectuer la compensation et <Maître> peut être un axe servo ou un codeur. Mode définit le mode mono-coup (0) ou permanent (1). Capture précise le registre de capture : registre 1 du maître (0) ou registre 1 de l'esclave (1). Fenêtre permet d'indiquer si la compensation doit s'effectuer dans une fenêtre. Maxi doit être supérieur à Mini. Intérieur permet de définir si la compensation s'effectue à l'intérieur (0) ou l'extérieur (1). Phase permet de définir Si la valeur de position lors d'une capture est égale à la phase, aucune compensation ne sera appliquée. Comme tous les paramètres sont stockés dans la carte de l'axe esclave, dès que l'événement est détecté, la compensation est traitée instantanément. Distance maître représente l'intervalle de correction. Vit.maxi esc. est la vitesse maxi appliquée lors de la compensation. Vit.mini esc. est la vitesse mini appliquée lors de la compensation. <Dist.acc.mini> définit la portion d'accélération ou de décélération de l'esclave sur une distance du maître. Si

CORRECTION est déjà activée (en mode permanent), on peut en temps réel la réexécuter pour faire une modification de paramètres (par exemple la phase). Par contre, il est nécessaire de remettre tous les paramètres de l'instruction même ce qui ne sont pas modifié.

Attention :

- La compensation précédente doit être complètement terminée avant de pouvoir en traiter une autre.
- Si le maître tourne en sens opposé à son sens normal (donné par le signe de « distance maître »), la compensation n'est pas appliquée et elle est perdue.
- L'esclave devra au préalable être lié à un maître par une fonction d'arbre électrique (GEARBOX ou GEARBOXM), de mouvement synchronisé (MOVS, MOVSM, MOVSP, MOVSC) ou came (CAM, CAMC) avant de lancer une compensation.
- Si la correction force à passer en dehors des limites définies par l'utilisateur, l'erreur est signalée et la correction maximale autorisée est appliquée.

Voir aussi : STOPCORRECTION, ICORRECTION

10-16-59- CORRECTION_S – Etat de la compensation

Syntaxe : <Variable>=**CORRECTION_S**(<Esclave>)

Types acceptés : <Variable> : octet

b0=1 : compensation demandée et en attente de l'événement capture.

b1=1 : compensation en cours.

b2=1 : erreur détectée : compensation hors limites (la compensation limitée à quand même été appliquée).

Description : Cette fonction permet de connaître l'état du cycle de compensation.

Remarques : <Esclave> doit être un axe servo SRV85. En mode rebouclé, la valeur retournée par CORRECTION_S varie entre « compensation en attente » et « compensation en cours » (| b0=1 b1=0 | b0=0 b1=1 | b0=1 b1=0 |...).

En mode mono-coup, elle passe par : | b0=1 b1=0 | b0=0 b1=1 | b0=0 b1=0 |.

Le bit d'erreur b2 est automatiquement remis à 0 après chaque lecture de CORRECTION_S ou au départ de la compensation suivante.

Voir aussi : CORRECTION, ICORRECTION

10-16-60- 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

10-16-61- COUNTER_S – lecture du compteur

Syntaxe : <Variable>=**COUNTER_S**(<Axe>,<Entrée>)

Types acceptés : <Variable> : Entier

<Entrée> : Octet

Description : Cette instruction permet de lire le compteur

Remarques : <Axe> : Nom de la carte servo

<Entrée> : Numéro de l'entrée (1 pour l'entrée C1, 2 pour l'entrée C2)

Voir aussi : SETUPCOUNTER, CLEARCOUNTER

10-16-62- 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$)
```

10-16-63- 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.

10-16-64- 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$, MKLR$

10-16-65- 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$, MKLR$

10-16-66- 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$, MKIR$

10-16-67- 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$, MKIR$

10-16-68- DAC – Sorties analogiques

Syntaxe : **DAC** (<Sortie>) = <Expression>

Unité : Expression : Volts

Limites : Expression : de 0 à +10

Types acceptés : Expression : réel

Description : Cette fonction envoie une tension sur une sortie analogique d'une carte SOA12.

Remarques : <Sortie> doit représenter un nom de sortie analogique. Les sorties analogiques peuvent également être lues.

☏ Attention : Si on utilise la sortie +/- 10V de la carte SOA12 (Ex : pin1 ou pin 2 du SUBD), <Expression>=0 donne -10V sur la sortie analogique, <Expression>=5 donne 0V et <Expression>=10 donne 10V.

Exemple :
```
DAC(Ana2)=5.2
IF ADC(Ana1)>DAC(Ana2) THEN ...
```

Voir aussi : ADC

10-16-69- 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

10-16-70- DEC - Décélération

Syntaxe 1 : **DEC**(<Axe>) = <Expression>

Syntaxe 2 : <Variable> = **DEC**(<Axe>)

Unité : unité utilisateur par s² (Ex : mm/s², degré/s², tr/s²…)

Limites : de 5.10^-5 à 1,5.10^6 incrément/s

Types acceptés : Variable, Expression : réel

Description : Cette instruction lit ou modifie la décélération courante en unités par s².

Remarques : La décélération courante peut être lue et modifiée à tout moment.

Exemple : `DEC(X)=500.`

Voir aussi : ACC, VEL

10-16-71- DEC% - Décélération en pourcentage

Syntaxe : **DEC%**(<Axe>) = <Expression>

Limites : Expression de 0 à 100

Types acceptés : Expression : réel

Description : Cette fonction fixe la décélération courante en pourcentage du paramètre de décélération (DEC_P).

Remarques : La valeur de DEC_P peut être entrée dans l'écran profil de vitesse lors de la configuration de la carte.

Exemple :
```
DEC_P(X)=500
DEC%(X)=10 'Décélération courante 50 unités / s²
```

Voir aussi : ACC% et VEL%

10-16-72- 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
```

10-16-73- 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

10-16-74- DISPLAY – Afficheur 7 segments

Syntaxe : **DISPLAY** "<Car>" ou **DISPLAY** <Expression>

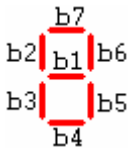
Types acceptés : Car : caractère de '0' à '9'

Expression : Octet

Description : Cette Instruction fixe l'état de l'afficheur status de la MCS32

Remarque : L'instruction utilisant <Car> permet d'afficher des caractères pré-programmés ("0" à "9") . Pour afficher autre chose, il faut utiliser l'instruction utilisant <Expression>. Chaque bit de <Expression> représente un segment. Le bit b0 n'est pas utilisé.

Exemple :
```
Display "5"       ' Equivalent à Display 10110110b
```



10-16-75- 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

10-16-76- 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 les Dialog 80 et 160 ou de 1 à 16 pour le Dialog 640.

Colonne : de 1 à 20 pour le Dialog 80 ou de 1 à 40 pour les Dialog 160 et 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
```

10-16-77- EDIT$

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

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

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

Types acceptés : Variable : chaîne de caractères

Ligne, Colonne, Longueur : Entier

Description : Cette fonction permet d'éditer une chaîne de caractères avec le Dialog 80 et le Dialog 640 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.
```

10-16-78- 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

10-16-79- ENDCAM – Arrêt d'une came

Syntaxe : **ENDCAM**(<Esclave>)

Description : Cette fonction permet d'arrêter le mouvement de l'axe <Esclave> à la fin de la came. Elle diffère de la fonction STOP qui met fin au mouvement immédiatement.

Remarques : Attention : Si ENDCAM s'applique à une came qui a été déclarée en mode non mono-coup et enchaînée avec une autre, la came termine son profil et enchaîne sur la suivante.

Voir aussi : CAM, STOP

10-16-80- EXIT SUB – Sortie d'un sous-programme

Syntaxe : **EXIT SUB**

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

Voir aussi : SUB

10-16-81- 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

10-16-82- FEMAX_S – Limite d'erreur de poursuite

Syntaxe : **FEMAX_S**(<Axe>)

Description : Cette fonction indique le dépassement de l'erreur de poursuite maximale.

Remarques : <Axe> doit être un axe servo. Cette fonction est utilisée pour savoir quel axe est passé en erreur de poursuite. Il est nécessaire de traiter ce flag dans une tâche de gestion des défauts si l'instruction SECURITY(0,0) ou SECURITY(0,1) a été utilisée.

Exemple :
```
IF FEMAX_S(X) THEN PRINT "Défaut Axe X"
IF FEMAX_S(Y) THEN PRINT "Défaut Axe Y"
```

Voir aussi : FE_S, SECURITY

10-16-83- FE_S

Syntaxe : **FE_S**(<Axe>)

Description : Cette fonction retourne une image de l'erreur de poursuite courante.

Remarques : <Axe> doit être un axe servo. Cette fonction est utilisée pour connaître la valeur courante de l'erreur de poursuite. on peut ainsi vérifier le comportement de la régulation en temps réel.

Exemple :
```
a!=FE_S(X)
PRINT  #1,a!
```

Voir aussi : FEMAX_S

10-16-84- 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

10-16-85- 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

10-16-86- FONT – Sélection police

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

Types acceptés : <Valeur> : octet.

Description : Cette fonction permet de définir une police sur le pupitre Dialog 640.

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"
```

10-16-87- 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

10-16-88- 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 "
```

10-16-89- 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

10-16-90- GEARBOX – Arbre électrique

Syntaxe : **GEARBOX**(<AxeEsclave>,<AxeMaître>,<Numérateur>,<Dénominateur>, <Réversible>,[<Accélération>])

Types acceptés : Numérateur, Dénominateur : réel

<Accélération> : Octet, Entier, Entier long ou réel

Réversible : bit

Description : Cette fonction lie 2 axes en arbre électrique.

Remarques : <AxeMaître> peut être un axe servo un codeur ou un codeur temporel.

<AxeEsclave> doit être un axe servo.

Le rapport de l'arbre est défini par <Numérateur> / <Dénominateur>.

<Réversible>est un booléen qui indique que l'arbre est réversible. <Accélération> est un paramètre optionnel qui permet de définir une phase d'accélération.

L'instruction est non-bloquante. La liaison établie entre maître et esclave est valide tant que l'instruction STOP(esclave) ou AXIS(esclave)=OFF n'a pas été activée.

Exemple :
```
GEARBOX(Esclave,Maître,-1,2,1) 'Arbre réversible et l'axe
                               'esclave va tourner 2 fois moins
                               'vite que l'axe maître et dans le
                               'sens inverse.
```

Voir aussi : CAM, CAMBOX, MOVS, STOP, AXIS

10-16-91- GEARBOXRATIO – Change le rapport de réduction d'un arbre électrique

Syntaxe : **GEARBOXRATIO**(<AxeEsclave>,<Ratio>)

Types acceptés : <Ratio> : réel

Description : Cette fonction modifie le rapport de la liaison arbre électrique.

Remarques : <AxeEsclave> : doit être un axe servo esclave d'une liaison arbre électrique. Le rapport de l'arbre est défini par <Ratio>×<Numérateur>/<Dénominateur>. <Numérateur> et <Dénominateur> sont les paramètres de l'instruction GEARBOX.

L'instruction est non-bloquante et permet de changer de ratio sans arrêter l'arbre électrique. Le ratio peut être positif ou négatif.

Exemple :
```
GEARBOX(Esclave, Maître, 1, 2, 1)  'Rapport nominal  : ratio 0.5
GEARBOXRATIO(Esclave,1.2)          '20% d'augmentation
                                   'Ratio  : (1.2×1/2)=0.6
GEARBOXRATIO(Esclave,0.8)          '20% de réduction
                                   'Ratio  : (0.8×1/2)=0.4
```

Voir aussi : CAM, CAMBOX, MOVS, STOP, AXIS

10-16-92- 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

10-16-93- 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

10-16-94- GETTIME – Heure courante

Syntaxe : **GETTIME**(<Heure>,<Minute>,<Seconde>)

Types acceptés : <Heure>, <Minute>, <Seconde> : Entier.

Description : Cette instruction lit l'heure courante.

Voir aussi : GETDATE

10-16-95- 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

10-16-96- 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

10-16-97- 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.

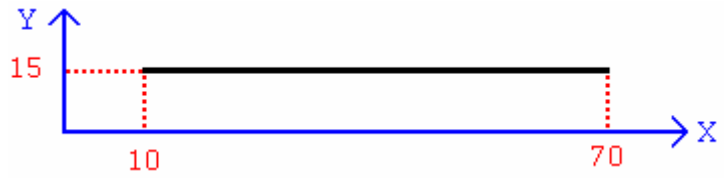
Couleur : Bit

Description : Trace une ligne horizontale avec le point de départ en X1, Y1 et d'arrivée en X2, Y1 sur le pupitre opérateur Dialog 640.

Remarques : Couleur permet de modifier la couleur du trait : noir (0) ou blanc (1)

Exemple :

```
HLINE(10,15,70,0)
```



10-16-98- HOME– Prise d'origine

Syntaxe : **HOME**(<Axe>[, <Type>])

Description : Cette fonction force <Axe> à se déplacer vers sa position d'origine en utilisant le <Type> de prise d'origine choisi. Cette instruction est bloquante pour la tâche tant que la prise d'origine n'est pas terminée et provoque le basculement vers la tâche suivante. La prise d'origine s'effectue à la vitesse VELHOME_P. Le capteur utilisé est indiqué dans INPHOME_P . A la fin de la prise d'origine l'axe se dégage à la position définie par DISHOME_P et la vitesse est maintenue à la valeur VELHOME_P.

Les valeurs de <Type> sont :

0 : Origine immédiate

1 : Sur zéro codeur en sens +

2 : Sur zéro codeur en sens -

3 : Sur capteur en sens +

4 : Sur capteur en sens -

5 : Rapide, sur capteur en sens +

6 : Rapide, sur capteur en sens -

7 : Sur capteur et zéro codeur en sens +

8 : Sur capteur et zéro codeur en sens -

9 : Rapide, sur capteur et zéro codeur en sens +

10 : Rapide, sur capteur et zéro codeur en sens -

Remarques : Utilisez AXIS(<Axe>)=Off pour arrêter une prise d'origine en cours. Si <Type> n'est pas spécifié, la valeur est celle indiquée dans l'écran prise d'origine de la configuration de la carte. Les paramètres VELHOME_P, DISHOME_P et INPHOME_P peuvent être choisis dans ce même écran. Ils devront être transférés dans la MCS au moyen de la commande "Envoyer la configuration" du menu Communication.

Exemple :

```
HOME(X)
```

Voir aussi : HOME_S

10-16-99- HOME_S– Etat de la prise d'origine

Syntaxe : **HOME_S**(<Axe>)

Description : Cette fonction indique l'état de la prise d'origine

Remarques : <Axe> doit être un axe servo. Cette fonction est utilisée pour savoir si la prise d'origine a été effectuée ou non. Pendant un cycle de prise d'origine, l'indicateur HOME_S est forcé en 0. Dès que le cycle est entièrement terminé, HOME_S passe à 1.

Exemple :
```
IF NOT HOME_S(X) THEN HOME(X)
```

Voir aussi : HOME

10-16-100- ICORRECTION – fonction de compensation

Syntaxe : **ICORRECTION**(<Esclave>,<Maître>,<Dist.maître>,<Dist.esclave>, <Dist. d'accél>)

Unités : <Dist.maître>, <Dist.esclave> : unité utilisateur (Ex : mm, degré,…)

<Dist.d'accél> : unité utilisateur/s²

Types acceptés : <Dist.maître>, <Dist.esclave>, <Dist.d'accél> : réel

Limites :

Description : Cette fonction permet d'appliquer un mouvement de correction sur un axe esclave pendant une distance de l'axe maître.

Remarques : L'esclave devra au préalable être lié à un maître par une fonction d'arbre électrique (GEARBOX ou GEARBOXM), de mouvement synchronisé (MOVS, MOVSM, MOVSP, MOVSC) ou came (CAM, CAMC) avant de lancer une compensation. Au mouvement de synchronisation normal de l'esclave, on superpose le mouvement suivant : Pendant que le maître parcourt une « distance maître », on ajoute un déplacement « distance esclave » avec une accélération et une décélération sur une distance maître de « distance d'accél ». <Esclave> doit être une carte servo SRV85 et <Maître peut-être une carte servo ou un codeur.

Voir aussi : CORRECTION

10-16-101- 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

10-16-102- 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
```

10-16-103- 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

10-16-104- 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

10-16-105- 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

10-16-106- 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

10-16-107- 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

10-16-108- 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

10-16-109- 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

10-16-110- 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

10-16-111- 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

10-16-112- 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

10-16-113- 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

10-16-114- 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$

10-16-115- LED – Pilotage des leds

Syntaxe : **LED**(numéro)=Etat

Types acceptés : Etat : octet.

Description : Cette fonction permet de piloter les leds des pupitres opérateurs Dialog 80 et 640.

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
```

10-16-116- 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

10-16-117- 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

10-16-118- LIMMAX_S – Etat de la butée maximale

Syntaxe : **LIMMAX_S**(<Axe>)

Description : Cette fonction indique si la position de l'axe est supérieure à sa butée maximale.

Remarques : <Axe> doit être un axe servo. Cette fonction est utilisée pour savoir si la butée maximale a été franchie.

Exemple :
```
SUB Butees
IF LIMMAX_S THEN STOP(X) 'si butée maxi atteinte, alors
END SUB                  'arrêt de l'axe
```

Voir aussi : LIM_S, LIMMIN_S

10-16-119- LIMMIN_S – Etat de la butée minimale

Syntaxe : **LIMMIN_S**(<Axe>)

Description : Cette fonction indique si la position de l'axe est inférieure à sa butée minimale.

Remarques : <Axe> doit être un axe servo. Cette fonction est utilisée pour savoir si la butée minimale a été franchie

Exemple :
```
SUB Butees
IF LIMMIN_S THEN STOP(X) 'si butée maxi atteinte, alors
END SUB                  'arrêt de l'axe
```

Voir aussi : LIM_S, LIMMAX_S

10-16-120- LIM_S – Etat des butées

Syntaxe : **LIM_S**(<Axe>)

Description : Cette fonction indique si l'axe est en dehors de ses butées mini et maxi.

Remarques : <Axe> doit être un axe servo. Cette fonction est utilisée pour savoir si au moins une des butées a été franchie.

Exemple :
```
SUB Butees
IF LIM_S THEN STOP(X)    'si butée maxi atteinte, alors
END SUB                  'arrêt de l'axe
```

Voir aussi : LIMMAX_S, LIMMIN_S

10-16-121- LOADCAMEX – charge une came dans la carte servo

Syntaxe : **LOADCAMEX**(<Esclave>,<Maître>,<N°came>,<Tableau>,<Nombre>,
<PremierPolynôme>, <Mono-coup>, <Réversible>,
<DirectionPositive>,<N°came suivante>,
<N°came précédente>)

Limites : <PremierPolynôme> : de 1 à 310
<N°Came>, <N°came suivante>, <N°came précédente> : de 1 à 5

Types acceptés : <PremierPolynôme>, <Nombre> : Entier
<N°Came>, <Nombre>, <N°came suivante>, <N°came précédente> : Octet
<Mono-coup>, <Réversible>, <DirectionPositive> : bit
<Tableau> : Tableau de came

Description : Cette instruction permet de charger une came dans la carte servo.

Remarques : <Esclave> : Nom de l'axe esclave où est effectuée la came (carte servo : SRV15, SRV85...)
<Maître> : Nom de l'axe maître (carte servo ou codeur : SCD22, SRV15, SRV85...)
<N°came> : numéro de la came

<Tableau> : Nom du tableau déclaré à partir de l'onglet variables globales du logiciel MCB (variable de type « réel »).

<Nombre> : nombre d'éléments du tableau pour définir la came.

Nombre = (NombrePointsCame + 2) *2

<PremierPolynôme> : Une carte servo possède une table globale de 310 polynômes pour l'ensemble des 5 cames. Rentrez une valeur entre 1 et 310 pour indiquer où sera stocké le premier polynôme de la came dans la table globale de la carte.

Du tableau de came est extrait un tableau de (NombrePointsCame-1) polynômes.

☏ Attention : <PremierPolynôme> + <NombrePointsCame-1> doit être inférieur à 310.

<Mono-coup> : Définit le rebouclage automatique de la came.

↳ Rentrez la valeur 0 pour une came qui va se reboucler sur son profil jusqu'à ce qu'un arrêt soit demandé, 1 pour une came qui va exécuter son profil une seule fois.

<Réversible> : Indique si l' <Esclave> doit suivre le <Maitre> dans les deux sens.

↳ Rentrez la valeur 0 pour une came non réversible : si le maître se déplace à l'inverse de son sens normal donné par <DirectionPositive>, l'esclave s'arrête ; il repartira lorsque le maître reprendra son sens normal et atteindra la position maître à laquelle l'esclave s'était arrêté.

↳ Rentrez la valeur 1 pour une came réversible : l'esclave suit son profil de came quel que soit le sens d'avance du maître.

<DirectionPositive> : Si la came n'est pas réversible, le sens normal de l'avance du maître doit être indiqué.Rentrez la valeur 0 pour un sens négatif, 1 pour un sens positif.

<N°came suivante> : Mettez 0 si la came ne doit pas être enchaînée sur une autre came. Dans le cas contraire, rentrez le numéro de la came suivante compris entre 1 et 5.

<N°came précédente> : Mettez 0 si la came n'enchaînera pas sur une came précédente. Dans le cas contraire, rentrez le numéro de la came précédente compris entre 1 et 5.

Voir aussi : STARTCAM

10-16-122- LOADPOINT – Le point d'une came dans la carte servo

Syntaxe : **LOADPOINT**(<Esclave>,<Maître>,<Tableau>,<Nombre>,
<PremierPolynôme>, <IndexVariable)

Limites : <PremierPolynôme> : de 1 à 310

Types acceptés : <PremierPolynôme>, <IndexVariable>, <Nombre> : Entier

<Tableau> : Tableau de came

Description : Cette instruction permet de changer deux polynômes à partir d'un point d'une came dans la carte servo.

Remarques : <Esclave> : Nom de l'axe esclave où est effectuée la came (carte servo : SRV15, SRV85...)

<Maître> : Nom de l'axe maître (carte servo ou codeur : SRV15, SRV85, SCD22...)

<Tableau> : Nom du tableau déclaré à partir de l'onglet variables globales du logiciel MCB (variable de type « réel »).

<Nombre> : nombre d'éléments du tableau pour définir la came.

Nombre = (NombrePointsCame + 2) × 2

<PremierPolynôme> : Une carte servo possède une table globale de 310 polynômes pour l'ensemble de 5 cames. Rentrez une valeur entre 1 et 310 pour indiquer où sera stocker le premier polynôme de la came dans la table globale de la carte.

<IndexVariable> : Index de la variable dans le <Tableau> où se situe la modification à prendre en compte. La modification des polynômes tient compte des positions du maître et de l'esclave. Il n'est pas nécessaire d'utiliser LOADPOINT deux fois pour modifier ces deux informations.

Voir aussi : LOADCAMEX, STARTCAM

10-16-123- LOADS – Charge un mouvement synchronisé

Syntaxe : **LOADS**(<AxeEsclave>, <AxeMaître>, <DistanceMaître>,
<DistanceEsclave>,<DistanceAccélération>, <DistanceDécélération>)

Unité : DistanceMaître, DistanceEsclave, DistanceAccélération,
DistanceDécélération : Unité utilisateur (Ex : mm)

Types acceptés : DistanceMaître, DistanceEsclave, DistanceAccélération,
DistanceDécélération : réel

Description : Cette instruction est utilisée pour lier un axe esclave à un axe maître pendant une certaine distance de l'axe maître avec des phases d'accélération et de décélération sur l'axe esclave.

<AxeMaître> peut être un axe servo ou codeur.<AxeEsclave> doit être un axe servo. Les <DistanceAccélélération> et <DistanceDécélération> sont exprimées en distance sur l'axe maître et peuvent être nulles.

Remarques : L'instruction LOADS est similaire à MOVS mais l'instruction STARTS doit être utilisée après LOADS pour commencer le mouvement synchronisé. Cette caractéristique est très utile pour commencer simultanément plusieurs mouvements synchronisés.

Exemple :

```
LOADS(Slave1,Master,90,180,10,10)     'chargement du mouvement
                                      'esclave 1 dans le buffer
LOADS(Slave2,Master,90,180,10,10)     'chargement du mouvement
                                      'esclave 2 dans le buffer
STARTS(Slave1,Slave2)                 'Liaison des esclaves avec
                                      'le maître
```

Voir aussi : MOVS, STARTS

10-16-124- LOCATE – Positionne le curseur

Syntaxe : **LOCATE** <Ligne>,<Colonne>

Limites : Ligne : de 1 à 4 pour les Dialog 80 et 160 ou de 1 à 16 pour le Dialog 640.
Colonne : de 1 à 20 pour le Dialog 80 ou de 1 à 40 pour les Dialog 160 et 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>"
```

10-16-125- 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

10-16-126- 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&)
```

10-16-127- LOOP – Mode virtuel

Syntaxe : **LOOP**(<Axes>)=ON/OFF

Description : Cette fonction passe l'axe en mode virtuel et permet de tester un programme sans codeur ni moteur. Elle permet aussi de gérer un axe maître sous la forme d'une base de temps et bénéficiant de toutes les possibilités d'un axe réel (Accélération, Décélération, Vitesse, …).

10-16-128- 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$

10-16-129- MERGE – définit l'enchaînement

Syntaxe : **MERGE**(<Axe>)=ON | OFF Ou **MERGE**(<AxeX>,<AxeY>)

Description : Cette instruction est utilisée pour activer ou désactiver l'enchaînement des mouvements consécutifs. La seconde forme est utilisée pour l'interpolation.

Exemple :
```
MERGE(X)=ON
TRAJ(POS(X)=1000,VEL(X)=500)    'Mouvements enchaînés sans
TRAJ(POS(X)=1500,VEL(X)=20)     'passage par une vitesse nulle
MERGE(X)=OFF
TRAJ(POS(X)=1800,VEL(X)=1000)   'passage par une vitesse nulle
                                'à la position 1500
```

10-16-130- 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$, RIGHT$

10-16-131- 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

10-16-132- 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

10-16-133- 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

10-16-134- 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

10-16-135- 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

10-16-136- MODIFYEVENT– Configuration des événements

Syntaxe : MODIFYEVENT (<Masque>,<Durée>)

Limites : <Durée> : de 10ms à 30000ms

Types acceptés : <Masque> : Entier

<Durée> : Entier

Description : Elle permet de configurer les événements souhaités.

Remarques : <Masque> :

↳ Bits 0...6 : activation des entrées n° 1 à n° 7 de la première carte entrée détectée par le système (la recherche se fait du slot A vers le slot J). 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 7 : base de temps

↳ Bits 8...15 : activation de la capture C1 ou C2 associée à l'instruction Capture1 des 1 à 8 cartes SRV 85 détectées par le système (la recherche se fait du slot A vers le J).

<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

10-16-137- MOVA – Mouvement absolu

Syntaxe : **MOVA**(<Axe>=<Distance> {,<Axe>=<Distance> ... })

Unité : Distance : unité utilisateur (Ex : mm, degré,…)

Types acceptés : Distance : réel

Description : Déplace l'axe à une position absolue. L'exécution de l'instruction provoque le basculement vers la tâche suivante.

Remarques : La tâche attend la fin du mouvement (condition MOVE_S(Axe)=0) avant d'exécuter la prochaine instruction. Les axes utilisent les valeurs courantes de vitesse, d'accélération et de décélération.<Axe> doit être un axe servo.

Exemple : MOVA(X=1200.00,Y=-100.00,Z=+550.00,W=Dist!)

Voir aussi : MOVR, STTA, STTR, STTI et MOVE_S

10-16-138- MOVAC – Mouvement absolu déclenché sur entrée Capture

Syntaxe : MOVAC(<AxeEsclav>=Distance,<Configuration>[,<AxeMaître>,<Fenêtre>, <Mini>,<Maxi>,<Intérieur>])

Types acceptés : Distance, Mini, Maxi : réel

Configuration : Octet

- b0 : non utilisé
- b1 : déclenchement sur top Z
- b2 : déclenchement sur entrée n°1 (C1)
- b3 : déclenchement sur entrée n°2 (C2)
- b4 : 1 : front descendant ; 0 : front montant
- b5…b7 : non utilisés

Intérieur : bit

Description : Cette instruction permet de définir un mouvement sur un axe esclave et un axe maître lorsque l'entrée capture est activée. L'exécution de l'instruction provoque le basculement vers la tâche suivante.

Remarques : Le système attend la fin du mouvement (condition MOVE_S(Axe)=0) avant d'exécuter la prochaine instruction. <AxeEsclave> doit être à l'arrêt (MOVE_S(AxeEsclave)=0) avant d'envoyer l'ordre sinon le mouvement peut-être obtenu même si la condition n'est pas valide. <AxeMaître> et <AxeEsclave> peuvent être représenté par le même axe. Les axes utilisent la vitesse, l'accélération et la décélération courante. <Mini> et <Maxi> définissent la fenêtre de déclenchement utilisée pour démarrer le mouvement sur l'esclave. <Mini> doit être inférieure à <Maxi>. <Intérieur> est utilisé pour indiquer si le déclenchement s'effectue à l'intérieur ou à l'extérieur de la fenêtre. Les paramètres <AxeMaître>, <Fenêtre>, <Mini>, <Maxi> et <Intérieur> sont optionnels.

Exemple :

```
MOVAC (X=200,X,4,0,0,0,0)      'envoi de X en 200 sur front montant
                               ' de son entrée n°1
MOVAC (X=500,Y,24,0,0,0,0)     'envoi de X en 500 sur front descendant
                               ' de l'entrée n°2 de l'axe Y
```

Voir aussi : MOVA, MOVR, STTA, STTR, STTI, MOVE_S

10-16-139- MOVAP – Mouvement absolu déclenché

Syntaxe : **MOVAP**(<AxeEsclave>=<Distance> ,<AxeMaître>,<PositionMini>, <PositionMaxi>,<Intérieur>)

Types acceptés : Distance, PositionMini, PositionMaxi : réel

Intérieur : bit

Description : Cette instruction permet de définir un mouvement sur un axe esclave lorqu'un axe maître entre dans une fourchette de position. L'exécution de l'instruction provoque le basculement vers la tâche suivante.

Remarques : Le système attend la fin du mouvement (condition MOVE_S(Axe)=0) avant d'exécuter la prochaine instruction.Les axes utilisent la vitesse, l'accélération et la décélération courante. <PositionMini> et <PositionMaxi> définissent la fenêtre de déclenchement utilisée pour démarrer le mouvement sur l'esclave. <PositionMini> doit être inférieure à <PositionMaxi>. <Intérieur> est utilisé pour indiquer si le déclenchement s'effectue à l'intérieur ou à l'extérieur de la fenêtre.

Exemple : MOVAP (Slave=1200.00,Master,0,0.1,True)

Voir aussi : MOVA, MOVR, STTA, STTR, STTI, MOVE_S

10-16-140- MOVC– Mouvement circulaire

Syntaxe : MOVC (<AxeX> = <DestinationX>, <AxeY> = <DestinationY>, <CentreX>, <CentreY>, <SensHoraire>)

Types acceptés : DestinationX, DestinationY, CentreX, CentreY : réel

SensHoraire : Bit

Description : Cette instruction est utilisée pour faire un mouvement circulaire avec deux axes. Le mouvement est envoyé dans le buffer de mouvement. Si le buffer est plein, la tâche se bloque jusqu'à ce qu'une place se libère dans le buffer. L'exécution de l'instruction provoque le basculement vers la tâche suivante.

Remarques : <AxeX> et <AxeY> doivent être des axes servo. <DestinationX> et <DestinationY> définissent la position à atteindre en 2D. <CentreX> et <CentreY> définissent la position relative du centre de l'arc de cercle en 2D.<SensHoraire> est le sens de parcours.

Exemple : MOVC(X=100,Y=100,100,0,1)

Voir aussi : MOVL, MOVE_S

10-16-141- MOVE_S – Etat du mouvement

Syntaxe : **MOVE_S**(<Axe>)

Types acceptés : Bit

Description : Cette fonction indique si l'axe est en mouvement.

Remarques : <Axis> doit être un axe servo. Si l'axe est en mode non asservi (AXIS(AXE)=OFF), l'instruction MOVE_S(Axe)=0. Si l'axe est en mode asservi, MOVE_S(Axe) est égale à 0 si les 4 points suivants sont vrais :

- le mouvement courant est terminé (trajectoire théorique terminée).
- l'erreur de poursuite est à l'intérieur de la fenêtre de positionnement (+/-POSMIN_P(Axe)).
- Le buffer de mouvement est vide.
- Dans le cas d'un axe esclave lié par une fonction CAM ou GEARBOX, le lien doit avoir été coupé.

Si l'un de ces points est faux, l'instruction MOVE_S(Axe) retourne la valeur 1.

Exemple :

```
STTA (X=100)
WAIT NOT MOVE_S(X) ' Attente que l'axe soit arrêté
```

10-16-142- MOVL – Mouvement linéaire

Syntaxe : **MOVL** (<AxeX>=<DestinationX>, <AxeY>=<DestinationY>)

Types acceptés : DestinationX, DestinationY : réel

Description : Cette instruction est utilisée pour faire un mouvement linéaire avec deux axes. Le mouvement est envoyé dans le buffer de mouvement. Si le buffer est plein, la tâche se bloque jusqu'à ce qu'une place se libère dans le buffer. L'exécution de l'instruction provoque le basculement vers la tâche suivante.

Remarques : <AxeX> et <AxeY> doivent être des axes servo. <DestinationX> et <DestinationY> définissent la position à atteindre en 2D.

Exemple :

```
MOVL(X=100,Y=100)
```

Voir aussi : MOVC, MOVE_S

10-16-143- MOVR – Mouvement relatif

Syntaxe : **MOVR**(<Axis>=<Distance> {,<Axis>=<Distance> ... })

Types acceptés : Distance : réel

Description : Déplace l'axe à une position relative. L'exécution de l'instruction provoque le basculement vers la tâche suivante.

Remarques : La tâche attend la fin du mouvement (condition MOVE_S(Axe)=0) avant d'exécuter la prochaine instruction. Les axes utilisent les valeurs courantes de vitesse, d'accélération et de décélération.<Axe> doit être un axe servo

Exemple :
```
MOVR (X=1200.00,Y=-100.00,Z=+550.00,W=Dist!)
```

Voir aussi : MOVA, STTA, STTR, STTI, MOVE_S

10-16-144- MOVS et MOVSP - Mouvement synchronisé

Syntaxe 1 : **MOVS**(<AxeEsclave>,<AxeMaître>,<DistanceMaître>,<DistanceEsclave>,
<DistanceAccélération>, <DistanceDécélération>)

Syntaxe 2 : **MOVSP**(<AxeEsclave>,<AxeMaître>,<DistanceMaître>,<DistanceEsclave>,
<DistanceAccélération>, <DistanceDécélération>,
<AxeDeclenchement>,<Mini>,<Maxi>,<Intérieur>)

Types acceptés : DistanceMaître, DistanceEsclave, DistanceAccélération,
DistanceDécélération, Mini, Maxi : réel
Intérieur : Bit

Limites : Distance Maître, Distance Esclave : de 0 à +/-2^{21} incréments

Description : Ces instructions sont utilisées pour lier un axe esclave à un axe maître pendant une certaine distance de l'axe maître avec des phases d'accélération et de décélération sur l'axe esclave. L'instruction MOVSP intègre un déclenchement sur une fenêtre de position d'un axe. Le mouvement est envoyé dans le buffer de mouvement. Si le buffer est plein, la tâche se bloque jusqu'à ce qu'une place se libère dans le buffer**.**

Remarques : <AxeMaître> peut être un axe servo ou codeur.<AxeEsclave> doit être un axe servo.Les <DistanceAccélélération> et <DistanceDécélération> sont exprimées en distance sur l'axe maître et peuvent être nulles. <Mini> et <Maxi> définissent la fenêtre de déclenchement sur l'axe <AxeDeclenchement> et <Intérieur> indique que la position doit se situer à l'intérieur de la fenètre. <Mini> doit toujours être inférieur à <Maxi>.

Exemple :
```
Coupe au vol
ORDER(Slave)=0    'initialise le n°+1 du prochain mouvement
                  'envoyé dans le buffer
MOVS(Slave,Master,0.8,0.4,0.8,0)      'Accélération mouvement 1
MOVS(Slave,Master,0.2,0.2,0,0)        'Phase synchro mouvement 2
MOVS(Slave,Master,0.8,0.4,0,0.8)      'Décélération mouvement 3
MOVS(Slave,Master,8.2,-1.0,0.5)       'Retour  mouvement 4
WAIT ORDER_S(Slave)>=2  'Attente fin exécution du mouvement 2
OUT(Couteau)=ON         'Activation du couteau
WAIT ORDER_S(Slave)>=3  'Attente fin exécution du mouvement 3
OUT(Couteau)=OFF        'Arrêt du couteau
```

Voir aussi : CAM, CAMBOX, GEARBOX

10-16-145- MOVSC – Mouvement synchronisé déclenché sur entrée Capture

Syntaxe : **MOVSC**(<AxeEsclave>,<AxeMaître>,<DistanceMaître>,<DistanceEsclave>,
<DistanceAccélération>,<DistanceDécélération>,<Configuration>
[,<AxeDeclenchement>,<Fenêtre>,<Mini>,<Maxi>,<Intérieur>])

Types acceptés : DistanceMaître, DistanceEsclave, DistanceAccélération,
DistanceDécélération, Mini, Maxi : réel
Configuration : Octet

- b0 : non utilisé
- b1 : déclenchement sur top Z
- b2 : déclenchement sur entrée n°1 (C1)
- b3 : déclenchement sur entrée n°2 (C2)
- b4 : 1 : front descendant ; 0 : front montant
- b5…b7 : non utilisés

Intérieur : Bit

Description : Ces instructions sont utilisées pour lier un axe esclave à un axe maître pendant une certaine distance de l'axe maître avec des phases d'accélération et de décélération sur l'axe esclave. L'instruction MOVSC intègre un déclenchement sur une entrée de capture. Le mouvement est envoyé dans le buffer de mouvement. Si le buffer est plein, la tâche se bloque jusqu'à ce qu'une place se libère dans le buffer**.** Les paramètres <AxeDéclenchement>, <Fenêtre>, <Mini>, <Maxi> et <Intérieur> sont optionnels.

Remarques : <AxeEsclave> doit être à l'arrêt (MOVE_S(AxeEsclave)=0) avant d'envoyer l'ordre sinon le mouvement peut-être obtenu même si la condition n'est pas valide. <AxeMaître> peut être un axe servo ou codeur. <AxeEsclave> doit être un axe servo. Les <DistanceAccélélération> et <DistanceDécélération> sont exprimées en distance sur l'axe maître et peuvent être nulles. <Mini> et <Maxi> définissent la fenêtre de déclenchement sur l'axe <AxeDeclenchement> et <Intérieur> indique que la position doit se situer à l'intérieur de la fenêtre. <Mini> doit toujours être inférieur à <Maxi>.

Exemple :

```
Coupe au vol
ORDER(Slave)=0    'initialise le n°+1 du prochain mouvement
                  'envoyé dans le buffer
MOVSC(Slave,Master,0.8,0.4,0.8,0)     'Accélération mouvement 1
MOVS(Slave,Master,0.2,0.2,0,0)        'Phase synchro mouvement 2
MOVS(Slave,Master,0.8,0.4,0,0.8)      'Décélération mouvement 3
MOVS(Slave,Master,8.2,-1.0,0.5)       'Retour  mouvement 4
WAIT ORDER_S(Slave)>=2   'Attente fin exécution du mouvement 2
OUT(Couteau)=ON          'Activation du couteau
WAIT ORDER_S(Slave)>=3   'Attente fin exécution du mouvement 3
OUT(Couteau)=OFF         'Arrêt du couteau
```

Voir aussi : CAM, CAMBOX, GEARBOX

10-16-146- 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

10-16-147- 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 de la MCS.

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 de la MCS, 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 de la MCS
```

Voir aussi : INPUT, PRINT, CLOSE

10-16-148- 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

10-16-149- ORDER – Numéro d'ordre du mouvement

Syntaxe 1 : **ORDER**(<Axe>) = <Expression>

Syntaxe 2 : **ORDER**(<Axe>)

Types acceptés : Expression : Entier

Description : Cette instruction fixe le numéro d'ordre+1 du prochain mouvement ou lit le numéro d'ordre du dernier mouvement déposé.

Remarques : Cette instruction peut être utilisée avec la fonction ORDER_S.

Exemple :
```
ORDER(X)=0
MOVS(X,Master,50,100,10,10)
A#=ORDER(X) 'Résultat : A#=1
```

Voir aussi : ORDER_S

10-16-150- ORDER_S – Numéro d'ordre courant

Syntaxe : **ORDER_S**(<Axe>)

Types acceptés : Entier

Description : Cette fonction retourne le numéro du mouvement en cours d'exécution.

Remarques : Cette fonction peut être utilisée pour connaître l'état du mouvement.

Exemple :
```
ORDER(X)=0
MOVS(X,Master,50,100,10,10)
MOVS(X,Master,50,100,10,10)
MOVS(X,Master,50,100,10,10)
IF ORDER_S(X)=2 THEN ...'Le second mouvement est commencé.
```

Voir aussi : ORDER

10-16-151- 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

10-16-152- 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

10-16-153- 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

10-16-154- OUTW - Ecriture d'un bloc de 16 sorties

Syntaxe : **OUTW** (<Sorties>) = <Expression>

Types acceptés : Expression : Entier

Description : Cette fonction envoie des états logiques à un bloc de 16 sorties TOR.

Remarques : <Sorties> doit représenter le nom d'un bloc de 16 sorties.

Exemple :
```
OUTW(Bloc1)=0FFFFh
```

Voir aussi : INP, INPB, INPW, OUT, OUTB

10-16-155- 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 sur le terminal opérateur Dialog 640.

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
```

10-16-156- POS – Position à atteindre

Syntaxe 1 : **POS**(<Axe>) = <Expression>

Syntaxe 2 : **POS**(<Axe>)

Types acceptés : Expression : réel

Description : Cette fonction retourne ou fixe la position à atteindre dans l'unité choisie.

Remarques : Cette fonction est utilisée pour changer la position finale en cours de mouvement. La position peut être modifiée à tout moment.

Exemple :
```
STTA(X=5000)              'Départ de l'axe
WAIT INP(Cellule)=On      'Attente Cellule
POS(X)=POS_S(X)+50.       'Arrêt 50 mm après le capteur
WAIT NOT MOVE_S(X)        'Attente arrêt de l'axe
```

Voir aussi : ACC, DEC, VEL

10-16-157- POS_S – Position réelle

Syntaxe : <Expression> = **POS_S**(<Axe>)

Types acceptés : Expression : réel

Description : Cette fonction retourne la position réelle de l'axe.

Remarques : <Axe> doit être un axe servo .On peut ainsi obtenir l'image en temps réel de la position de l'axe.

Exemple :
```
STTA (X=100)              'Départ de l'axe
REPEAT
    LOCATE 1,1            'Positionnement du curseur
    PRINT POS_S(X);       'Affichage de la position de l'axe
```

```
UNTIL NOT MOVE_S(X)      'Tant que l'axe se déplace
```

Voir aussi : VEL_S

10-16-158- POWERFAIL – Gestion des microcoupures

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

Description : Si POWERFAIL=ON, activation de la gestion des microcoupures.

Remarques : A chaque redémarrage de la MCS, cette instruction est activée automatiquement.

10-16-159- 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 de la MCS.

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 de la MCS. 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

10-16-160- 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

10-16-161- 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 de la MCS 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, la MCS 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, la MCS ne lance pas les tâches et indique une erreur 20 sur son afficheur de status.

Voir aussi : FLASHOK, RAMTOFLASH, FLASHTORAM

10-16-162- 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

10-16-163- 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!)
```

10-16-164- 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!)
```

10-16-165- 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!)
```

10-16-166- REGPOS_S– Position capturée

Syntaxe : <Expression>=**REGPOS_S**(<Axe>)

Types acceptés : Expression : réel

Description : Cette fonction retourne la dernière position capturée sur l'axe.

Remarques : <Axe> doit être un axe servo.

Exemple :
```
CAPTURE(X,Off,On,10,20,On)
WAIT REG_S(X)
XPosition! = REGPOS_S(X)
```

Voir aussi : REG_S, CAPTURE, REGPOS1_S, REGPOS2_S

10-16-167- REGPOS1_S – Position capturée

Syntaxe : <Expression>=**REGPOS1_S**(<Axe>)

Types acceptés : Expression : réel

Description : Cette fonction retourne la dernière position capturée sur l'axe par l'exécution de l'instruction CAPTURE1.(SRV85 seulement)

Remarques : <Axe> doit être un axe servo.

Exemple :
```
CAPTURE1(Y,X,4,On,10,20,On)
WAIT REG1_S(X)
XPosition!= REGPOS1_S(X)
```

Voir aussi : REG_S, CAPTURE, REGPOS_S, REGPOS2_S

10-16-168- REGPOS2_S – Position capturée

Syntaxe : <Expression>=**REGPOS2_S**(<Axe>)

Types acceptés : Expression : réel

Description : Cette fonction retourne la dernière position capturée sur l'axe par l'exécution de l'instruction CAPTURE2.(SRV85 seulement)

Remarques : <Axe> doit être un axe servo.

Exemple :
```
CAPTURE2(X,24,Off,0,0,Off)
WAIT REG2_S(X)
XPosition!= REGPOS2_S(X)
```

Voir aussi : REG_S, CAPTURE, REGPOS_S, REGPOS1_S

10-16-169- REG_S – Etat de la capture

Syntaxe : **REG_S**(<Axe>)

Description : Cette fonction indique si une capture de position a été effectuée.

Remarques : <Axe> doit être un axe servo. La valeur retournée n'est vraie qu'une fois par capture. REG_S est remis automatiquement à 0 sur une opération de lecture et lorsqu'il vaut 1. Sur une relance d'une autre capture et si REG_S vaut 1 alors REG_S est mis à 0.

Exemple :
```
CAPTURE(X,Off,On,10,20,On)
WAIT REG_S(X)            'REG_S redevient faux automatiquement
XPosition! = REGPOS_S(X)
```

Voir aussi : REGPOS_S, CAPTURE, REG1_S, REG2_S

10-16-170- REG1_S – Etat de la capture

Syntaxe : **REG1_S**(<Axe>)

Description : Cette fonction indique si une capture de position a été effectuée.(exécution de l'instruction CAPTURE1 avec SRV85 seulement)

Remarques : <Axe> doit être un axe servo. La valeur retournée n'est vraie qu'une fois par capture. REG1_S est remis automatiquement à 0 sur une opération de lecture et lorsqu'il vaut 1. Sur une relance d'une autre capture et si REG1_S vaut 1 alors REG1_S est mis à 0.

Exemple :
```
CAPTURE1(Y,X,4,On,10,20,On)
WAIT REG1_S(X)           'REG_S redevient faux automatiquement
XPosition!= REGPOS1_S(X)
```

Voir aussi : REGPOS_S, CAPTURE, REG_S, REG2_S

10-16-171- REG2_S – Etat de la capture

Syntaxe : **REG2_S**(<Axe>)

Description : Cette fonction indique si une capture de position a été effectuée.(exécution de l'instruction CAPTURE2 avec SRV85 seulement)

Remarques : <Axe> doit être un axe servo. La valeur retournée n'est vraie qu'une fois par capture. REG2_S est remis automatiquement à 0 sur une opération de lecture et lorsqu'il vaut 1. Sur une relance d'une autre capture et si REG2_S vaut 1 alors REG2_S est mis à 0.

Exemple :
```
CAPTURE2(X,24,Off,0,0,Off)
WAIT REG2_S(X)          'REG_S redevient faux automatiquement
XPosition!= REGPOS2_S(X)
```

Voir aussi : REGPOS_S, CAPTURE, REG_S, REG1_S

10-16-172- 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

10-16-173- 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.

10-16-174- 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$

10-16-175- 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$

10-16-176- 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

10-16-177- SECURITY – Définit les actions de sécurités

Syntaxe : **SECURITY**(<OuvreBoucle>,<OuvreWatchDog>)

Description : Cette instruction est utilisée pour définir comment le système doit réagir si une erreur de poursuite sur un axe est détectée. <OuvreBoucle> définit si tous les axes doivent se débrayer. <OuvreWatchDog> indique si le watch dog doit s'ouvrir. Les valeurs par défaut à la mise sous tension sont SECURITY(On,On).

Remarques : Si l'instruction SECURITY est utilisée, le niveau de sécurité peut être diminué suivant l'écriture du programme. Il est conseillé de ne pas utiliser cette instruction.

Exemple :
```
SECURITY(ON,OFF)  ' Chien de garde ne s'ouvre pas sur un
                  ' défaut de poursuite
```

10-16-178- 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

10-16-179- SENSOR_S – Etat du capteur

Syntaxe : **SENSOR_S** (<Axe>)

Description : Cette fonction donne l'état de l'entrée home/capture de la carte d'axe.

Exemple : `IF SENSOR_S(X) THEN ...`

Voir aussi : SENSOR1_S, SENSOR2_S

10-16-180- SENSOR1_S – Etat du capteur C1 (SRV85)

Syntaxe : **SENSOR1_S** (<Axe>)

Description : Cette fonction donne l'état de l'entrée home/capture C1 de la carte servo SRV85.

Exemple : `IF SENSOR1_S(X) THEN ...`

Voir aussi : SENSOR_S, SENSOR2_S

10-16-181- SENSOR2_S – Etat du capteur C2 (SRV85)

Syntaxe : **SENSOR2_S** (<Axe>)

Description : Cette fonction donne l'état de l'entrée home/capture C2 de la carte servo SRV85.

Exemple : `IF SENSOR2_S(X) THEN ...`

Voir aussi : SENSOR_S, SENSOR1_S

10-16-182- 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

10-16-183- 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
```

10-16-184- 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
```

10-16-185- 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

10-16-186- SETUPCOUNTER – Configure le compteur

Syntaxe : **SETUPCOUNTER**(<Axe>,<Entrée>,<Inversion>,<Filtre>)

Types acceptés : <Entrée> : Octet

<Inversion>, <Filtre> : bit

Description : Cette instruction permet de configurer le compteur

Remarques : <Axe> : Nom de la carte servo

<Entrée> : Numéro de l'entrée (1 pour l'entrée C1, 2 pour l'entrée C2)

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

<Filtre> : Validation du filtre : 0 pour sans filtrage, 1 pour filtre de 2 ms.

Voir aussi : COUNTER_S , CLEARCOUNTER

10-16-187- 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
```

10-16-188- 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

10-16-189- 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

10-16-190- 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

10-16-191- SQR – Racine carrée

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

Types acceptés : Expression : réel

Description : Cette fonction retourne la racine carrée de <Expression>.

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

10-16-192- SSTOP – Arrêt d'un axe

Syntaxe : **SSTOP**(<Axe>[,<Axe>]...)

Description : Cette fonction stoppe <Axe> avec la décélération courante. La fonction n'est pas bloquante pour la tâche.

Remarques : Si <Axe> est un axe maître lié avec la fonction CAM, GEARBOX ou MOVS, alors <Axe> et son esclave sont stoppés mais le lien n'est pas rompu.

Si <Axe> est un axe esclave lié avec la fonction CAM, GEARBOX ou MOVS, alors <Axe> s'arrête et il n'est plus lié avec le maître.

L'instruction SSTOP vide le buffer de mouvement et stoppe l'axe en utilisant la décélération courante. Cette instruction n'est pas bloquante et n'attend pas que MOVE_S(Axe) soit égal à 0.

Exemple : SSTOP(Master,Slave)

Voir aussi : STTA, STTR, STTI, CAM, GEARBOX, MOVS

10-16-193- STARTCAM – Exécute une came

Syntaxe : **STARTCAM**(<Esclave>,<Maître>,<N°came>)

Limites : <N°came> : de 1 à 5

Types acceptés : <N°came> : Octet

Description : Cette instruction lance l'exécution d'une came

Remarques : <Esclave> : Nom de l'axe esclave où est effectuée la came (carte servo : SRV15, SRV85...)

<Maître> : Nom de l'axe maître (carte servo ou codeur : SCD22, SRV15, SRV85...)

<N°came> : numéro de la came de la carte servo <Esclave>.

Voir aussi : LOADCAMEX, CAM_S

10-16-194- STARTCAMBOX – Lance une boîte à cames

Syntaxe : **STARTCAMBOX**(<Num boîte>)

Description : Cette instruction lance une boîte à contact précédemment définie.

Remarques : Si la boîte à came n'est pas défine, la fonction n'a pas d'effet.<Num boîte> est le numéro utilisé dans la fonction CAMBOX.

Exemple : STARTCAMBOX(1)

Voir aussi : CAMBOX

10-16-195- STARTS– Lance un mouvement synchronisé

Syntaxe : **STARTS**(<Axis>,[<Axis>]...)

Description : Cette instruction lance un mouvement synchronisé précédemment chargé dans le buffer de mouvement avec LOADS.

Remarques : LOADS et STARTS doivent être utilisés à la place de MOVS pour commencer des mouvements simultanément.

Exemple :
```
LOADS(Slave1,Master,90,180,10,10) 'chargement du mouvement 1
                                  'dans le buffer de mouvement
LOADS(Slave1,Master,90,180,10,10) 'chargement du mouvement 2
                                  'dans le buffer de mouvement
STARTS(Slave1,Slave2)             'exécution du mouvement 1
```

Voir aussi : MOVS, STARTS

10-16-196- 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
```

10-16-197- STOP - Arrêt d'un axe

Syntaxe : **STOP**(<Axe>[,<Axe>]...)

Description : Cette fonction stoppe <Axe> avec la décélération courante. La fonction est bloquante tant que l'axe n'est pas arrêté.

Remarques : Si <Axe> est un axe maître lié avec la fonction CAM, GEARBOX ou MOVS, alors <Axe> et son esclave sont stoppés mais le lien n'est pas rompu.

Si <Axe> est un axe esclave lié avec la fonction CAM, GEARBOX ou MOVS, alors <Axe> s'arrête et il n'est plus lié avec le maître.

L'instruction STOP vide le buffer de mouvement et stoppe l'axe en utilisant la décélération courante. Cette instruction est bloquante tant que MOVE_S(Axe) est différent de 0.

Exemple :
```
STOP(Master)
```

Voir aussi : STTA, STTR, STTI, CAM, GEARBOX, MOVS

10-16-198- STOPCAMBOX – Arrête une boîte à came

Syntaxe : **STOPCAMBOX**(<Num boîte>)

Description : Cette instruction arrête une boîte à contact précédemment définie.

Remarques : <Num boîte> est le numéro utilisé dans la fonction CAMBOX. Cette fonction ne détruit pas la boîte.

Exemple : STOPCAMBOX(1)

Voir aussi : CAMBOX, CAMBOXSEG, STARTCAMBOX

10-16-199- STOPCORRECTION– Arrêt de la fonction de compensation

Syntaxe : **STOPCORRECTION** (<Esclave>)

Description : Cette fonction permet d'arrêter la fonction de compensation déclarée en mode permanent

Remarques : <Esclave> doit être un axe servo SRV85

Voir aussi : CORRECTION, ICORRECTION

10-16-200- STOPI – Arrêt d'une interpolation

Syntaxe : **STOPI**(<AxeX>,<AxeY>)

Description : Cette fonction stoppe <AxeX> et <AxeY> avec la décélération d'interpolation courante. La fonction est bloquante tant que l'axe n'est pas arrêté et provoque le basculement vers la tâche suivante.

Exemple : STOPI(X,Y)

Voir aussi : STTA, STTR, STTI, CAM, GEARBOX, MOVS

10-16-201- 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

10-16-202- STR$ - Conversion en chaîne de caractères

Syntaxe : **STR$**(<Expression>)

Types acceptés : Expression : Chaîne de caractères

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.

Exemple :
```
a%=10
b$=STR$(a%) 'Résultat b$=" 10"
```

Voir aussi : VAL

10-16-203- STTA – Lance un mouvement absolu

Syntaxe : **STTA**(<Axe>=<Distance> {,<Axe>=<Distance>})

Types acceptés : Distance : réel

Description : Lance un mouvement à une position absolue.

Remarques : Le système n'attend pas la fin du mouvement (MOVE_S(Axe)=0) et exécute la prochaine instruction. Les axes utilisent la vitesse, l'accélération et la décélération courante.<Axe> doit être un axe servo.

Exemple :
```
STTA (X=1200.00,Y=-100.00)
WAIT(NOT MOVE_S(X)) AND (NOT MOVE_S(Y))
```

Voir aussi : MOVA, MOVR, STTR, STTI

10-16-204- STTI – Lance un mouvement infini

Syntaxe : **STTI**(<Axe>=[+|-] {,<Axe> =[+|-], ... })

Description : Lance un mouvement infini.

Remarques : Le système n'attend pas la fin du mouvement et exécute la prochaine instruction. Vous devez utiliser l'instruction STOP ou SSTOP pour arrêter le mouvement.<Axe> doit être un axe servo. Les axes utilisent la vitesse, l'accélération et la décélération courante.

Exemple :
```
STTI (X=+, Y=-)   'commence un mouvement infini sur l'axe X
                  'dans la direction positive et l'axe Y
                  'dans la direction négative.
```

Voir aussi : MOVA, MOVR, STTA, STTR

10-16-205- STTR – Lance un mouvement relatif

Syntaxe : **STTR**(<Axe>=<Distance> {,<Axe>=<Distance> ... })

Types acceptés : Distance : réel

Description : Lance un mouvement relatif.

Remarques : Le système n'attend pas la fin d'un mouvement (MOVE_S(Axe)=0) avant d'exécuter la prochaine instruction. Les axes utilisent la vitesse, l'accélération et la décélération courante.<Axe> doit être un axe servo.

Exemple :
```
STTR (X=1200.00,Y=-100.00,Z=+550.00,W=Dist!)
```

Voir aussi : MOVA, MOVR, STTA, STTI

10-16-206- 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

10-16-207- 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

10-16-208- 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

10-16-209- 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 la MCS 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

10-16-210- 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$, TIME, DATE$

10-16-211- 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$

10-16-212- TRAJ - Trajectoire

Syntaxe : **TRAJ**(<Paramètre>=<Valeur> {,<Paramètre>=<Valeur> ... })

Types acceptés : Valeur : réel

Description : Cette instruction effectue une trajectoire complexe. L'exécution de cette tâche provoque le basculement vers la tâche suivante.

Remarques : Le système attend la fin du mouvement (MOVE_S(Axe)=0) avant d'exécuter la prochaine instruction. <Paramètre> est POS(<Axe>) pour une position absolue, VEL(<Axe>) pour une vitesse , ACC(<Axe>) pour une accélération et DEC(<Axe>) pour une décélération. Les axes utilisent la vitesse, l'accélération et la décélération courante si elles ne sont pas définies dans l'instruction TRAJ. <Axe> doit être un axe servo.

Exemple :
```
MERGE(X)=On                              'Passage en petite vitesse
TRAJ (POS(X)=1000.00, VEL(X)=Vrapide)'à la position 1000,
TRAJ (POS(X)=1500.00, VEL(X)=Vlente)  ' sans arrêt de l'axe
MERGE(X)=Off
```

Voir aussi : MOVA, MOVR, STTA, STTI

10-16-213- 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
```

10-16-214- 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$

10-16-215- 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$

10-16-216- VEL - Vitesse

Syntaxe 1 : VEL(<Axe>) = <Expression>

Syntaxe 2 : VEL(<AxeX>, <AxeY>) = <Expression>

Unité : Expression : unités utilisateurs par seconde (Ex : mm/s, tr/s, degré/s).

Types acceptés : Expression : réel

Description : Cette valeur spécifie la vitesse courante en unités par seconde.

Remarques : <Expression> doit être une expression réelle valide. Cette valeur de vitesse peut être modifiée à tout moment. La seconde forme est utilisée pour l'interpolation. Il est déconseillé de rafraîchir la vitesse plus de 10 fois par seconde sur un axe utilisant une accélération sinus : un comportement anormal pourrait apparaître lors des phases d'accélération ou de décélération.

Exemple :
```
VEL(X)=2000
```

Voir aussi : ACC, DEC, POS

10-16-217- VEL% - Fixe la vitesse en pourcentage

Syntaxe : **VEL%**(<Axe>) = <Expression>

Limite : Expression : de 0 à 100

Types acceptés : Expression : réel

Description : Cette fonction ajuste la vitesse courante en pourcentage du paramètre de vitesse VEL_P.

Remarques : La valeur de VEL_P peut être entrée dans l'écran profil de vitesse lors de la configuration de la carte. Il est déconseillé de rafraîchir la vitesse plus de 10 fois par seconde sur un axe utilisant une accélération sinus : un comportement anormal pourrait apparaître lors des phases d'accélération ou de décélération.

Exemple :
```
VEL_P(X)=2000
VEL%(X)=20 'La vitesse courante est de 400 unités / s
```

Voir aussi : ACC%, DEC%

10-16-218- VEL_S - Vitesse

Syntaxe : **VEL_S**(<Axe>)

Description : Cette fonction retourne la vitesse courante.

Remarques: <Axe> doit être un axe servo. La valeur renvoyée est l'image de la vitesse réelle pour une carte SRV85, l'image de la vitesse calculée pour toutes les autres cartes servo.

Exemple :
```
STTA (X=100)
WHILE NOT MOVE_S(X) DO PRINT #1,VEL_S(X)
```

Voir aussi : POS_S

10-16-219- 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

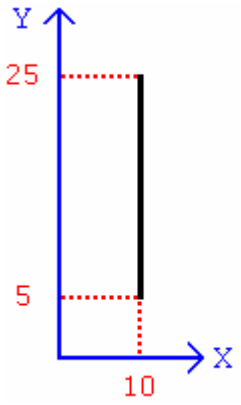
Types acceptés : X1, Y1, Y2 : octet.

Couleur : Bit

Description : Trace une ligne horizontale avec le point de départ en X1, Y1 et d'arrivée en X1, Y2 sur le terminal opérateur Dialog 640.

Remarques : Couleur permet de modifier la couleur du trait : noir (0) ou blanc (1)

Exemple : VLINE(10,5,25,0)



10-16-220- 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

10-16-221- 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 ...
...
```

10-16-222- 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

10-16-223- 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
```

10-16-224- 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

10-16-225- 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

10-16-226- ZERO_S – Etat du zéro codeur

Syntaxe : **ZERO_S** (<Axe>)

Description : Cette fonction restitue l'état du zéro codeur de la carte d'axe.

Exemple :
```
IF  ZERO_S(X) THEN STOP(X)
```

11- CANopen

11-1- Définition

11-1-1- 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.

11-1-2- 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.

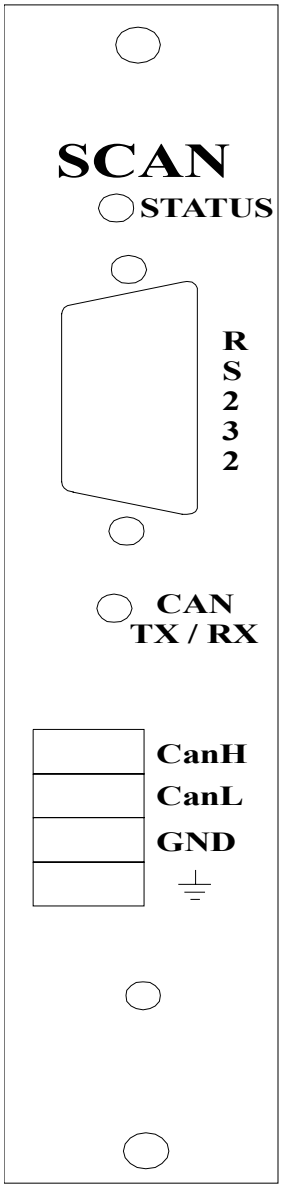
11-2- Carte CANopen pour MCS32 EX : SCAN

11-2-1- Caractéristiques

- Un serveur SDO par défaut pour le paramétrage de la carte à distance par un superviseur.
- 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 une autre MCS.
- 8 PDO en réception pour recevoir les entrées des modules I/O ou recevoir les événements à une autre MCS.
- 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.

11-2-2- Raccordement

↳ Cavalier de validation des résistances (120Ω) de terminaison



11-2-3- Dictionnaire

Le dictionnaire contient les différents paramètres et variables de la carte. Ils sont directement accessibles par la MCS 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.

Index	Sub-idx	Nom	Type	Attr.	Défaut	Description
1000	0	Device Type	32 bits non signé	ro	403	Type d'appareil
1001	0	Error register	32 bits non signé	ro	0	Registre d'erreur interne
1002	0	Manufacturer Status Register	32 bits non signé	ro	0	Registre d'état spécifique au constructeur
1003	0	Predefined error field	8 bits non signé	ro	1	Nombre d'erreurs apparues
	1	Actual error	32 bits non signé	ro	0	Dernière erreur apparue
1004	0	Number of PDO's supported	32 bits non signé	ro	00080008h	Nombre de PDO supporté
	1	Number of synchronous PDO	32 bits non signé	ro	0	Nombre de PDO synchrone supporté
	2	Number of asynchronous PDO	32 bits non signé	ro	00080008h	Nombre de PDO asynchrone supporté
100B	0	NODE-ID	32 bits non signé	ro	aucune	
100F	0	Number of SDO's supported	32 bits non signé	ro	00010001h	Nombre de SDO supporté
1200	0	Number of elements	8 bits non signé	ro	2	Paramètre du 1er SDO serveur
	1	SDO Receive COB-ID	32 bits non signé	ro	600h+node-id	COB-ID de réception du 1er SDO serveur
	2	SDO Transmit COB-ID	32 bits non signé	ro	580h+node-id	COB-ID d'émission du 1er SDO serveur
	3	NODE-ID of the SDO client	8 bits non signé	rw	aucune	NODE-ID du SDO client
1280	0	Number of elements	8 bits non signé	ro	2	Paramètre du 1er SDO client
	1	SDO receive COB-ID	32 bits non signé	ro	aucune	COB-ID de réception du 1er SDO client
	2	SDO transmit COB-ID	32 bits non signé	ro	aucune	COB-ID d'émission du 1er SDO client
	3	NODE-ID of the SDO server	8 bits non signé	rw	aucune	NODE-ID du SDO serveur
1400	0	Number of elements	8 bits non signé	rw	2	Paramètre de réception du 1er PDO
	1	COB-ID	32 bits non signé	rw	aucune	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de la réception
...	...	...	...	...	...	...
1407	...	...	...	...	...	...
1800	0	Number of elements	8 bits non signé	rw	2	Paramètre d'émission du 1er PDO
	1	COB-ID	32 bits non signé	rw	aucune	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de l'émission
...	...	...	...	...	...	...
1807	...	...	...	...	...	...

Index	Sub-idx	Nom	Type	Attr.	Défaut
7180	de 1 à FEh	Lecture de variables 32 bits	32 bits signé	ro	aucune
7200	de 1 à FEh	Lecture de variables 8 bits	8 bits non signé	ro	aucune
7280	de 1 à FEh	Lecture de variables 16 bits	16 bits non signé	ro	aucune
8180	de 1 à FEh	Ecriture de variables 32 bits	32 bits signé	wo	aucune
8200	de 1 à FEh	Ecriture de variables 8 bits	8 bits non signé	wo	aucune
8280	de 1 à FEh	Ecriture de variables 16 bits	16 bits non signé	wo	aucune

11-3- Liste des instructions

11-3-1- 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

11-3-2- 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.

11-3-3- 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.

11-3-4- 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.

11-3-5- 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.

11-3-6- 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 la MCS. 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.

11-3-7- 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 de la carte CANopen de la MCS.

Remarques : <Carte> doit être une carte CANopen. <Index> et <Sub-Index> doivent se référer à un élément du dictionnaire.

11-3-8- 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 de la carte CANopen de la MCS.

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.

11-3-9- 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.

11-3-10- 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.

11-3-11- 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.

11-3-12- 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.

11-3-13- 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.

11-4- Exemples

11-4-1- Liaison CANopen entre deux MCS

Le paramétrage de la communication entre deux MCS consiste à attribuer un NodeID à chaque MCS. Une communication par SDO est alors possible après paramétrage de ceux-ci. Il est également possible d'échanger des événements par PDO.

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 de la MCS 1

```
'Démarrage de la carte à 500KBits/s sur le nœud 1
StartCan(Can1,1,5)
'COBID ClientSDO Rx Mcs1= COBID ServerSDO Tx Mcs2
CanSetup&(Can1,1280h,1,582h)
'COBID ClientSDO Tx Mcs1= COBID ServerSDO Rx Mcs2
CanSetup&(Can1,1280h,2,602h)
'COBID TxPDO1 = COBID RxPDO2
CanSetup&(Can1,1800h,1,202h)
```

↳ Initialisation de la MCS 2

```
'Démarrage de la carte à 500KBits/s sur le nœud 2
StartCan(Can2,2,5)
'COBID ClientSDO Rx Mcs2= COBID ServerSDO Tx Mcs1
CanSetup&(Can2,1280h,1,581h)
'COBID ClientSDO Tx Mcs2= COBID ServerSDO Rx Mcs1
CanSetup&(Can2,1280h,2,601h)
'COBID TxPDO2 = COBID RxPDO1
CanSetup&(Can2,1800h,1,201h)
```

Cette initialisation terminée les MCS peuvent échanger des variables et des événements. Dans cet exemple, la MCS 2 envoie des ordres de positionnement à l'axe X de la MCS 1. La MCS 1 reçoit les ordres à exécuter par un PDO et signale la fin de l'ordre en envoyant un PDO. La position à atteindre est lue dans la variable 5 du tableau "Lecture de variable 32 bits" de la MCS 2. La MCS 1 met également à disposition la position de son axe X dans la variable 1 de son tableau "Ecriture de variable 32 bits".

```
Wait PDOEvent(Can1,1)              'Attente du PDO signalant le message
O$=PDO(Can1,1)                     'Lecture du PDO
Ordre#=Asc(Left$(O$,1))            'Décodage de l'ordre
Pos&=CanRemote&(Can1,7180h,5)      'Lecture de la position
If Ordre#=1 Then Stta(X=Pos&)             'Exécution en absolu
If Ordre#=2 Then Sttr(X=Pos&)             'Exécution en relatif
...
```

```
Repeat
  P&=RealToLong(Pos_S(X))          'Lecture de la position
  CanLocal&(Can1,1,P&)             'Mise à disposition de la position
Until Move_S(X)=0
O$=Chr(0)                          'Réponse
PDO(Can1,1,O$)                     'Acquittement de l'ordre
```

La MCS 2 envoie les ordres, lit la position de l'axe X de la MCS 1 dans la variable 1 du tableau "Lecture de variables 32 bits" de celle-ci et envoi les positions dans la variable 5 de son tableau "Ecriture de variables 32 bits".

```
CanLocal&(Can2,5,10.25)      'Ecriture de la position
O$=Chr(1)                    'Envoi d'un ordre de mouvement en absolu
PDO(Can2,1,O$)               'Envoi du PDO
Repeat
  P&=CanRemote&(Can2,7180h,1)       'Lecture de la position
  ...
Until PDOEvent(Can2,1)       'Jusqu'à la fin du mouvement
```

11-4-2- Liaison CANopen entre une MCS et un module d'entrées/sorties

Le paramétrage de la communication entre une MCS et un module I/O consiste à attribuer un NodeID a chacun. Le NodeID d'un module I/O est généralement paramétré par des interrupteurs. 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 de la MCS

```
'Démarrage de la carte à 500KBits/s sur le nœud 1
StartCan(Can1,1,5)
'COBID ClientSDO Rx Mcs= COBID ServerSDO Tx I/O
CanSetup&(Can1,1280h,1,582h)
'COBID ClientSDO Tx Mcs= COBID ServerSDO Rx I/O
CanSetup&(Can1,1280h,2,602h)
'COBID TxPDO MCS = COBID RxPDO I/O
CanSetup&(Can1,1800h,1,202h)
'COBID RxPDO MCS = 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 server 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
```

12- ANNEXES

12-1- 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.

12-2- Messages afficheur STATUS 7 segments

L'afficheur status situé en face avant de la MCS peut-être traité à l'intérieur d'une tâche pour indiquer l'évolution de la tâche, il peut indiquer l'état de la MCS et les messages d'erreurs. Par défaut l'afficheur indique 'G'comme go. Si la MCS est connectée avec le logiciel MCB, elle indique 'S' comme système. Les messages d'erreurs sont prioritaires et sont de la forme :

ERREUR N°1 à ERREUR N°10 :

Les erreurs de 1 à 10 indiquent qu'une carte est mal déclarée ou qu'une carte déclarée dans la configuration est absente ou à été remplacée par un autre type. Le numéro suivant le E indique le slot. Par exemple E6 indique que la carte dans le slot 6 est mal déclarée. Le système ne charge pas les paramètres et ne lance pas les tâches utilisateur.

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 de la MCS 32 si un paramètre de la configuration est erroné. Les paramètres doivent être corrigés et transférés avant de pouvoir redémarrer la MCS. 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 la MCS.

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 et tous les axes servo passeront en boucle ouverte.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z