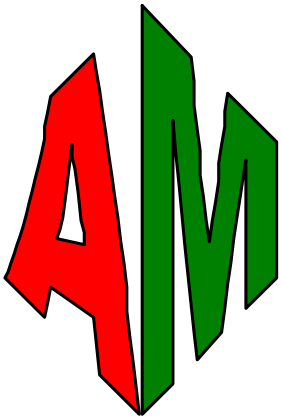
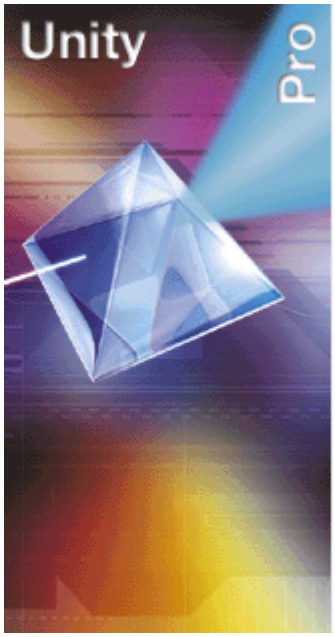


NOTICE DE PROGRAMMATION D'AUTOMATES SCHNEIDER MODICON, PREMIUM, QUANTUM

SOUS UNITY-PRO

Ecrit par Alain Malvoisin

PROGRAMMATION

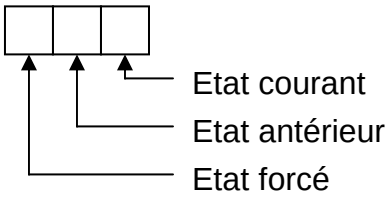


SOMMAIRE

1. LES VARIABLES

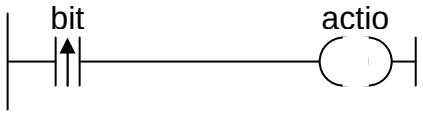
1.1. Topologie

1.1.1. Les bits de type EBOOL



Ce sont des bits que l'on peut mettre à 1 ou à 0, que l'on peut tester.

Ces bits on l'avantage de pouvoir tester leurs **fronts montants ou descendants**, grâce à leur état antérieur
L'état antérieur est **mise à jour à chaque fois qu'il est écrit**, c'est-à-dire que l'état antérieur prend la valeur de l'état courant, puis l'état courant est mis à jour



Il y a des instructions spécifiques de test de fronts montants ou descendants

Ces bits on l'avantage de pouvoir être **forcé**, grâce au bit "Etat forcé", ils peuvent être forcés (ou figés) à l'aide de la console de programmation et cet état reste même lors que la console est débranchée.

Attention 1: forcer un bit peut gêner la détection des fronts, c'est-à-dire que le front peut être détecté en permanence, par exemple vous avez une instruction qui détecte le front montant d'un bit, ce bit vos 0 et vous le forcé à 1, sont état courant prend la valeur 1 mais sont état antérieur qui vaut 0 n'est plus changé, puisqu'il est "forcé". On détecte le front à chaque passage.

Attention 2: L'état antérieur est mis à jour à chaque fois qu'une instruction de mise à jour est exécutée, pour les instructions d'assignation il n'y à pas de problème, mais il reste le problème des instructions de mémorisation (set ou reset) en langage **LADDER**, qui sont systématiquement exécutés, même lorsque la condition amont est fausse.

Même si cond est faux, l'instruction s'exécute.
C'est-à-dire : la condition est fausse donc ne fait pas le "set", mais fais quand même la mise à jour de l'état antérieur

1.1.2. Les mots de type INT, UINT, WORD

Ce sont des mots de 16 bits que l'on peut écrire ou lire

1.1.3. Les bits de mots de type BOOL

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

Ceux sont des bits extraits de mots, il y a donc 16 bits par mot

1.1.4. Les doubles mots de type DINT, UDINT, DWORD, REAL, DATE, TIME, TIME OF DAY,

Ceux sont deux mots consécutifs, chacun faisant 16 bits, un double mot fait donc 32 bits.

1.1.5. Les structures STRUCT

Les structures peuvent contenir l'ensemble des variables vues ci-dessus, la taille des structures dépend des éléments qui la constituent

1.1.6. Les tableaux ARRAY[n..m] OF xxx

Ceux sont des ensembles allant de n à m de même type

1.2. Types des variables

TYPE	Définition	Représentation	Example
BOOL	Bit simple	True/false	TRUE
EBOOL	Bit avancé	True/false	FALSE
BYTE	Octet		201
INT	Entier signé sur 16 bits		-154
UINT	Entier non signé sur 16 bits		64200
WORD	Mot de 16 bit		
DINT	Entier signé sur 32 bits		-1478230
UDINT	Entier non signé sur 32 bits		5412410
DWORD	Double mot sur 32 bits		
REAL	Réel sur 32 bits		154e10
DT	Date and time sur 64 bits	DT#	DT#2009-05-09-10:54:12
DATE	Date sur 32 bits	D#	D#2009-05-09
TIME	Temps 32 bits	T#	T#10D_3H_42M_5S_290MS
TOD	Heure du jour sur 32 bits	TOD#	TOD#10:54:10
STRING	Chaîne de caractères	'chaîne'	

1.3. Représentation des nombres

Format	Représentation	Exemples
Décimal		214, -58
Hexadécimal	16#	16#5FA2
Binaire	2#	2#1001101011
Octal	8#	8#571
Réel		12.21 -458e-24
Durée de tempo	T#D_H_M_S_MS	T#2H_50M_35S_12MS T#5S
Date	D#aaa-mm-jj	D#1998-12-28
Heure	TOD#hh:mm:ss	TOD#10:32:50
Date et Heure	DT#aaaa-mm-jj-hh:mm:ss	DT#1998-10-25-07:15:00
Chaîne car.	‘ ‘	‘A’, ‘COUCOU’, ‘$N'indice’

Pour les chaînes de caractères
Le signe $ (dollar) est un caractère spécial, suivi de certaines lettres il indique :
$L ou $l, aller à la ligne suivante (line feed),
$N ou $n, aller au début de la ligne suivante (new line),
$P ou $p, aller à la page suivante (go to next page),
$R ou $r, retour chariot (carriage return),
$T ou $t, tabulation (Tab),
$$, Représente le caractère $ dans une chaîne,
$’, Représente le caractère quotte dans une chaîne.

1.4. Adresses

Les différentes variables ci-dessus ont une adresse, qu'il n'est pas nécessaire de connaître ni de définir.
On peut donc avoir besoin d'un entier signé de 16 bits, qu'on appellera par exemple "toto" on définira son type "INT" et on pourra lui affecter un commentaire, par exemple "toto est un mot entier signé de 16 bits"

1.4.1. Les variables internes

Ceux sont des variables utilisateurs, que l'on peu lire ou écrire

%Mtn	t:	type topologique	rien ou X bits, W mot, D double mots, B octet
	n:	numéro	
Exemples :	%M12		bit numéro 12 de type EBOOL
	%MW54		Mot numéro 54
	%MD157		Double mot numéro 157
	%Mx12		bit numéro 12 de type EBOOL

1.4.2. Les constantes

Ceux sont des valeurs qui sont définis et qu'on ne pourra pas faire varier lors de la programmation, *il n'y a pas de bits constants, mais il y a les valeurs TRUE ou FALSE*

%Ktn	t	type topologique:	W mot, D double mots, B octet
	n:	numéro	
Exemples:	%KW63		Mot numéro 63
	%KD70		Double mot numéro 70

1.4.3. Les variables systèmes

Ceux sont des variables qui nous donnent des informations sur l'état du système, que l'on peut lire ou écrire et dans ce cas elles permettent d'agir sur le système.

%Stn	t:	type topologique	rien ou X bits, W mot, D double mots, B octet
	n:numéro		
Exemples :	%S1		Bit système numéro 1
	%SW50		Mot système numéro 50
	%SD20		Double mot système numéro 20

1.4.4. Les entrées de l'automate

Ce sont des variables qui sont échangé implicitement avec les périphériques d'entrées, cet échange à lieu au départ de la tâche à laquelle les cartes sont affectées, par défaut la tache maître MAST, sinon la FAST ou l'AUX. Cet échange se défini lors de la configuration de la carte

%Itr,m,v.d	t:	type topologique	rien ou X bits, W mot, D double mots
	r:	n°du rack	
	m:	n°du module	
	v:	n°de la voie	
	d:	n°de rang (si =0 on peu s'en passer)	

Exemples	%I0.2.45	Voie 45 du module 2 dans le rack 0, c'est un bit d'entrée
	%IW0.4.2	Voie 2 du module 4 dans le rack 0
	%I0.4.2.3	Bit 3 de la voie 2 du module 4 dans le rack 0

1.4.5. Les sorties de l'automate

Ce sont des variables qui sont échangé implicitement avec les périphériques de sorties, cet échange à lieu à la fin de la tâche à laquelle les cartes sont affectées, par défaut la tâche maître MAST, sinon tâche FAST ou AUX. Cet échange se défini lors de la configuration de la carte

%Qtr,m,v.d	t:	type topologique	rien ou X bits, W mot, D double mots
	r:	n°du rack	
	m:	n°du module	
	v:	n°de la voie	
	d:	n°de rang (si =0 on peu s'en passer)	

Exemples	%QW0.3.2	Voie 2 du module 3 dans le rack 0, c'est un mot de sortie
	%QW0.2.45.3	mot 3 de la voie 45 du module 2 dans le rack 0, c'est un mot

1.4.6. Les entrées déporté de l'automate (bus FIPIO, AsI, Profibus, Ethernet)

Ce sont des variables qui sont échangé implicitement avec les périphériques déportés d'entrées, cet échange à lieu au départ de la tâche à laquelle les cartes sont affectées, par défaut la tache maître MAST, sinon la FAST ou l'AUX. Cet échange se défini lors de la configuration de la carte sur le périphérique déporté

%It\b.e\r.m.v.d	t:	type topologique	rien ou X bits, W mot, D double mots
	b	Identificateur de bus	
	e	point de connexion	
	r:	n° du rack	
	m:	n° du module	
	v:	n° de la voie	
	d	rang (si =0 on peu s'en passer)	

Exemples	%I\2.4\0.2.45	Voie 45 du module 2 dans le rack 0, c'est un bit
	%IW\2.4\0.2.45.2	Mot 2 de la Voie 45 du module 2 dans le rack 0, c'est un bit

1.4.7. Les sortie déporté de l'automate (bus FIPIO, AsI, Profibus, Ethernet)

Ce sont des variables qui sont échangé implicitement avec les périphériques déportés de sorties, cet échange à lieu à la fin de la tâche à laquelle les cartes sont affectées, par défaut la tâche maître MAST, sinon tâche FAST ou AUX. Cet échange se défini lors de la configuration de la carte sur le périphérique déporté

%Qt\b.e\r.m.v.d	t:	type topologique	rien ou X bits, W mot, D double mots
	b	Identificateur de bus	
	e	point de connexion	
	r:	n° du rack	
	m:	n° du module	
	v:	n° de la voie	
	d	rang (si =0 on peu s'en passer)	

Exemples	%Q\2.4\0.2.45	Voie 45 du module 2 dans le rack 0, c'est un bit
	%QW\2.4\0.2.45.2	Mot 2 de la Voie 45 du module 2 dans le rack 0, c'est un bit

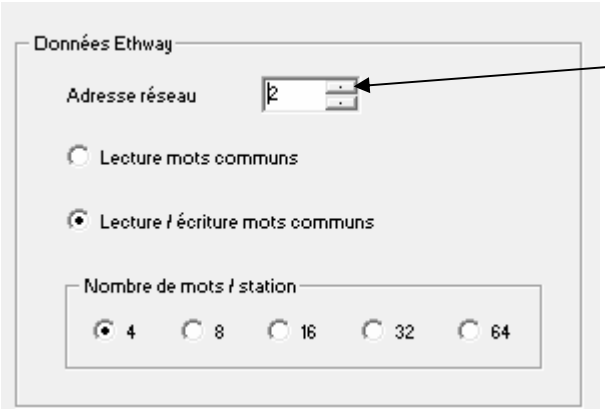
Pour les objets d'Entrée/Sortie de l'automate, le bit ou le mot de rang "0" correspond à l'état de la voie TOR ou Analogique ou de comptage, le numéro de rang "0" n'est pas obligatoire (on n'écrit pas .0)

Exemple :	**%I0.2.3**	**Correspond à l'entrée 3 du module 2 dans le rack 0**
	%IW0.3.2	**Correspond à l'entrée 2 du module 3 dans le rack 0**
	%Q0.4.61	**Correspond à la sortie 61 du module 4 dans le rack 0**

1.4.8. Les mots communs (communication par réseaux FIPWAY ou ETHWAY)

%NWr,s,m	r:	n° du réseau
	s:	n° de station
	m:	n° du mot
Exemple:	%NW0.2.4	Mot numéro 4 de la station 2 du réseau 0
	%NW3.2.4	Mot numéro 4 de la station 2 du réseau numéro 4

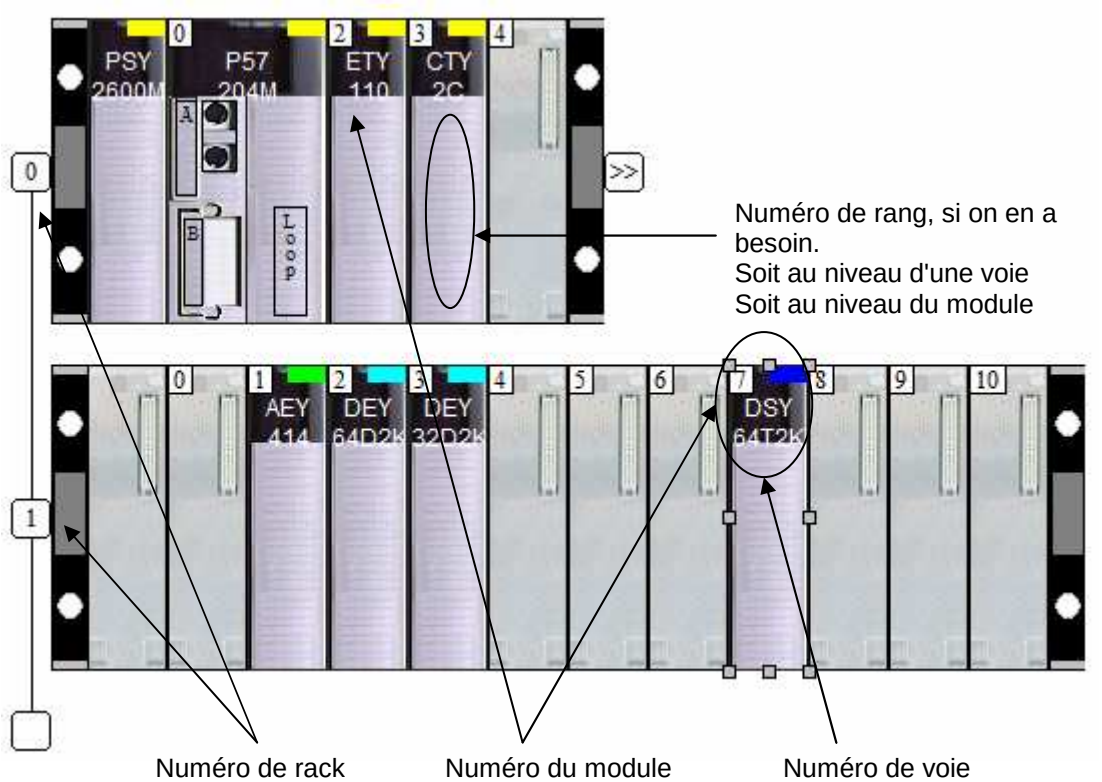
Il faut qu'un réseau soit configuré, soit FIPWAY soit ETHWAY et les mots commun doivent être configurés, il y a un total de 256 mots commun à répartir sur chaque station, sur le réseau FIPWAY, il y a systématiquement 4 mots par station, sur le réseau ETHWAY il y a 4, 8, 16, 32 ,64 mots à définir par station.

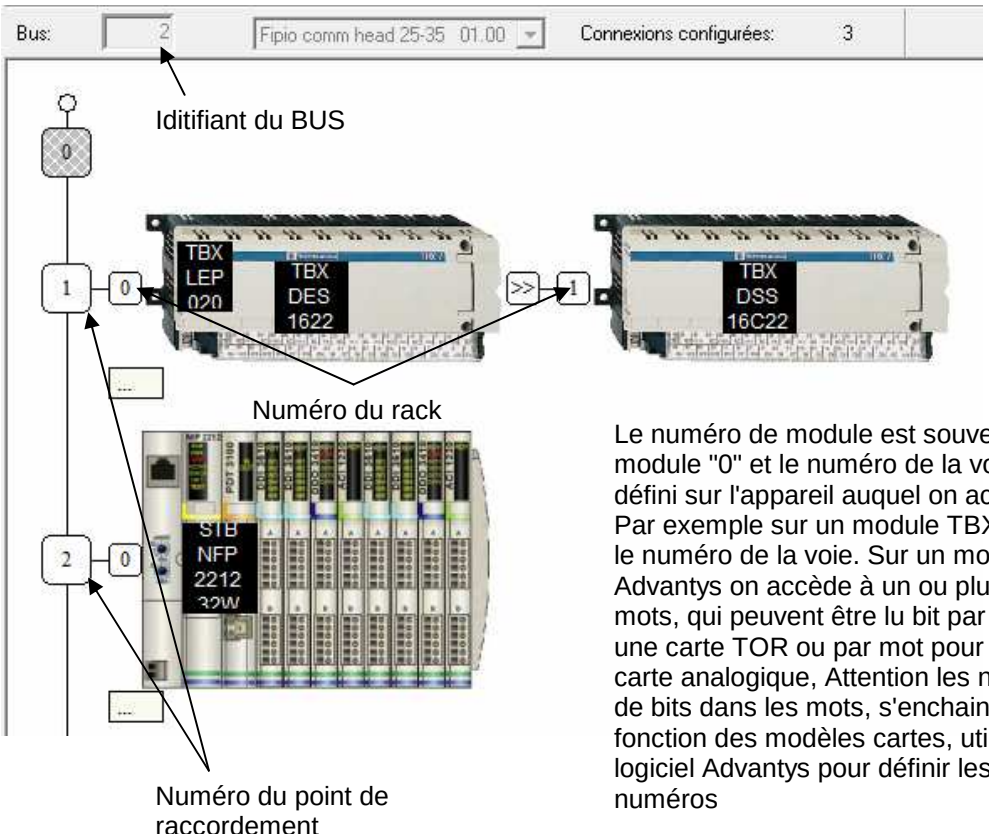


Numéro de réseau
Le numéro de la station est défini sur la carte elle même

Numéro de mot est défini de 0 à 3 pour cette station, mais peut être défini 0 à 7, 0 à 15, 0 à 31 ou 0 à 63

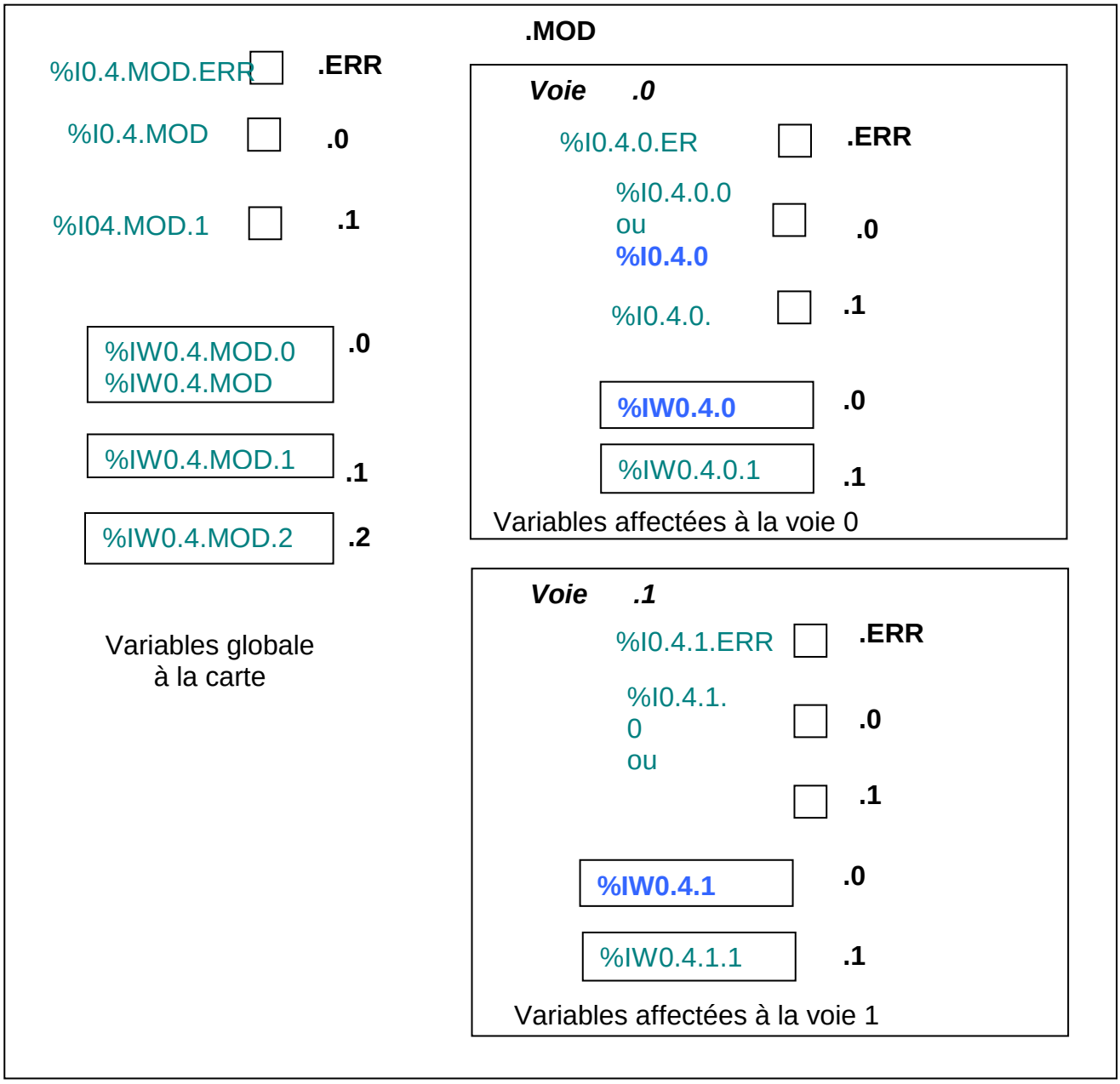
2. ADRESSES DES CARTES





Le numéro de module est souvent le module "0" et le numéro de la voie est défini sur l'appareil auquel on accède. Par exemple sur un module TBX, c'est le numéro de la voie. Sur un module Advantys on accède à un ou plusieurs mots, qui peuvent être lu bit par bit pour une carte TOR ou par mot pour une carte analogique, Attention les numéros de bits dans les mots, s'enchaine en fonction des modèles cartes, utilisez le logiciel Advantys pour définir les numéros

2.1. Exemple de l'objet carte d'entrée emplacement 4 dans le rack 0



3. LES MNEMONIQUES

Saisir les mnémoniques est très utile, il vaut mieux saisir un programme entièrement en symbole qu'en adressage absolu, c'est beaucoup plus lisible et compréhensible.

Il suffit d'aller dans la table des Mnémoniques et y entrer les différents éléments.

Le nom du symbole, sont type, éventuellement sont adresse réel et sa valeur par défaut, sont commentaire

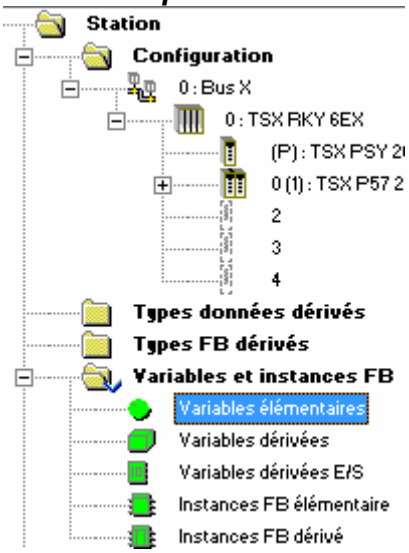
3.1. Voici un exemple

Variables | Types DDT | Blocs fonction | Types DFB

Filtre — Nom: * — EDT ☑ DDT ☐ IODDT ☐

Nom	Type	Adresse	Valeur	Commentaire
Doseur2_m	EBOOL			Acquittement mémorisé: Tapis doseur tremie 2
Doseur2_man	EBOOL			Mode manu: Tapis doseur tremie 2
Doseur2_nd	INT			Nb de demmarrage 1 heure: Tapis doseur tremie 2
Doseur2_o	EBOOL	%m215		Ordre de marche: Tapis doseur tremie 2
Doseur2_rem	EBOOL		TRUE	En remote: Tapis doseur tremie 2
Doseur2_rm	EBOOL			Retour de marche: Tapis doseur tremie 2
Doseur2_tc	EBOOL			Commande en manuel: Tapis doseur tremie 2
Doseur2_td	INT		5	Temps de discordance: Tapis doseur tremie 2
Doseur2_tdm	INT			Temps mesuré de discordance: Tapis doseur tremie 2
Doseur2_tdr	INT	%mw455		Temps de discordance capteur: Tapis doseur tremie 2
Doseur2_tdrm	INT	%mw535		Temps mesuré discordance capteur: Tapis doseur tremie 2
Doseur2_th	EBOOL		TRUE	Défaut thermique: Tapis doseur tremie 2
Doseur2_vit	INT			Consigne de vitesse: Tapis doseur tremie 2
Doseur3_acq	EBOOL			Acquittement: Tapis doseur tremie 3
Doseur3_amo	EBOOL			Condition amont: Tapis doseur tremie 3
Doseur3_arr	EBOOL			Mode arret: Tapis doseur tremie 3
Doseur3_aut	EBOOL			Mode auto: Tapis doseur tremie 3
Doseur3_ava	EBOOL			Condition aval: Tapis doseur tremie 3
Doseur3_c	EBOOL			Commande en automatique: Tapis doseur tremie 3
Doseur3_cel	EBOOL			Capteur de fin: Tapis doseur tremie 3
Doseur3_cs	DINT			Cpt secondes en service: Tapis doseur tremie 3
Doseur3_def	EBOOL			Défaut: Tapis doseur tremie 3
Doseur3_disc	EBOOL			Discordance: Tapis doseur tremie 3
Doseur3_dt	EBOOL			Defaut temps trop long: Tapis doseur tremie 3

3.2. On peut accéder à la table des symboles depuis variable et instance FB



4. ACCES AUX VARIABLE

On accède aux variables en indiquant leur nom ou leur adresse si elle est définie
.**Exemple** : titi accède à la variable titi
%mw10 accède à la variable %mw10
Si titi est déclaré de type INT avec l'adresse %mw10, on peut y accéder soit par le terme titi soit par son adresse %mw10

4.1. Accès aux bits de mots

On accède aux bits de mots en indiquant le mot suivi d'un point, puis le numéro du bit
Exemple : tutu.10 accède au bit 10 du mot tutu
La variable tutu doit être du type INT ou WORD

On accède au bit de mot, tous simplement en définissant une variable de type BOOL
Exemple : toto accède à une variable de type BOOL (bit de mot)
La variable doit être de type BOOL, son adresse peut être défini par exemple %mw20.5, le bit 5 du mot 20

4.2. Accès aux éléments d'une instance de structure

On accède à un élément d'une structure en indiquant le nom de l'instance suivi d'un point suivi de l'élément auquel on veut accéder

Exemple :	mastruct.ele1	accède à la variable ele1 de l'instance mastruct La variable mastruct doit être déclarer du type d'une structure

4.3. Accès aux éléments d'un tableau

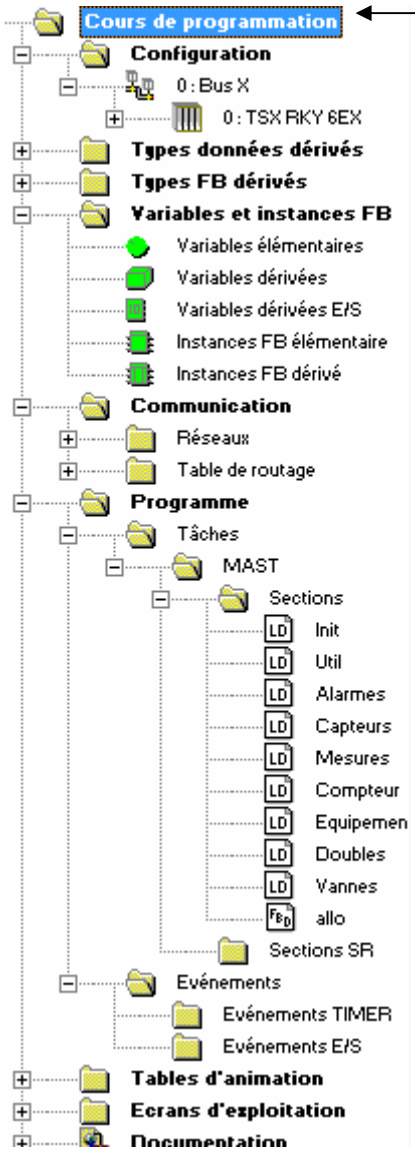
On accède à un élément d'un tableau en indiquant le nom du tableau suivi du rang entre crochets

Exemple :	tab[54]	accède à l'élément numéro 54 du tableau d'entier tab doit être de type ARRAY[n..m] OF INT, la valeur 54 doit être comprise dans la plage n..m
	tab2[9]	accède à l'élément 9 d'un tableau de réel allant de 0 à 19 tab2 doit être de type ARRAY[0..19] OF REAL
	tab3[index]	accède à l'élément indexé d'un tableau d'entier tab3 doit être de type ARRAY[0..9] OF INT index doit être de type INT l'index doit rester dans la zone 0 à 9
	tab4[5].ele1	accède à la variable ele1 de la structure numéro 5 d'un tableau de structure "azerty" tab4 doit déclarer sous la forme ARRAY[0..9] OF azerty

5. LE NAVIGATEUR

C'est simple, il suffit de lire pour comprendre

5.1. Vue structurelle



Nom du programme

Configuration de l'automate

Définition des variables

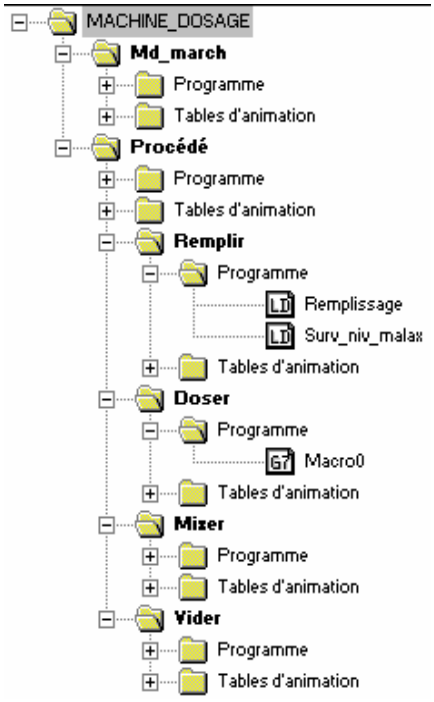
Définition des instances si

Configuration des réseaux

Ecriture des programmes

Définition des tables d'animations
Définition des Ecrans

Exemple de vue Fonctionnelle

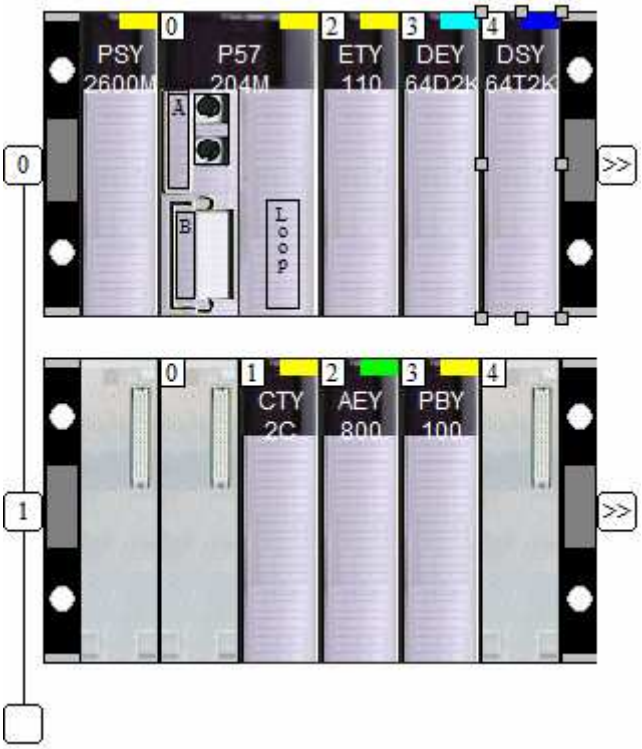


5.2. Vue fonctionnelle

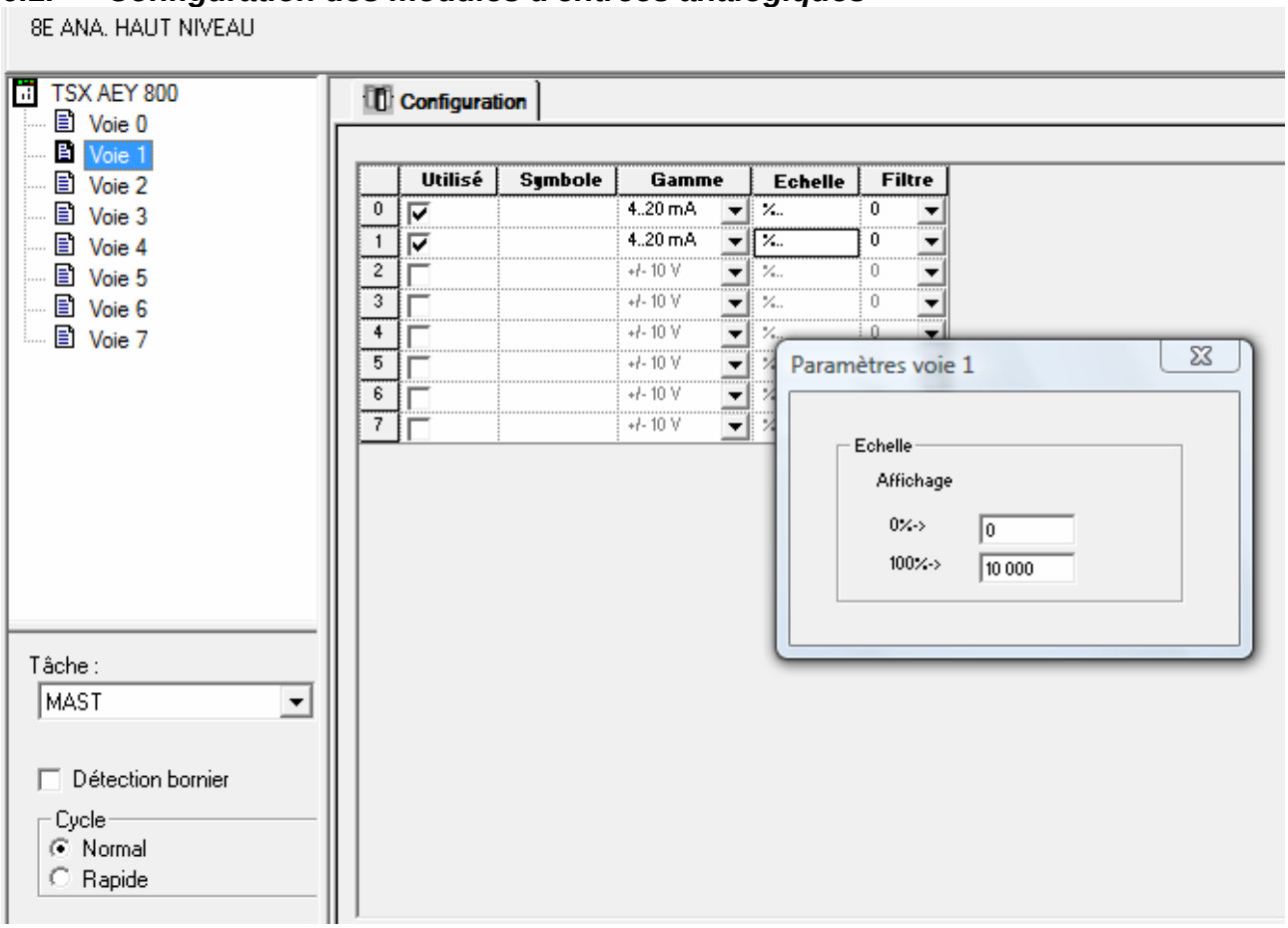
Une fois que vous avez configuré l'ensemble de l'automate et des variables, je vous suggère de passer en vue fonctionnelle pour écrire votre programme.
La première étape dans l'écriture d'un programme consiste à définir sa structure en partant d'une feuille blanche, c'est l'occasion de passer en vue fonctionnelle, ou vous aurez une vue claire de votre programme, vous aurez écrit les différents titres mais pas encore le programme, et vous verrez que se sera plus facile.

On accède au navigateur depuis le menu "outils" 

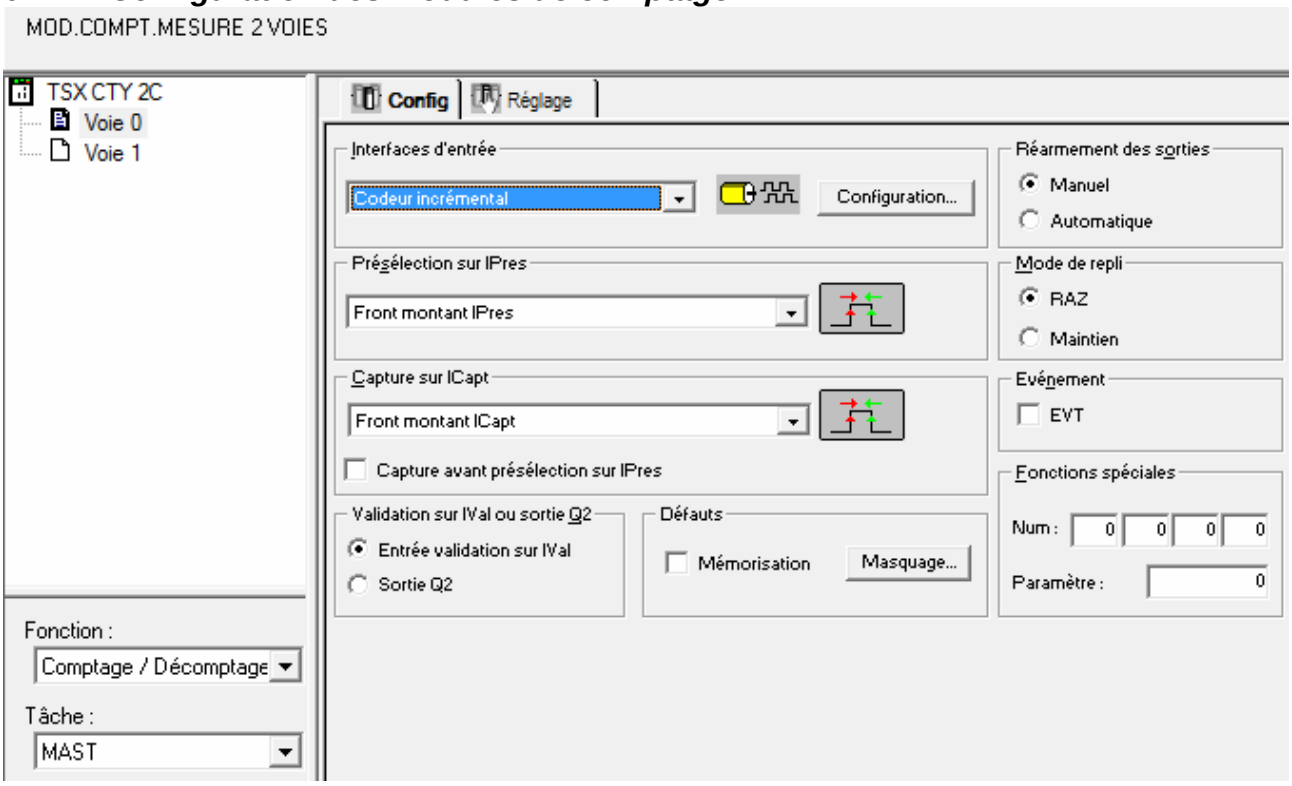
6. LA CONFIGURATION



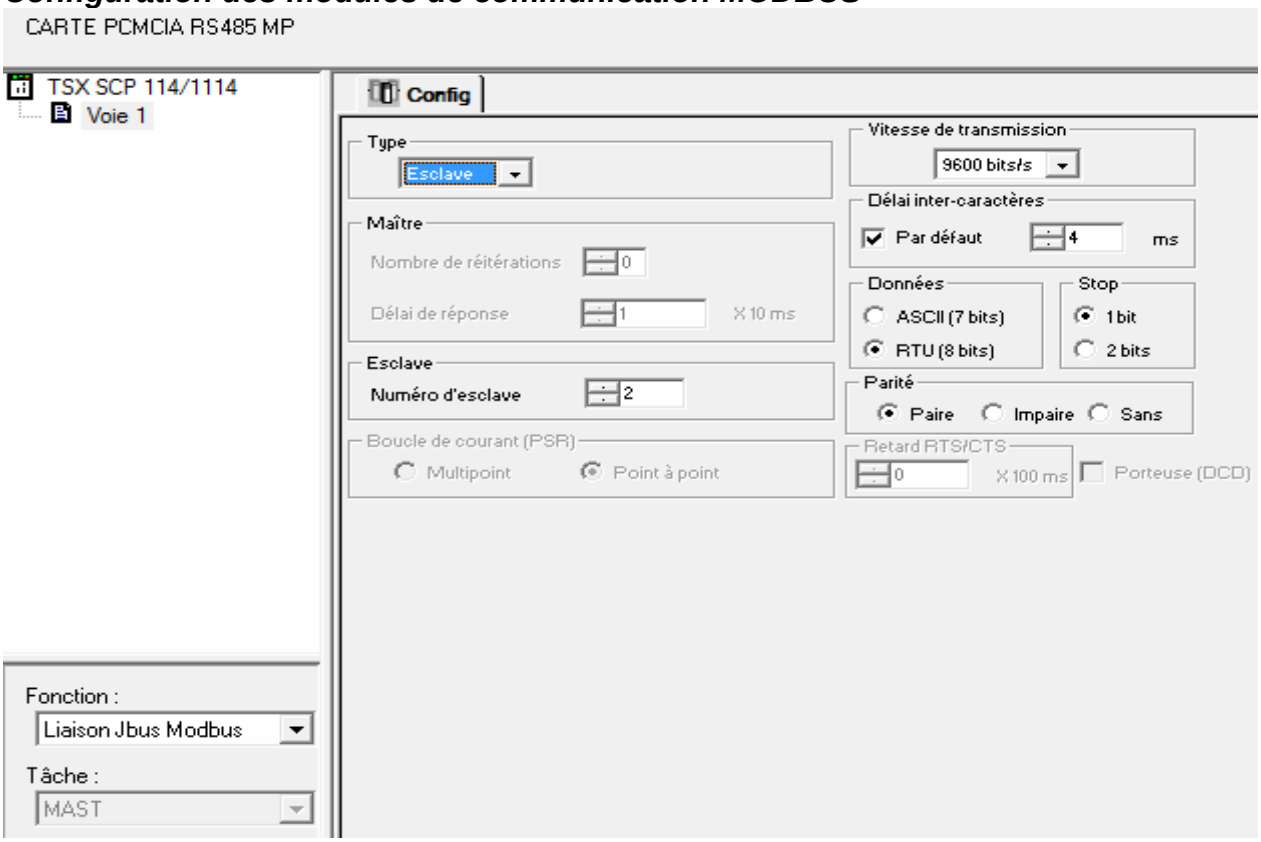
6.1. Configuration des modules d'entrées analogiques



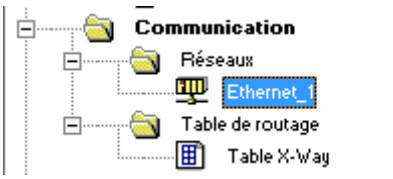
6.2. Configuration des modules de comptage



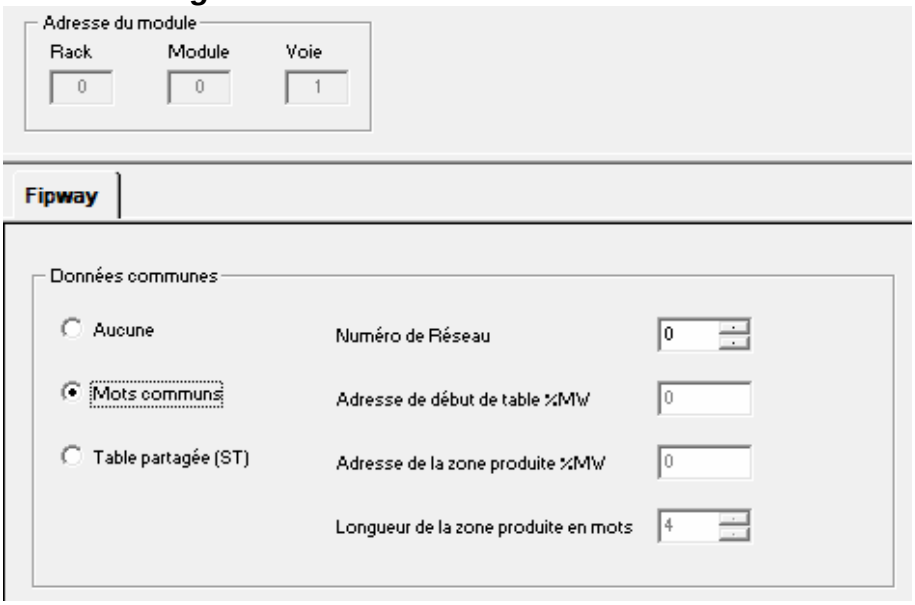
Configuration des modules de communication MODBUS



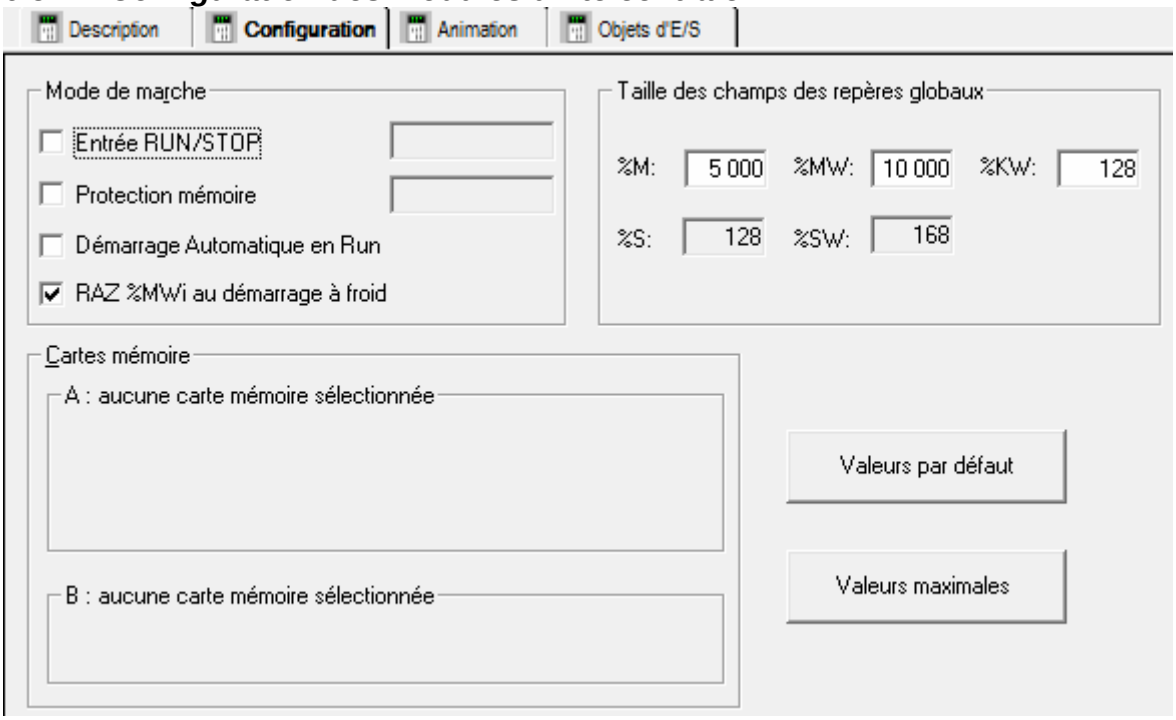
6.3. Configuration des modules de communication réseaux Ethernet



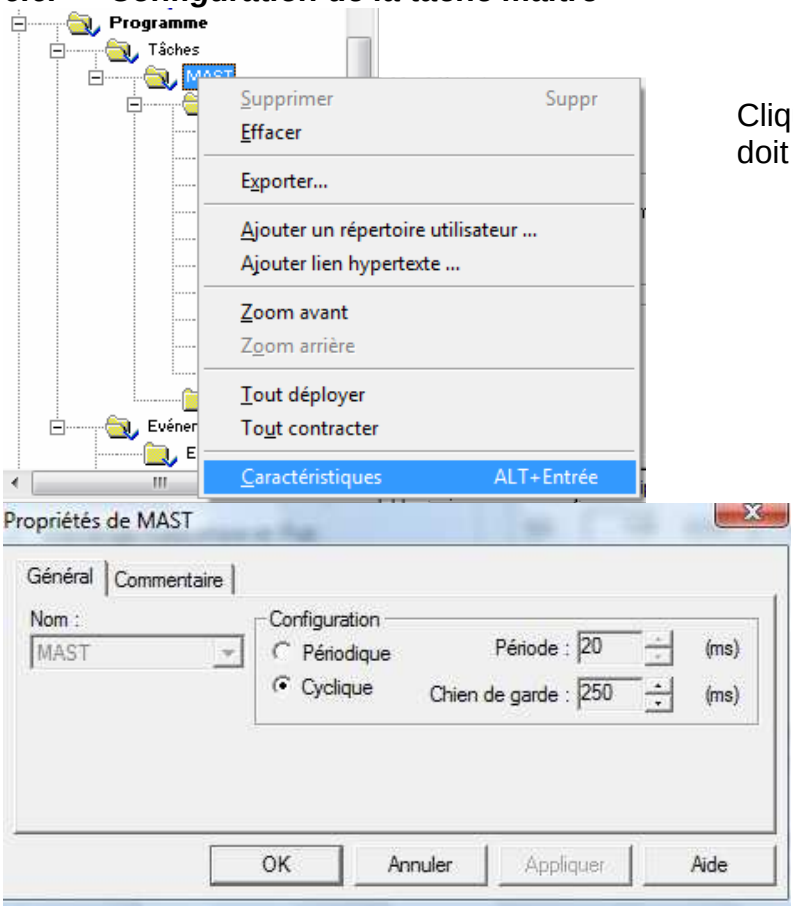
6.4. Configuration des modules de communication réseaux FIPWAY



6.5. Configuration des modules unité centrale



6.6. Configuration de la tâche maître

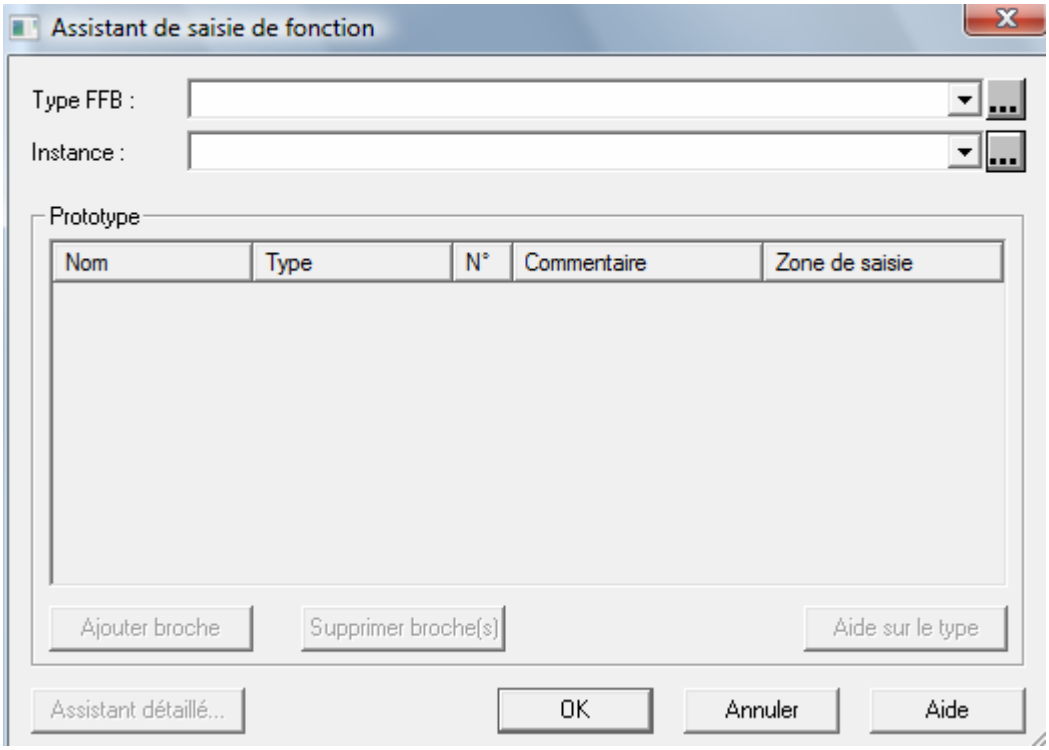


Cliquer avec le bouton droit sur "MAST"

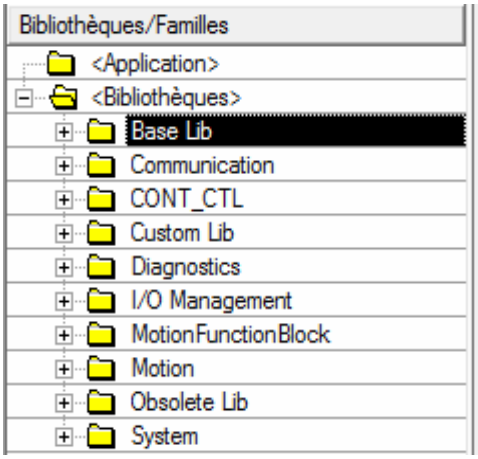
7. LES FONCTIONS DANS LA BIBLIOTHEQUE

Il y a les fonctions élémentaires EF, qui n'ont pas besoins d'être instanciées, et les blocs fonctions élémentaires EFB qui doivent être instanciées, on peu se faire soit même des EFB en langage C, mais il faut avoir le logiciel SDKC, sinon on peut faire en langage automate nos propres blocs fonctions dérivées DFB (voir chapitre 18)

Cliquer sur l'un des boutons suivants  **OU** 

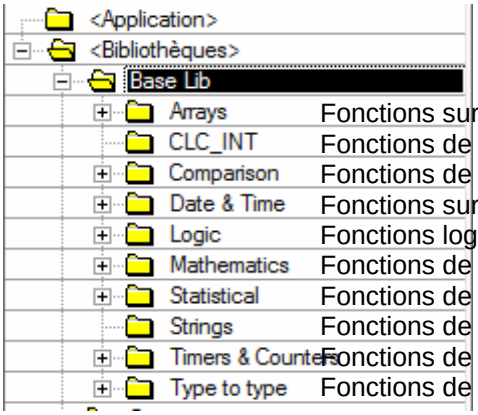


Cliquer sur les trois petits points après "Type FFB"



Fonctions de base
Fonctions communications
Fonctions de régulations
Fonctions utilisateurs
Fonctions de diagnostiques
Fonctions d'entrées sorties explicites
Fonctions pour les cartes d'AXES
Fonctions pour les cartes d'AXES
Ex fonction du PL7
Fonctions systèmes tel les contrôle d'événements, de mise à l'heure, gestion des grafcets ou encore la fonction SCHEDULE qui permet de monter un bit à certaines périodes

7.1. Les fonctions de bases standards



Fonctions sur les tableaux
Fonctions de régulation sur des entiers PID, PWM, SERVO
Fonctions de comparaisons, égal, supérieur, inférieur etc.
Fonctions sur les temps et les dates, ajouter, multiplier un temps, etc.
Fonctions logique sur mot ET, OU , XOR, décalage etc.
Fonctions de mathématique, addition, soustraction, multiplication etc. cosinus, exponentiel
Fonctions de statistique, valeur moyenne, maxi, mini, limite, etc.
Fonctions de chaîne de caractères, concatène, insert, extrait, remplace etc.
Fonctions de comptage et temporisation
Fonctions de conversions, BCD, ENTIER, TIME, REEL, CHAINE etc.

7.2. Les EF de communications

ADDR	Lit une adresse
CANCEL	Annule un échange
DATA_EXCH	Envoi ou reçoit des données
INPUT_BYTE	Lit des octets
INPUT_CHAR	Lit une chaine de caractère
OUT_IN_CHAR	Envoi une question attend la réponse
PRINT_CHAR	Ecrit une chaine de caractères
RCV_TLG	Reçoit un télégramme
READ_ASYN	Lis des bits ou mot s de manière asynchrone
READ_GDATA	Lis de donnée MODBUS +
READ_VAR	Lis de données standard
SEND_REQ	Envoi une requête
SEND_TLG	Envoi un télégramme
UNITE_SERVER	Active le serveur UNITE
WRITE_ASYN	Ecrit des bits ou mots de manière asynchrone
WRITE_GDATA	Ecrit des données MODBUS +
WRITE_VAR	Ecrit des données standard

7.3. Les EFB et EF de régulation

AUTOTUNE <EFB>	Réalise un "AUTO TUNE"
IMC <EFB>	Correcteur à modèle
PI_B <EFB>	Régulateur simple en PI
PIDFF <EFB>	Régulateur complet PID+FeedForward
SAMPLETM <EFB>	Défini la période d'échantillonnage
STEP2 <EFB>	Régulation à deux états (Température)
STEP3 <EFB>	Régulation à trois états (chaud/froid)
PID_INT <EF>	Régulateur PID sur des valeurs entières
PWM_INT <EF>	Sortie PWM sur valeurs entières
SERVO_INT <EF>	Sortie SERVO sur valeurs entières

7.4. Les EFB de sortie des régulateurs

MS <EFB>	Contrôle manuel de la sortie
MS_DB <EFB>	Contrôle manuel de la sortie avec bande morte
PWM1 <EFB>	Sorite en PWM
SERVO <EFB>	Sortie en SERVO
SPLRG <EFB>	Sortie en Split Range

7.5. Les EFB d'entrée des régulateurs

RAMP	<EFB>
RATIO	<EFB>
SP_SEL	<EFB>

Générateur de rampe de consigne
Génère un RATIO
Sélectionne la consigne

7.6. Les EF sur chaîne de caractères

CONCAT_STR	<EF>
DELETE_INT	<EF>
EQUAL_STR	<EF>
FIND_INT	<EF>
INSERT_INT	<EF>
LEFT_INT	<EF>
LEN_INT	<EF>
MID_INT	<EF>
REPLACE_INT	<EF>
RIGHT_INT	<EF>

Concatène deux chaines
Supprime un sous chaine
Compare deux chaines
Recherche une chaine dans une chaine
Insert des caractères dans une chaine
Extrait la partie gauche d'une chaine
Retourne la longueur d'une chaine
Extrait le milieu d'une chaine
Remplace une chaine
Extrait la partie droite d'une chaine

7.7. Présentation des EF mathématiques

ABS	Valeur absolu
ACOS	Arc cosinus
ADD	Addition
ASIN	Arc sinus
ATAN	Arc tangente
COS	Cosinus
DEC	Décrémentation d'un point
DIV	Division
DIVMOD	Division ave modulo
EXP	Exponentiel
INC	Incrémentation d'un point
LN	Logarithme naturel
LOG	Logarithme en base 10
MOD	Modulo
MOVE	Assignation
MUL	Multiplication
NEG	Inverse la sortie
SIGN	Reconnaît le signe
SIN	Sinus
SQRT	Racine carré
SUB	Soustraction
TAN	Tangente

7.8. Présentation des EF statistiques

AVE	Valeur moyenne
LIMIT	Valeur limite
LIMIT_IND	Valeur limite avec indication
MAX	Valeur maxi
MIN	Valeur mini
MUX	Multiplexeur
SEL	Sélectionne une valeur sur deux

7.9. Présentation des EF de comparaisons

EQ	Egal
GE	Plus grand ou égal
GT	Plus grand
LE	Plus petit ou égal
LT	Plus petit
NE	Non égal, différent

7.10. Présentation des EF logiques

AND	ET
FE	Front descendant
NOT	PAS
OR	OU
RE	Front montant
RESET	Met à zéro
ROL	Rotation sur la gauche
ROR	Rotation sur la droite
SET	Met à UN
SHL	Décale sur la gauche
SHR	Décale sur la droite
XOR	Ou exclusif
TRIGGER	Détecte tout front

7.11. Présentation des EF sur tableaux

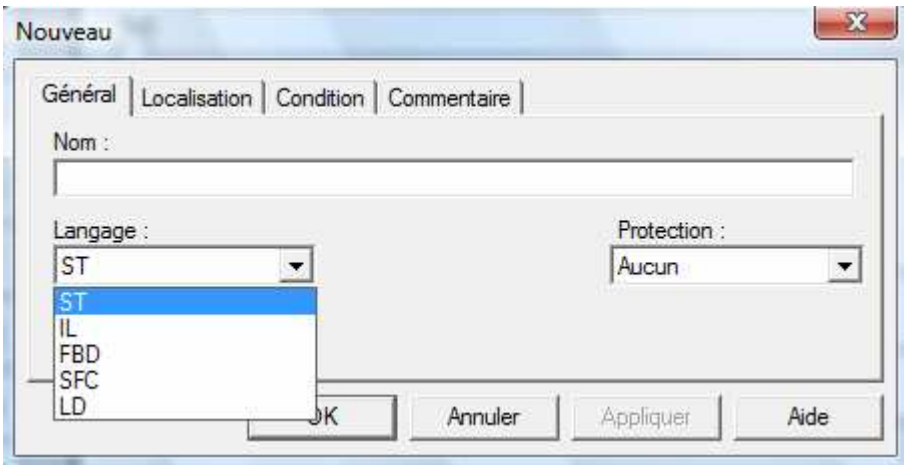
ADD_ARxx	Additionne deux tableaux
COPY_ARxx	Copy un tableau
DIV_ARxx	Divise deux tableaux
EQUAL_ARxx	Compare deux tableaux
FIND_EQ_ARxx	Recherche l'équivalence dans deux tableaux
FING_GT_ARxx	Recherche le plus grand des deux tableaux
FIND_LT_ARxx	Recherche le plus petit des deux tableaux
LENGTH_ARxx	Retourne la longueur d'un tableau
MAX_ARxx	Retourne la valeur maximum d'un tableau
MIN_ARxx	Retourne la valeur minimum d'un tableau
MOD-ARxx	Retourne le modulo de deux tableaux
MOVE_AREBOOL_xx	Ecrit une valeur dans une chaîne de bit
MOVE_xx_ARxx	Assigne une valeur à un tableau

7.12. Présentation des EF sur date et temps

ADD_DT_TIME	Ajoute une durée à une date
ADD_TOD_TIME	Ajoute une durée à un temps
DIVTIME	Divise le temps
MULTIME	Multiplie le temps
SUB_DATE_DATE	Ecart entre deux date
SUB_DT_DT	Ecart entre deux dates
SUB_DT_TIME	Soustrait une durée à une date
SUB_TOD_TIME	Soustrait une durée à un temps
SUB_TOD_TOD	Soustrait deux date

Il y a beaucoup d'autres fonction que je ne vous présente pas, mais que vous pouvez aller voir, tel les fonctions sur les compteur et temporisateurs, les fonction d'échange explicite sur les entrées sortie, les fonction de diagnostiques, les fonctions systèmes, les fonctions sur les mouvements, les anciennes fonctions du PL7 (obsolète lib)

8. LES SECTIONS DE PROGRAMMES



Une section peut être créée en Littéral (**ST**) en liste d'instruction (**IL**) en Bloc fonction (**FBD**) en grafet (**SFC**) ou en LADDER (**LD**)
Vous devez lui donner un nom, vous pouvez lui donner une "condition", c'est-à-dire un bit d'autorisation d'exécution, vous pouvez lui affecter un "commentaire" pour expliquer son rôle et vous pouvez éventuellement la protéger en lecture seule voire même qu'elle ne soit pas lisible.
Lorsque vous protéger des sections, il faut alors protéger l'ensemble de l'application et lui donner un mot de passe (qu'il ne faut pas oublier).

Vous pouvez créer une section directement dans la vue fonctionnelle, ce qui est préférable, sinon vous pouvez donner sa "localisation", sachant que de toute façon, vous pourrez changer sa localisation en faisant un glisser déplacer.

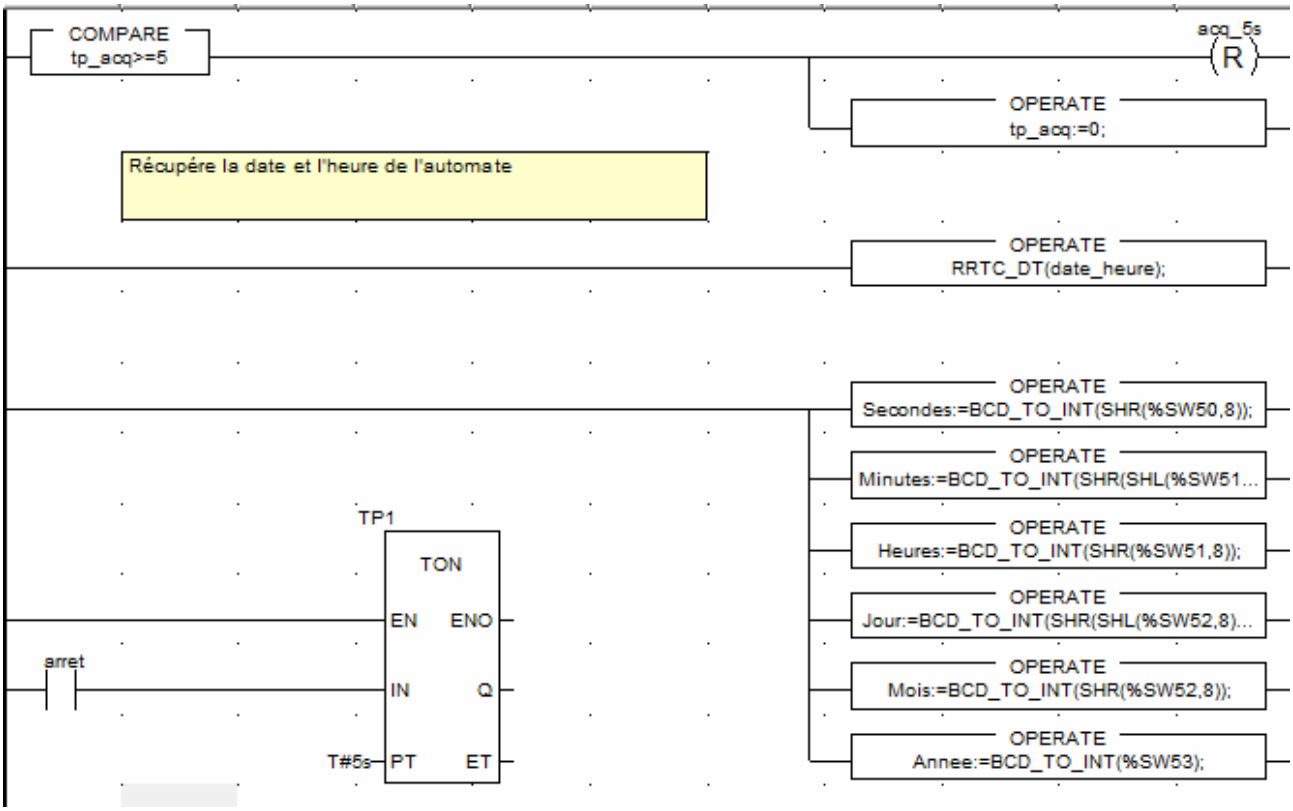
Le choix du langage dépend uniquement de vous. Bien sur elle dépend de ce que vous imposent votre client.
Si vous avez le choix, choisissez le langage LADDER pour toutes les fonctions de bases, choisissez le LITTERAL si vous avez de gros calcul à faire, ou des algorithmes compliqués, choisissez la représentation en BLOCS fonctions si vous avez des boucles de régulations complexes à réaliser, enfin choisissez le GRAFCET si vous avez des séquences à réaliser,

Mise en garde : N'oubliez pas que le grafcet est compliqué à modifier, s'il faut modifier sa structure, s'il y a beaucoup de saut d'étapes à faire, si un imprévu arrive, dans certain cas la structure est entièrement à revoir, il vaut mieux utiliser un autre langage avec des mémoire ou des valeurs de mots (case of).

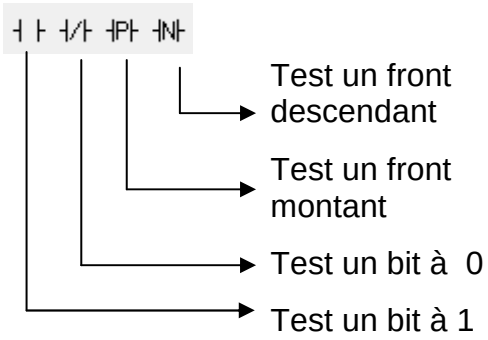
La question qu'il faut se poser est: l'action de je tente de réaliser dépend elle réellement d'une autre action, par exemple J'attends une réponse, dépend elle de je pose une question, la réponse est oui, dans ce cas le grafcet peut être envisagé, un autre exemple: je sors la butée dépend elle de je fais avancé le tapis, la réponse est ambigu, évidement si mon tapis n'avance pas la pièce n'arrivera pas, on serai tenté de dire "oui", mais en fait la réponse est que cela dépend du numéro de pièce, la butée ne sortira que lorsque le bon numéro de pièce arrivera et le grafcet n'est pas adapté au problème.

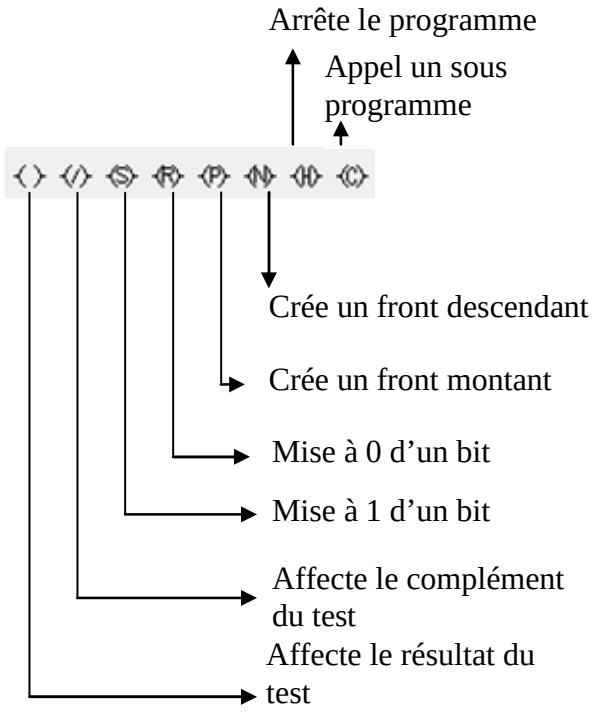
9. LE LANGAGE LADDER OU "LD"

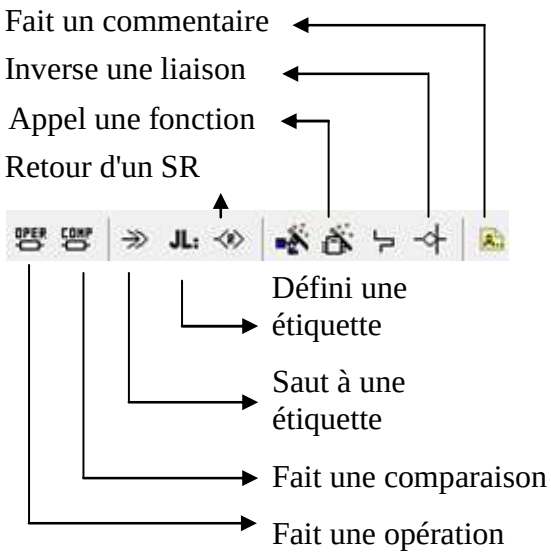
9.1. Exemple d'un programme en LADDER



9.2. Présentation de la barre d'outils







Lors d'un appel d'une fonction, voir chapitre 5, l'ensemble des fonctions disponibles, les temporisateurs les compteurs y est, mais bien d'autres fonctions encore. Vous avez aussi les fonctions du PL7-PRO qui existent, allez dans "obsolete lib", mais essayez d'utiliser les nouvelles fonctions.

10. LE LANGAGE LITTERAL STRUCTURE OU "ST"

Le langage littéral structuré se programme sous forme de phrases d'une longueur maximale de 300 caractères, comportant éventuellement un commentaire et une étiquette, suivant le même principe que le langage liste d'instructions.

10.1. Structure de programme

Structure inconditionnelle

Une suite d'actions séparées par des ";"

<Action>;<Action>;<Action>;
<Action>;
<Action>;

Une action fini toujours par un **";"**

Structures conditionnelle

```
IF <condition> THEN
        <programme>
ELSE
        <programme>
END_IF;
```

```
IF <condition> THEN
        <programme>
ELSEIF <condition> THEN
        <programme>
ELSE
        <Programme>
END_IF;
```

(Le nombre de ELSEIF est illimité)

```
CASE <variable> OF
        <valeur1>: <Programme1>
        <valeur2>:<Programme2>
        <valeur3>:<Programme3>
END_CASE;
```

Structures itératives

```
WHILE <condition> DO
        <programme>
END_WHILE;
```

```
REPEAT
        <programme>
UNTIL <condition> END_REPEAT;
```

!!! Attention : l'UC ne fait rien d'autre pendant le programme, c'est à dire qu'elle ne va pas tester les entrées ni mettre à jour les sorties

Structure répétitive

```
FOR <indice>:=<valeur départ> TO <Valeur arrivé> DO
        <programme>
END_FOR;
```

Instruction de saut

```
JMP <étiquette>

<étiquette>:
```

Le mot clé "EXIT"

Le mot clé "**EXIT**" permet de sortir prématurément d'une boucle

Appel des fonctions EF

Appel en faisant passer les paramètres de manières informelles, c'est-à-dire en passant tous les paramètres, dans l'ordre.

Exemples :

Appel d'un EF de soustraction de deux durée

```
duree := SUB_TOD_TOD(heure1, heure2);
```

Appel d'un PID simple

```
PID_INT('lic', 'm',pv, m_a, Tpara, sortie);
```

Appel de la fonction multiplication

```
Produit := MUL (Facteur1, Facteur2) ;
Produit := Facteur1 * Facteur2 ; on aurait pu résoudre le problème de cette façon.
```

Appel de la fonction exponentiel

```
Expo := EXP_REAL(valeur);
```

Appel de fonctions EFB ou DFB

Appel en faisant passer les paramètres de manières formelles, c'est-à-dire en nommant les paramètres, cette forme à l'avantage de ne faire passer que les paramètres utiles

Exemples :

Appel d'un DFB instancié

MON_DFB (IN1 := var1, IN2 := var12, OUT => var3);

Appel de la fonction PIDFF instancié

PID1 (PV:=ProcessVariable,SP:=Setpoint,MAN_AUTO:=OperatingMode,
PARA:=FIC_Eau_Para, OUT:=sortie, STATUS=>Etat);

Appel d'une temporisation instancié

TP1 (IN:=Start, PT:=t#5s, Q=>Outputa);

Appel de la fonction gestion d'un grafcet instancié, *on ne fait que l'initialisation*

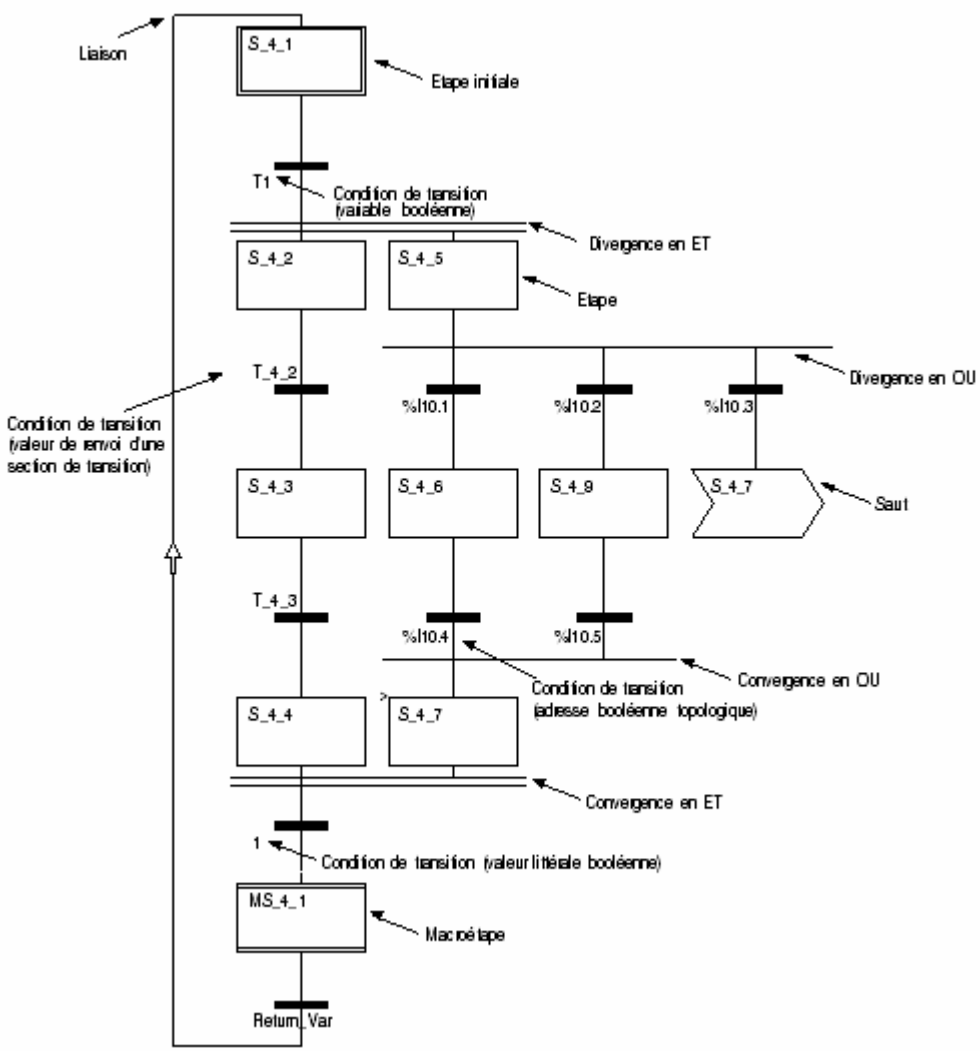
G7CTRL (CHARTREF:=G7, INIT:=init_g7) ;

10.2. Exemples de programme en littéral structuré

```
%M11:=%M10 AND (%I0.3.2 OR %I0.3.3);(* structure inconditionnelle *)

FOR %MW50:=0 TO 20 DO
        IF (%MW100[%MW50]> 10) THEN
                SET (%M20);
                EXIT; (* Quitte la boucle FOR *)
        END_IF;
END_FOR;

REPEAT
        INC (%MW4);
        SET (%M10[%MW4]);
UNTIL (%MW4 >=10) END_REPEAT;

IF (%MW10<>0) THEN
        WHILE NOT %M0[%MW10] and (%MW10<16)DO
                IF(%MW10 MOD 3=0)THEN
                        SET (%M0[%MW10]);(* mise à 1 des bits modulo 3 *)
                END_IF;
                INC (%MW10);
        END_WHILE;
END_IF;
```

11. LE GRAFCET OU "SFC"

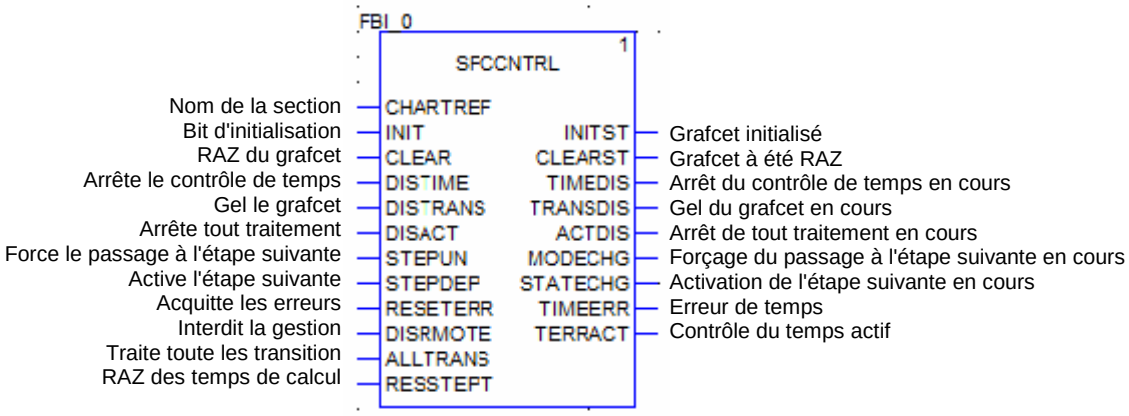
11.1. Exemple d'un grafcet



Chaque grafcet se défini dans une section et porte son nom, on peut donc avoir autant de grafcet indépendant les uns des autres
On peut initialiser, figer, vider, prépositioner n'importe quel grafcet, il suffit d'utiliser la fonction SFCCNTRL

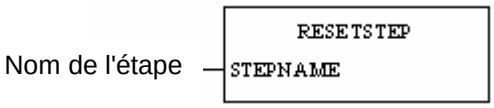
11.2. La fonction SFCCNTRL

Cette fonction permet de gérer un grafcet, on peu l'initialiser, le figer, le forcer etc.



11.3. Les fonctions SETSTEP et RESETSTEP

Ces fonctions sont utile pour mettre un 1 ou à 0 une étape, *à n'utiliser que dans des circontences particulières.*



11.4. Les variables associés aux étapes

Les variables associés aux étapes de grafcet son de type de la structure **SFCSTEP_STATE**

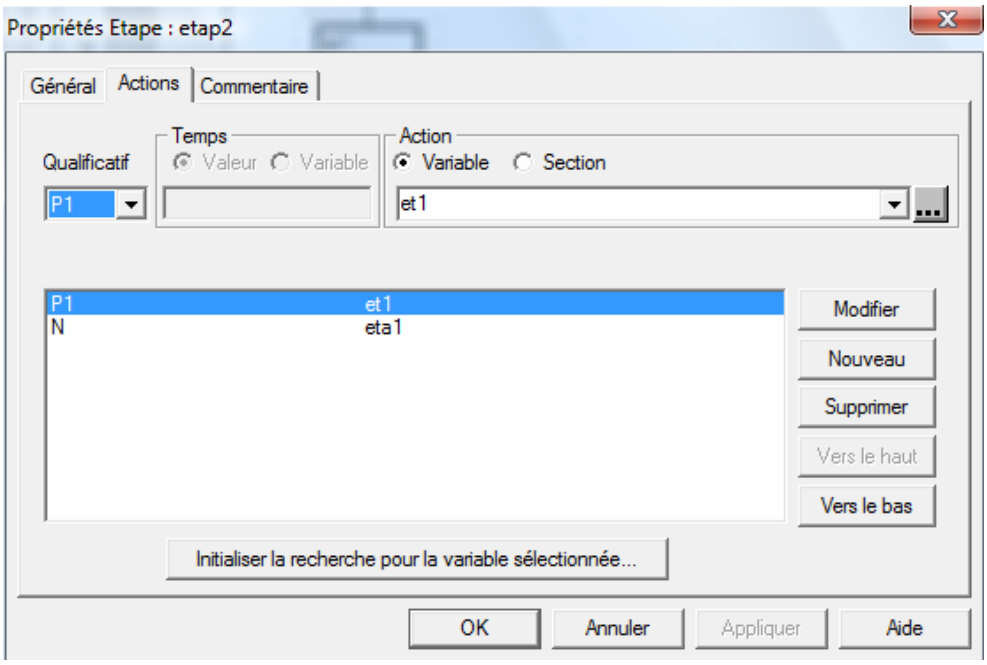
Chaque variable porte le nom de l'étape, par exemple pour l'étape dont le nom est S_4_9, il y a une variable qui porte ce nom et les champs de la structure s'appellent:

S_4_9.x	état de l'étape 0 ou 1
S_4_9.t	Temps d'activation de l'étape en durée de type TIME "T#"
S_4_9.tminErr	Erreur de durée d'étape, temps trop court
S_4_9.tmaxErr	Erreur de durée d'étape, temps trop long

On peut tester ces bits de l'étape dans l'ensemble du programme
On peut modifier le nom par défaut de l'étape

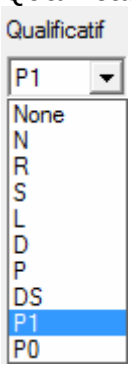
11.5. Actions associer aux étapes

Pour réaliser les actions d'un étape, on a plusieurs possibilités



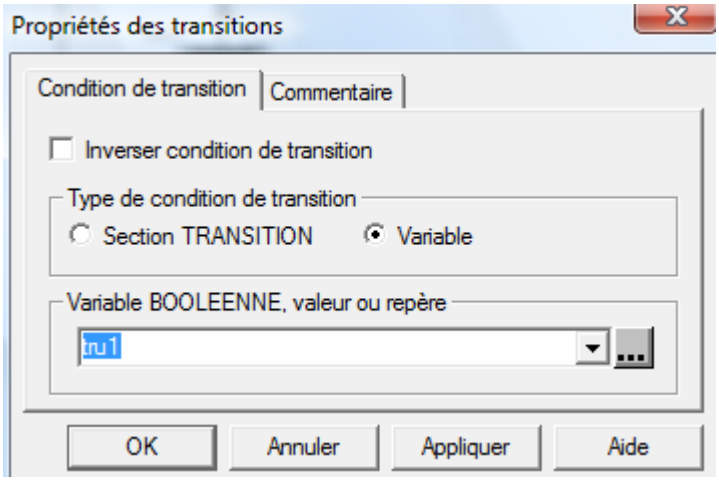
On peut lui associé plusieurs sections de programme qui seront réalisé conformément au "qualificatif"
On peut lui associé plusieurs bits variables qui seront réalisé conformément au "qualificatif"

Qualificatifs



N : Normal
R : Raz l'action précédente
S : Mémorise l'action qui sera désactivés par une étape "R"
L : Action temporisé, le temps est défini ou dans une variable
D : Action retardé, le temps est défini ou dans une variable
P : Action ne dure d'un tour de scrutation
DS: Action retardé puis mémorisé, désactivé par une étape "R"
P1: Action sur front montant, s'exécute en premier, "Equivalent à l'activation du PL7"
P0: Action sur front descendant, s'exécute en dernier "Equivalent à la désactivation du PL7"

11.6. Transitions associés aux étapes



On peut associé un bit ou une section à chaque transitions
On peut inverser la condition de transition, utile pour une divergence en OU

12. LE LANGAGE LISTE D'INSTRUCTION OU "IL"

12.1. Présentation

Le langage IL se programme sous forme de phrases de 128 instructions maximum, comportant 1 commentaire et une étiquette.

Exemple

```
! (* Attente séchage *)          ! indique le début de la phrase (* *) commentaire
%L2:                             Etiquette facultative
      LD    %M2                  Liste d'instructions
      OR    %I1.1
      ST    %Q2.4

! (* Une autre phrase *)
%L7:
      LD    TRUE
      [%MW2:=%MW4*5/SQRT(%MW20)]
```

Etc.

REMARQUE : Une phrase peut s'écrire en une seule ligne, l'éditeur la présentera sous la forme ci dessus après validation.

Exemples

```
! (*phrase IL*) %L4: LD [%MW10<4] AND [%MW20>10] [%MW40:=%MW50/2]
!(* autre phrase *) %L20 : LD %M10 AND %M11 ST %M12
!(* Une autre *) %L100 : LD %M20 AND %M21 OR ( %M22 AND  %M23 ) ST %M24
```

Etc.

12.2. Les instructions

INSTRUCTIONS BOOLEENNES

LD	Charge un résultat booléen (commence une phrase)
LDN	Charge le complément
LDF	Charge le front montant
LDR	Charge le front descendant
AND	Et
OR	Or
ANDN	Et pas
ORN	Ou pas
ANDF	Et front montant
ORF	Ou front montant
ANDR	Et front descendant
ORR	Ou front descendant

XOR	Ou exclusif
XORN	Ou pas exclusif
XORF	Ou front montant exclusif
XORR	Ou front descendant exclusif
ST	Range le résultat
STN	Range le complément

MPS	Stock (empile) le résultat booléen (pour une utilisation ultérieure)
MPP	Destock (dépile) un résultat
MRD	Lit la dernière valeur stockée sans la dépiler

VALEUR « VRAI » ET « FAUX »

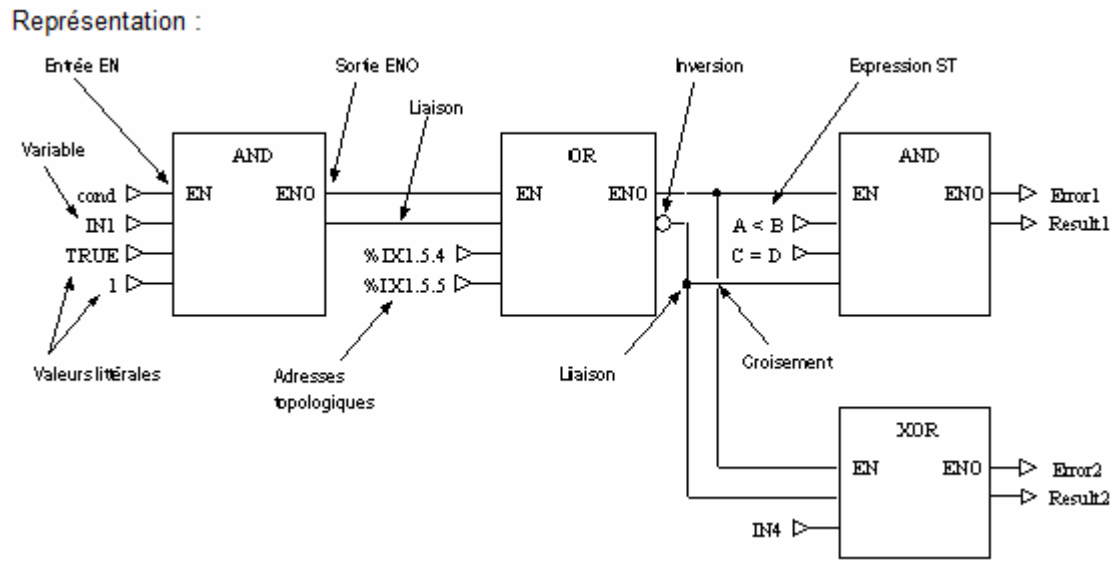
Les valeurs "vrai" ou "faux" peuvent êtres utilisés dans des équations booléennes, en général pour commencer une équation.

TRUE	toujours Vrai
FALSE	toujours Faux

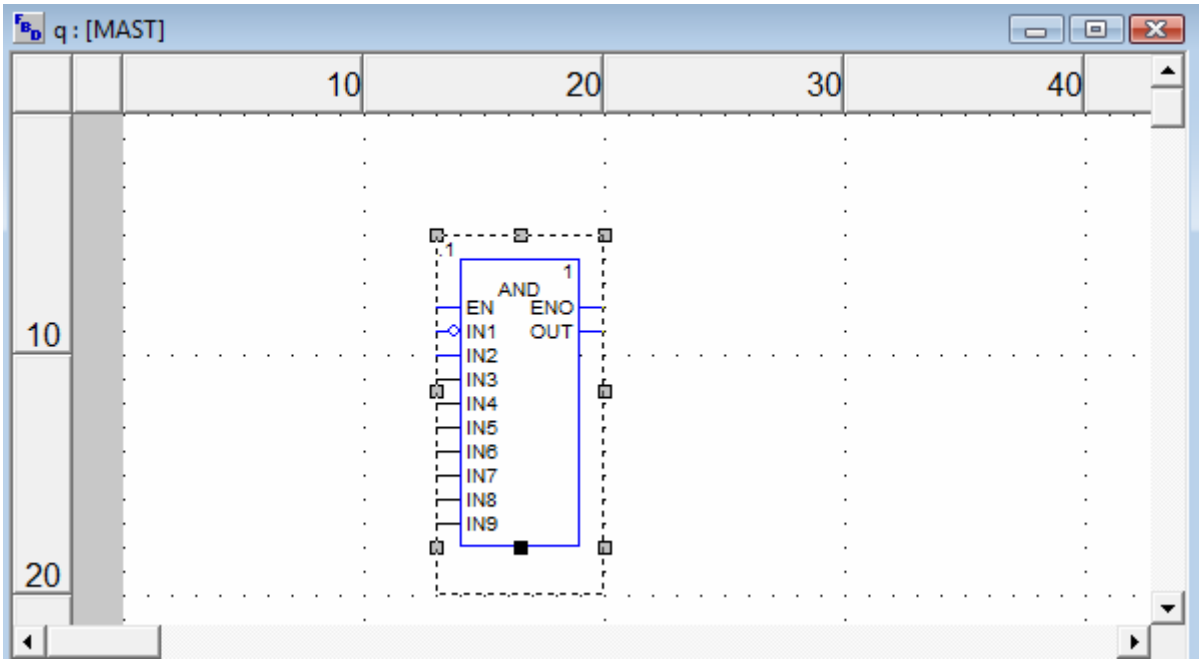
Exemple

```
LD    TRUE
[%MW2:=%MW10/5]
```

13. LE LANGAGE BLOCS FONCTIONNELS OU "FBD"



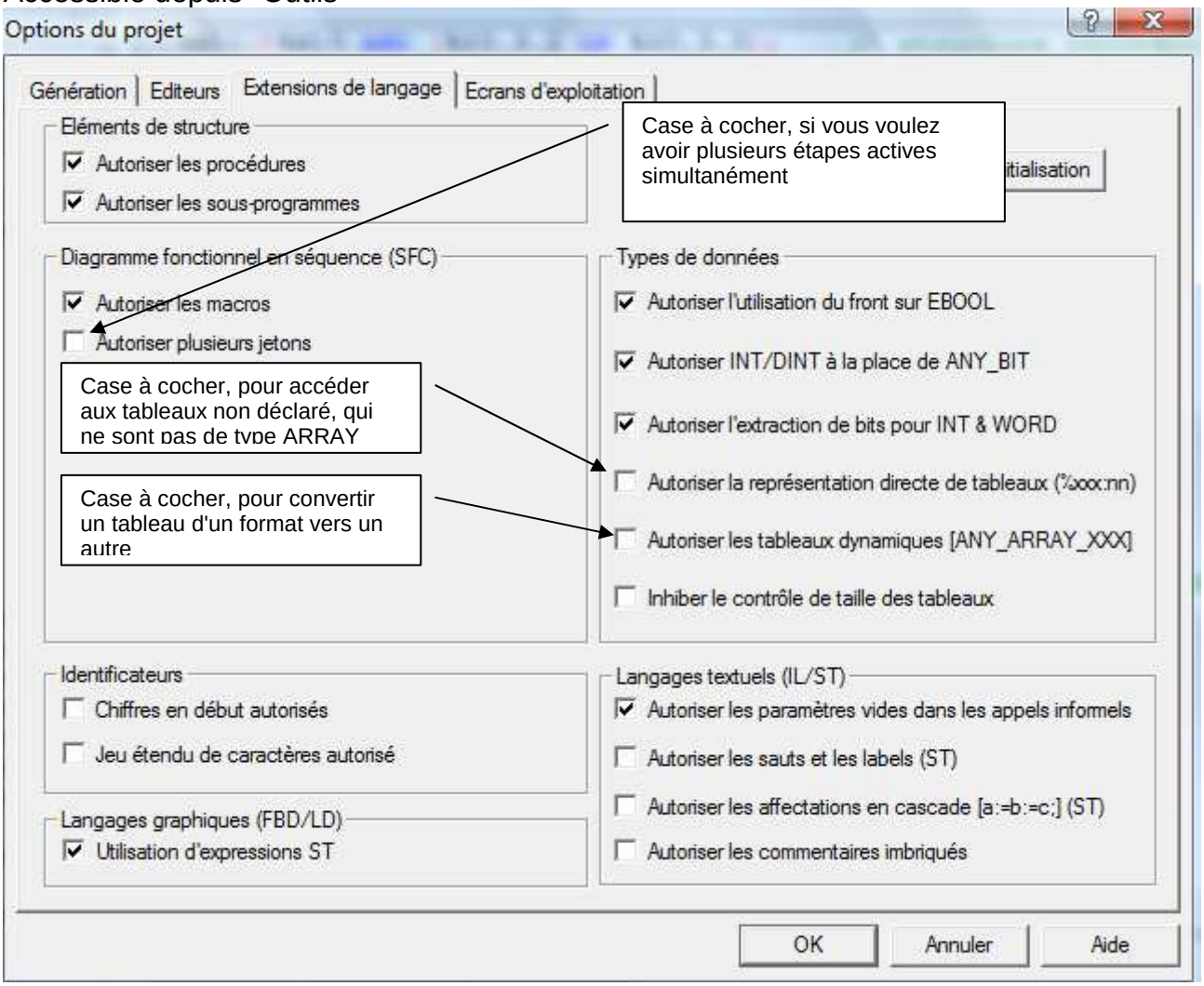
Voyez ici, comment augmenter le nombre d'entrées d'une fonction, il suffit d'élargir la fonction elle-même, en cliquant avec le bouton gauche sur la poigné.



L'augmentation du nombre d'entrée n'est possible qu'avec certaines fonctions.

14. LES OPTIONS DU PROJET

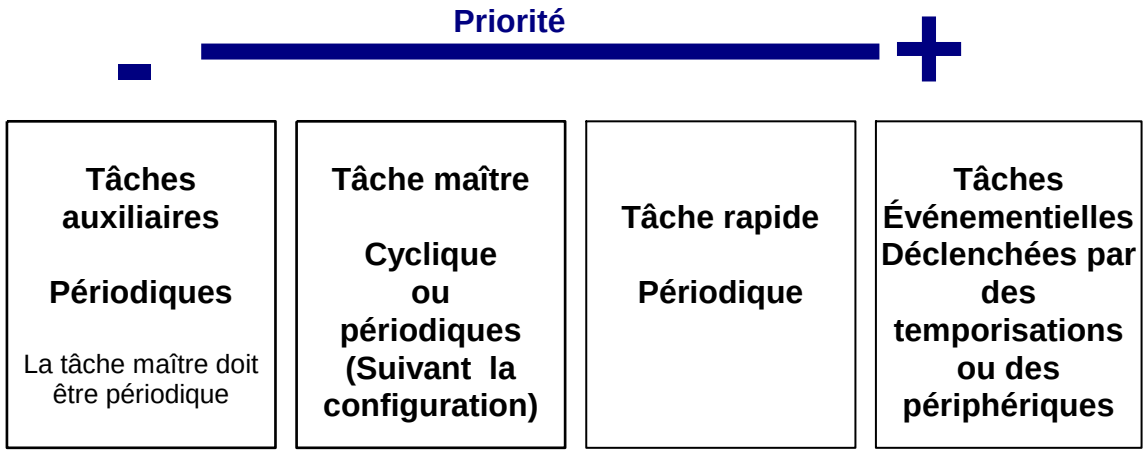
Accessible depuis "Outils"



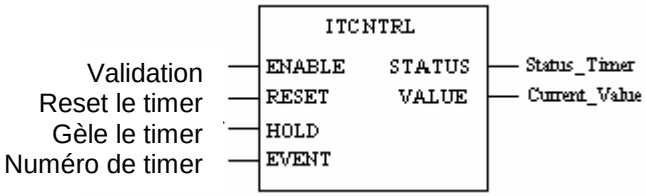
Si vous utilisez les tableaux issu du PL7, par exemple le tableau %mw10:10, si vous voulez accéder à un élément de ce tableau %mw10[%mw50] par exemple, cette syntaxe ne sera pas accepté ou alors, il faudra cocher la case "Autoriser le représentation direct de tableaux (%xxx:nn)

Si vous voulez transférer une chaine de bit dans un mot, il faut utiliser la fonction "move_arebool_int", le problème c'est que se ne sera pas accepté, il faudra cocher la case "Autoriser les tableaux dynamiques [ANY_ARRAY_XXX]", ou sinon utiliser la fonction "bit_to_word", l'inconvénient c'est qu'il faut écrire l'ensemble des 16 bits à convertir.
Ou encore écrivez une fonction DFB qui le fera, en entrée un tableau de bit, en sortie un entier et vous affecter chaque bit du mot avec chaque bit du tableau.

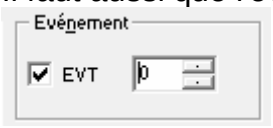
15. LES DIFFRENTES TACHES



Pour déclencher les tâches "timer" , il faut utiliser la fonction **ITCNTRL**

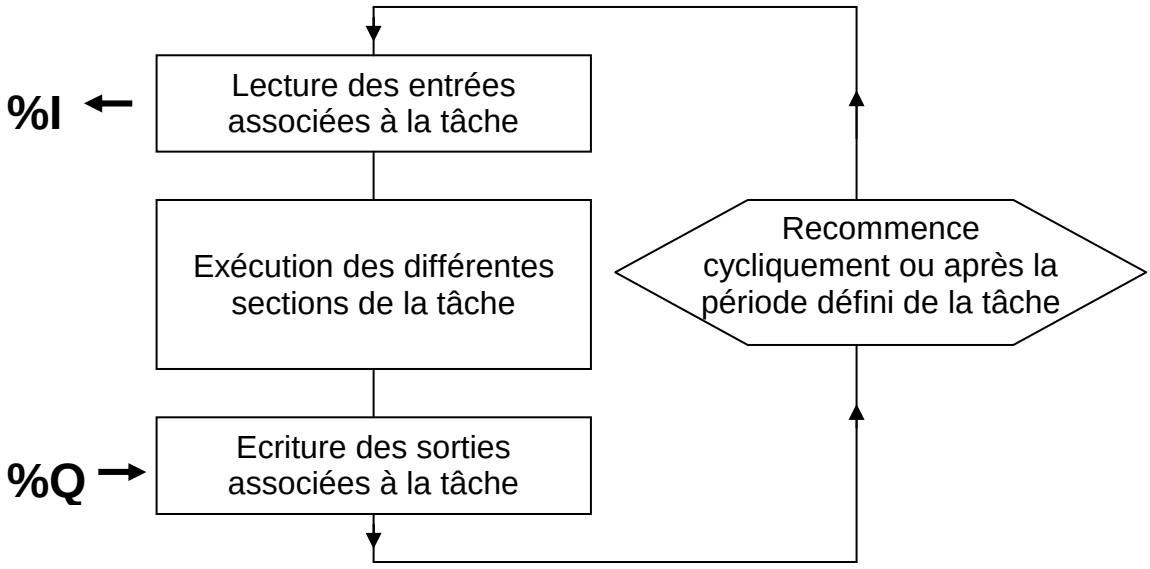


Pour autoriser les événements, vous pouvez utiliser le bit systéme **%S38** ou utiliser les fonctions **UNMASKEVT** ou **MASKEVT**
Il faut aussi que l'événement soit choisis dans la configuration de la carte périphérique

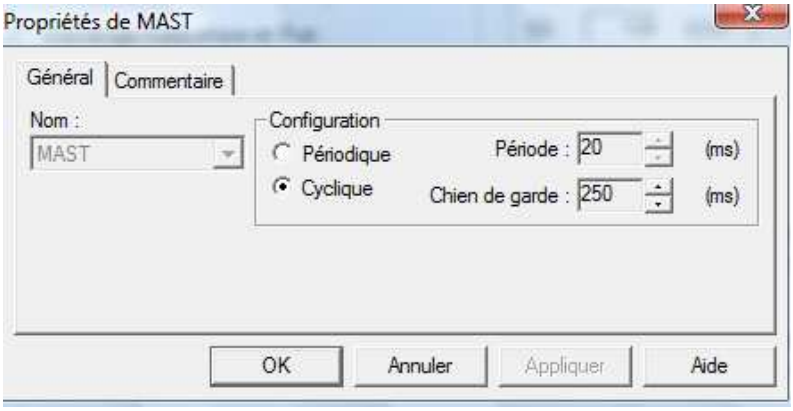


Dans cette exemple ou choisi l'événement numéro 0

15.1. Principe de scrutation d'une tâche



15.2. Configuration d'une tâche



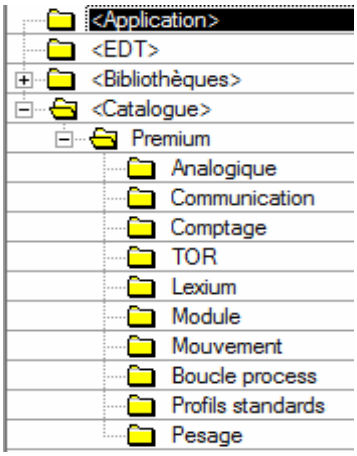
La tâche peut être périodique c'est-à-dire s'exécuter après le temps défini
Ou cyclique (ce n'est valable que pour la tâche maitre), c'est-à-dire s'exécuter dés qu'elle est finie
On peu définir la valeur du chien de garde

La tâche maître doit être défini périodique si vous utiliser une tâche auxiliaire

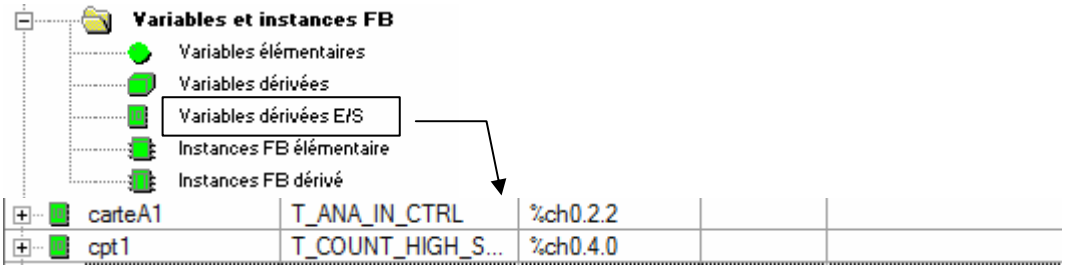
16. LES VARIABLES DERIVEES D'ENTREES/SORTIES

Chaque carte d'entrées sorties de comptage de positionnement de communication, analogique, de pesage, a sa structure spéciale qui donne des informations ou donne des ordres.

On défini ces structures par l'éditeur de variable, le nom de chaque structure est prédéfini, il est accessible par le catalogue ci après.



On défini le nom de la voie à laquelle on s'adresse, on choisi la structure dans le catalogue et on défini l'adresse de la voie sous la forme **%CHrack.module.voie**



L'avantage de ces structures est qu'il donne les adresses et les symboles de tous les paramètres.

Exemple d'une voie de communication sur le rack 1, le module 4 et la voie 0

com1	T_COM_C...	%ch1.4.0		
CH_ERROR	BOOL	%I1.4.0.ERR		Erreur voie
INPUT_SIGNALS	INT	%IW1.4.0.0		Signaux d'entrée
DCD	BOOL	%IW1.4.0.0.0		Détection de porteuse
RI	BOOL	%IW1.4.0.0.1		Indicateur d'appel
CTS	BOOL	%IW1.4.0.0.2		Prêt à émettre
DSR	BOOL	%IW1.4.0.0.3		Poste de données prêt
STOP_EXCH	BOOL	%QW1.4.0.0.0		Arrêt échange sur front montant
EXCH_STS	INT	%MW1.4.0.0		Etat de l'échange
STS_IN_PROGR	BOOL	%MW1.4.0.0.0		Lecture du paramètre d'état en cours
CMD_IN_PROGR	BOOL	%MW1.4.0.0.1		Ecriture du paramètre de commande en cours
ADJ_IN_PROGR	BOOL	%MW1.4.0.0.2		Echange du paramètre de réglage en cours
EXCH_RPT	INT	%MW1.4.0.1		Rapport de la voie
STS_ERR	BOOL	%MW1.4.0.1.0		Erreur lors de la lecture de l'état de la voie
CMD_ERR	BOOL	%MW1.4.0.1.1		Erreur lors de l'émission d'une commande sur la voie
ADJ_ERR	BOOL	%MW1.4.0.1.2		Erreur lors du réglage de la voie
CH_FLT	INT	%MW1.4.0.2		Défauts sur la voie
NO_DEVICE	BOOL	%MW1.4.0.2.0		Aucun équipement ne fonctionne sur la voie
ONE_DEVICE_FLT	BOOL	%MW1.4.0.2.1		Un équipement sur la voie est défectueux
BLK	BOOL	%MW1.4.0.2.2		Défaut externe : bornier
TO_ERR	BOOL	%MW1.4.0.2.3		Erreur timeout (vérification du câblage)

On s'aperçoit qu'il y a des objets **%I** et **%Q** à échange implicite, qui se fait à chaque scrutation et des objets à échange explicite **%MW**, qui se font par programme
Ces structures ne sont pas obligataires, elles sont particulièrement utiles pour des traitements un peu spéciaux sur certaines cartes

16.1. Les échanges explicites

Les échanges implicites avec les cartes d'entrées et de sorties se font à chaque cycle **(%I et %Q**), il n' y à pas bessoin d'instruction pour aller lire les entrées ni pour écrire les sorties
Toutefois, on peu faire des échanges explicites avec les cartes périphérique, c'est-à-dire lire et écrire dans les cartes avec des EF de programme.

READ_PARAM	<EF>
READ_STS	<EF>
READ_TOPO_ADDR	<EF>
RESTORE_PARAM	<EF>
SAVE_PARAM	<EF>
TRF_RECIPE	<EF>
WRITE_CMD	<EF>
WRITE_INPUT_DINT	<EF>
WRITE_INPUT_EBOOL	<EF>
WRITE_INPUT_INT	<EF>
WRITE_INPUT_REAL	<EF>
WRITE_PARAM	<EF>

Lis les paramètres de réglage d'une carte
Lis les états de la carte
Lis les adresses topologiques d'une carte
Restaure les paramètres de réglage d'une carte
Sauve les paramètres de réglage d'une carte
Transférer une recette dans une carte (carte d'AXE par exemple)
Ecrire les commandes d'une carte
Ecrire les entrés d'une carte
Ces fonctions permettent de simuler
Les entrées
A n'utiliser que pour faire une simulation
Ecrit les paramètres d'une carte

Il y a des mots interne %mw qui sont dédiés aux cartes périphériques, ces mots on une adresse qui est:

%MWr.m.c.d
r : numéro du rack
m: numéro du module
c: numéro de la voie
d: numéro du rang (si =0 on peu s'en passer)

%MWr.m.MOD.d
r: numéro du rack
m numéro du module
MOD : écrit en toute lettre, signifie qu'on s'adresse au module lui-même et nom pas à une voie en particulier
d: numéro du rang (si =0 on peu s'en passer)

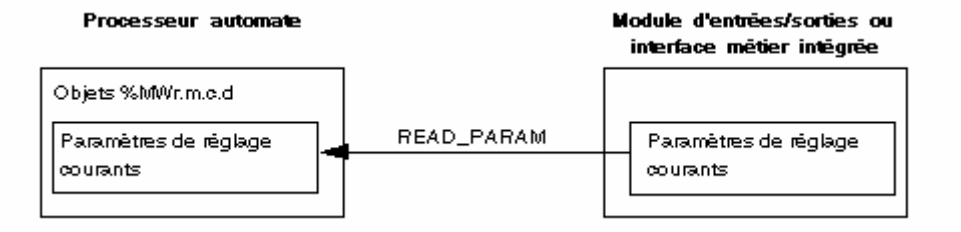
Exemples :

%MW0.2.3.4 Le mot de rang 4 de la voie 3 du module 2 du rack 0

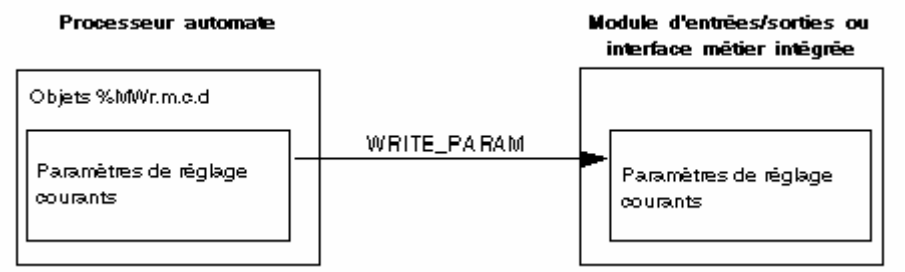
On peu exrtaire un bit d'un mot

%MW0.2.1.0.3 Le bit 3 du mot de rang 0 de la voie 1 du module 2 dans le rack 0

Exemple de lecture des paramétres d'une carte

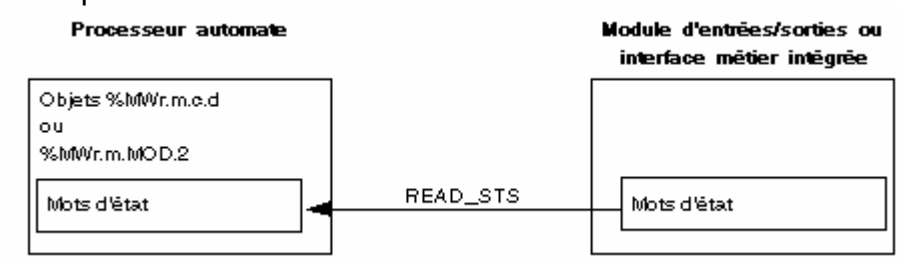


Exemple d'écriture des paramétres d'une carte



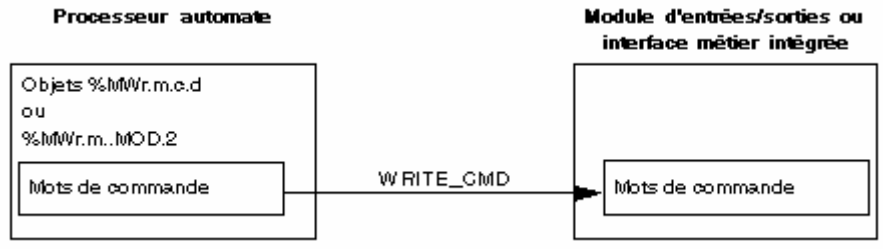
Il faut avoir la documentation de la carte, pour savoir qu'elles sont les différents paramètres.

Exemple de lecture de l'état d'une carte



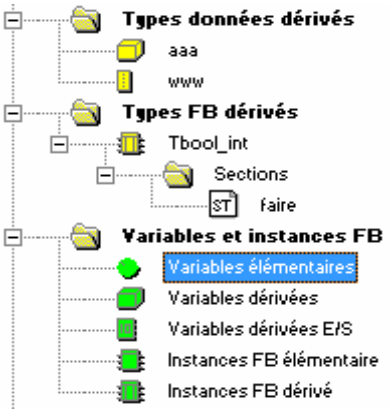
Là aussi, il faut avoir la documentation de la carte pour interpréter les différents états

Exemple d'éciture des états d'une carte



17. SYNTHESE D'ACCES AUX VARIABLES

17.1. Vue d'ensemble



On défini les types utilisateurs STRUCT ou ARRAY, bien que les types ARRAY sont rarement définis

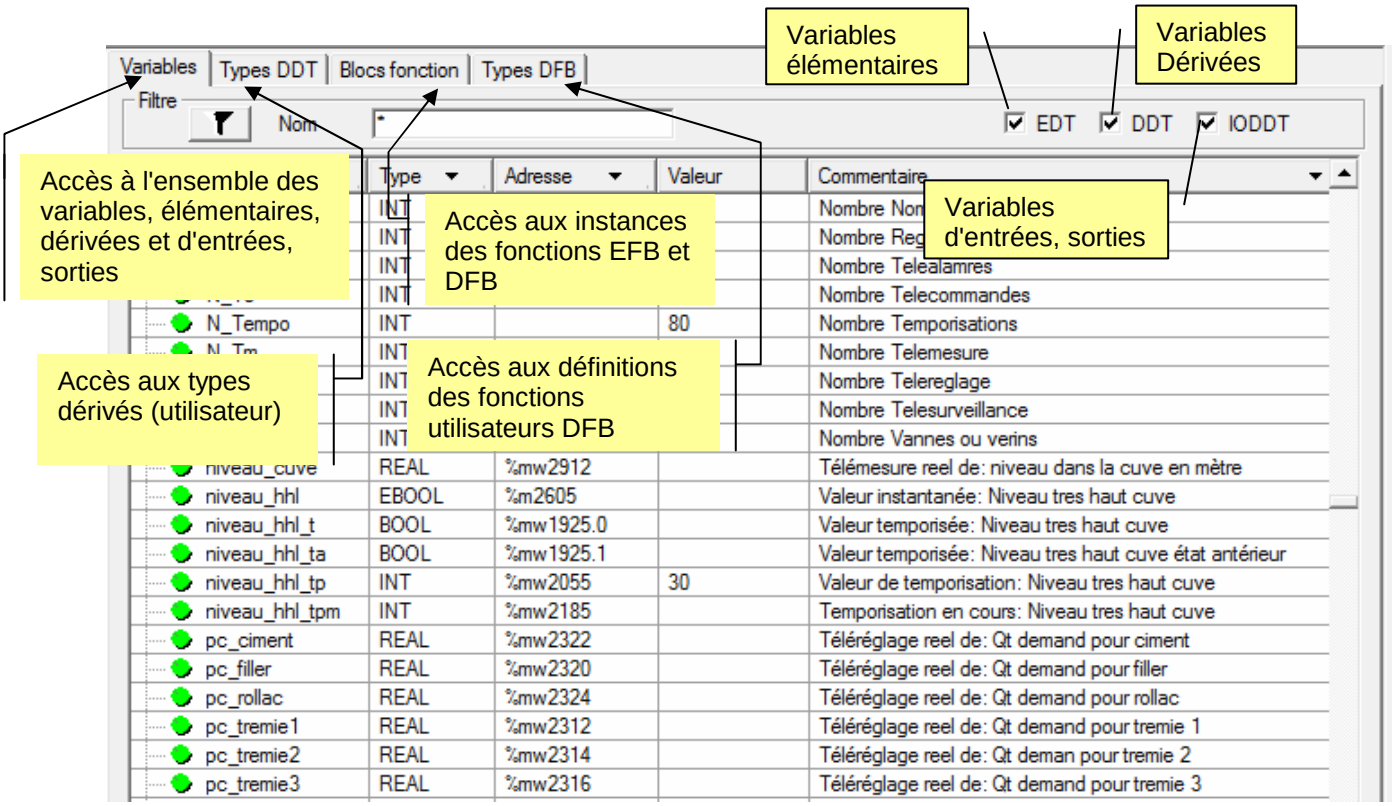
On défini les fonctions utilisateur DFB

Accès global aux types, variables et instances

On défini l'ensemble des variables de type prédéfini (bool, int, real, ARRAY)
On instancie les variables dont le type est défini par l'utilisateur (dérivés)
On défini ici les variable des cartes d'entrée, sorties
On instancie les fonctions de type prédéfini (EFB)
On instancie les fonctions de type défini par l'utilisateur (DFB)

17.2. Accès rapide à l'ensemble des types, variables et instances

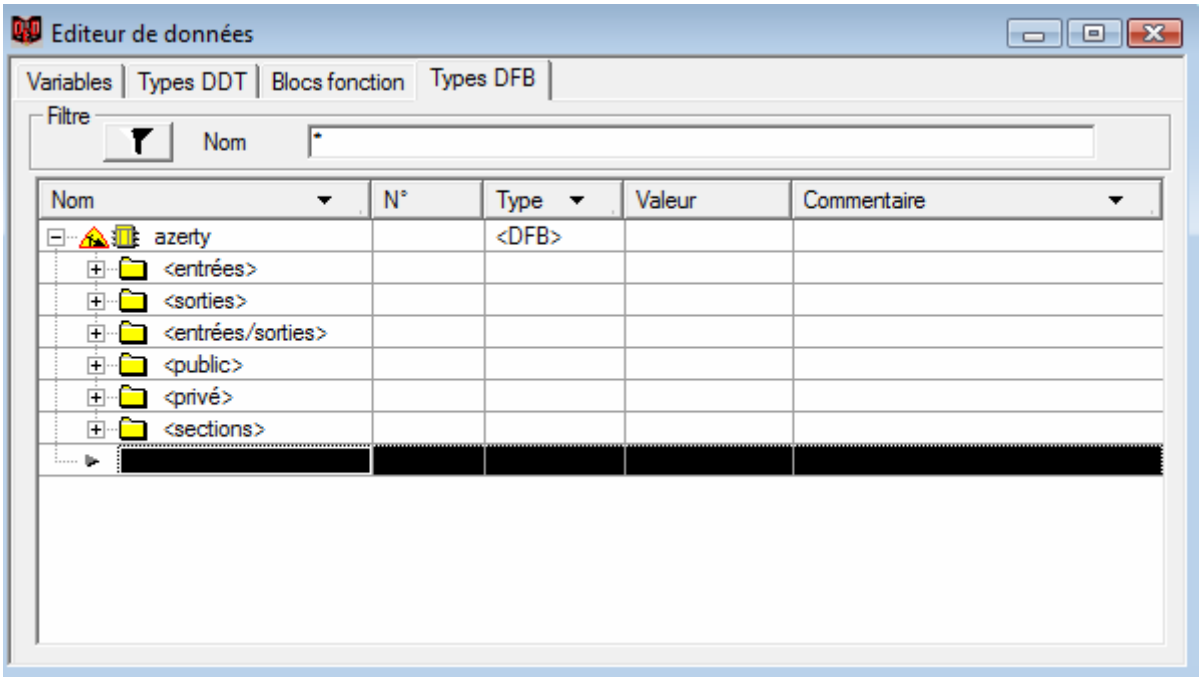
Il suffit de double cliquer ou ouvrir "**Variables et instances FB**"
L'ensemble des types, des variables et des instances sont accéssible ICI, il suffit d'aller dans le bon onglet et cocher les bonnes cases



18. LES FONCTIONS DERIVEES DE TYPE DFB

Ce sont des fonctions écritent par l'utilisateur, à ces fonctions on peut faire passer des paramétres d'entrées (qui ne peuvent être que lu), des paramétres de sortie (qui ne peuvent être qu'écit) et des paramétres d'entrée/sortie (qui peuvent être lu et écrit)

En plus il y a des paramétres qui ne sont pas passés lors de l'appel de la fonction, mais qui peuvent être utile pour la fonction elle-même. Il y a deux type de paramètres, les privées qui ne sont accessibles que par la fonction elle-même et les publics qui sont accessible par la fonction, mais aussi par l'appplication.

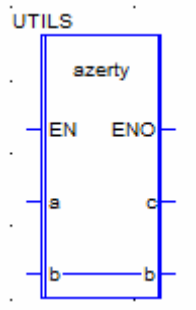


Ces fonctions sont écritent dans des sections qui leurs sont affectés

On accéde à ces section depuis le navigateur, le langage de ces sections peut être ST, IL, LD ou FBD, mais ne peuvent pas être en grafcet.

→Les fonctions DFB sont très utiles, en plus vous pouvez les exporter dans une bibliothéque et les rendre réutilisable.

Exemple d'une fonction DFB



Cette fonction est du type "azerty" et est instancié sous le noms "UTILS", on peut rappeler cette fonction avec cette instance ou la rappeler avec une autre instance. Il est utile de créer une nouvelle instance lorsque vous avez besoins de mémoriser les paramétres d'une scrutation sur l'autre, sinon utilisez la même instance.

Vous avez accés dans l'application appelelante aux diffrérents paramétres de sorties, et public sous la forme :
<nom de l'instance>.<nom du paramétre>

Exemple:
UTILS.c donne accés au paramètre de sortie c
UTILS.d donne accés au paramètre public d (en lecture ou écriture)

19. LES EFB (voir chapitre 7)

Se sont des fonctions équivalentes au DFB, qui sont écritent en langage C qui necessite d'avoir le supplément "SDKC"

20. LES PRINCIPAUX BIT SYSTEMES

Bits de reprises secteurs

Bit Symbole	Fonction	Description	Etat initial	Quantum	Premium Atrium
%S0 COLDSTART	Démarrage à froid	Normalement à l'état 0 est mis à l'état 1 par : • reprise secteur avec perte des données (défaut de batterie), • programme utilisateur, • terminal, • changement de cartouche. • chargement d'un programme. Ce bit est mis à 1 durant le premier cycle complet de reprise que l'automate soit en RUN ou en STOP. Il est remis à 0 par le système avant le cycle suivant.	1 (1 cycle)	OUI	OUI
%S1 WARMSTART	Reprise à chaud	Normalement à l'état 0, est mis à l'état 1 par : • reprise secteur avec sauvegarde des données, • programme utilisateur, • terminal, • action sur le changement de cartouche. Il est remis à 0 par le système à la fin du premier cycle complet et avant la mise à jour des sorties.	0	OUI	OUI

Bits bases de temps

%S4 TB10MS	Base de temps 10 ms	Bit dont le changement d'état est cadencé par une horloge interne. Il est asynchrone par rapport au cycle de l'automate. Diagramme : 	-	OUI	OUI
%S5 TB100MS	Base de temps 100 ms	Idem %S4	-	OUI	OUI
%S6 TB1SEC	Base de temps 1 s	Idem %S4	-	OUI	OUI
%S7 TB1MIN	Base de temps 1 min	Idem %S4	-	OUI	OUI

Bits d'état de l'automate

%S10 IOERR	Défaut d'entrées/sorties	Normalement à l'état 1, est mis à l'état 0 quand un défaut d'E/S d'un module en rack ou d'un équipement sur Fipio est détecté (configuration non conforme, défaut d'échange, défaut matériel...). Le bit %S10 est remis à 1 par le système dès la disparition du défaut.	1	OUI	OUI
%S11 WDG	Débordement du chien de garde	Normalement à l'état 0, est mis à l'état 1 par le système dès que le temps d'exécution d'une tâche devient supérieur au temps d'exécution maximum (chien de garde) déclaré dans les propriétés de la tâche.	0	OUI	OUI
%S12 PLCRUNNING	Automate en RUN	Ce bit est mis à l'état 1 par le système lorsque l'automate est en RUN. Il est mis à 0 par le système dès que l'automate n'est plus en RUN (état STOP, INIT...).	0	OUI	OUI

Bits de débordements

%S17 CARRY	Sortie décalage circulaire	Normalement à l'état 0. Lors d'une opération de décalage circulaire, il prend l'état du bit sortant.	0	OUI	OUI
%S18 OVERFLOW	Débordement ou erreur arithmétique	Normalement à l'état 0, est mis à l'état 1 en cas de débordement de capacité si : • résultat supérieur à + 32 767 ou inférieur à - 32 768, en simple longueur, • résultat supérieur à + 65 535 , en entier non signé, • résultat supérieur à + 2 147 483 647 ou inférieur à - 2 147 483 648, en double longueur, • résultat supérieur à +4 294 967 296, en entier double longueur non signé, • valeurs réelles hors bornes, • division par 0, • racine d'un nombre négatif, • forçage à un pas inexistant sur un programmateur cyclique, • empilage d'un registre plein, dépilage d'un registre vide. Doit être testé par le programme utilisateur, après chaque opération où il y a risque de débordement puis remis à 0 par l'utilisateur en cas de débordement. Lorsque le bit %S18 passe à 1, l'application s'arrête en erreur si le bit %S78 a été positionné à 1.	0	OUI	OUI

Bits d'activation des tâches

%S30 MASTACT	Activation/désactivation tâche maitre	Normalement à l'état 1, la mise à l'état 0 par l'utilisateur provoque la désactivation de la tâche maître. Ce bit est pris en considération par le système à la fin de chaque cycle de la tâche MAST.	1	OUI	OUI
%S31 FASTACT	Activation/désactivation tâche rapide	Normalement à l'état 1 lorsque la tâche est créée par l'utilisateur. La mise à zéro par l'utilisateur provoque la désactivation de la tâche.	0	OUI	OUI
%S32 AUX0ACT	Activation/désactivation tâche auxiliaire 0	Normalement à l'état 1 lorsque la tâche est créée par l'utilisateur. La mise à zéro par l'utilisateur provoque la désactivation de la tâche auxiliaire.	0	OUI	OUI
%S33 AUX1ACT	Activation/désactivation tâche auxiliaire 1	Normalement à l'état 1 lorsque la tâche est créée par l'utilisateur. La mise à zéro par l'utilisateur provoque la désactivation de la tâche auxiliaire.	0	OUI	OUI
%S34 AUX2ACT	Activation/désactivation tâche auxiliaire 2	Normalement à l'état 1 lorsque la tâche est créée par l'utilisateur. La mise à zéro par l'utilisateur provoque la désactivation de la tâche auxiliaire.	0	OUI	OUI
%S35 AUX3ACT	Activation/désactivation tâche auxiliaire 3	Normalement à l'état 1 lorsque la tâche est créée par l'utilisateur. La mise à zéro par l'utilisateur provoque la désactivation de la tâche auxiliaire..	0	OUI	OUI
%S38 ACTIVEVT	Validation/inhibition des événements	Normalement à l'état 1, la mise à l'état 0 par l'utilisateur provoque l'inhibition des événements.	1	OUI	OUI
%S39 EVTOVR	Saturation dans le traitement des événements	Ce bit est mis à 1 par le système pour indiquer qu'un ou plusieurs événements ne peuvent pas être traités par suite de saturation des files d'attente. Ce bit est remis à l'état 0 par l'utilisateur.	0	OUI	OUI

21. QUELQUES MOTS SYSTEMES

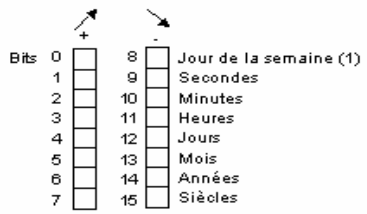
Mots de durée

%SW18 **%SW19** 100MSCOUNTER	Compteur de temps absolu	Les mots %SW18 et %SW19 permettent d'effectuer des calculs de durée. Ils sont incrémentés toute les 1/10ème de secondes par le système (même automate en STOP, ils ne sont plus incrémentés si l'automate est hors tension). Ils peuvent être lus et écrit par programme utilisateur ou par terminal.	0	OUI	OUI

Mots horodateurs

%SW49
DAYOFWEEK
%SW50
SEC
%SW51
HOURMIN
%SW52
MONTHDAY
%SW53
YEAR
	Fonction Horodateur	Mots système contenant la date et l'heure courante (en BCD) :
		• %SW49 : jour de la semaine
		  ○ 1 = Lundi
		  ○ 2 = Mardi
		  ○ 3 = Mercredi
		  ○ 4 = Jeudi
		  ○ 5 = Vendredi
		  ○ 6 = Samedi
		  ○ 7 = Dimanche
		• %SW50 : Secondes (16#SS00).
		• %SW51 : Heures et Minutes (16#HHMM).
		• %SW52 : Mois et Jour (16#MMJJ).
		• %SW53 : Année (16#AAAA).
		Ces mots sont gérés par le système lorsque le bit %S50 est à 0.
		Ces mots peuvent être écrits par le programme utilisateur ou par le terminal lorsque le bit %S50 est mis à 1.
			-	OUI	OUI

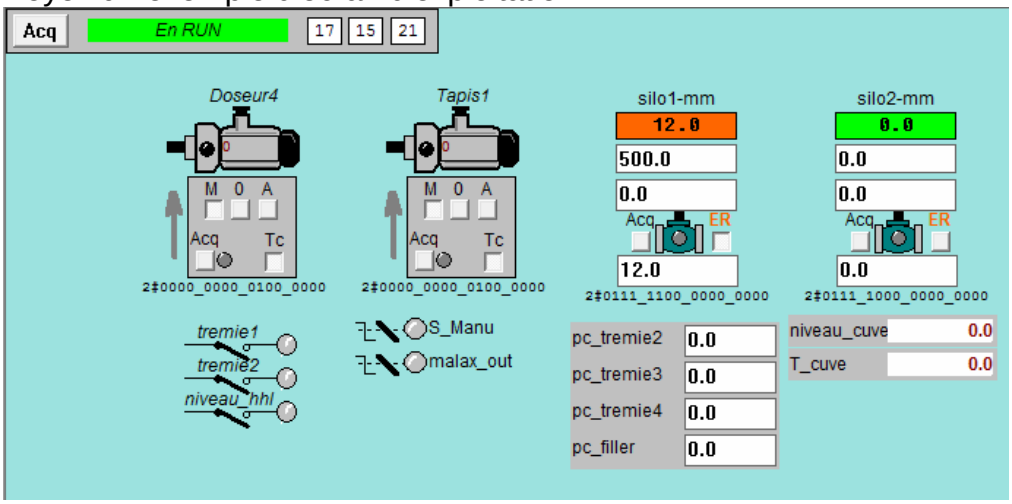
%SW54
STOPSEC
%SW55
STOPHM
%SW56
STOPMD
%SW57
STOPYEAR
%SW58
STOPDAY
	Fonction Horodateur sur dernier arrêt	Mots système contenant la date et l'heure du dernier défaut secteur ou arrêt automate (en BCD) :
		• %SW54 : Secondes (00SS).
		• %SW55 : Heures et Minutes (HHMM).
		• %SW56 : Mois et Jour (MMJJ).
		• %SW57 : Année (AAAA).
		• %SW58 octet de poids fort contient le jour de la semaine (1 pour Lundi à 7 pour Dimanche).
		• %SW58 octet de poids faible contient le code du dernier arrêt.
		  ○ 1= passage de RUN en STOP par le terminal ou l'entrée dédiée
		  ○ 2=arrêt sur défaut logiciel (débordement de tâche automate ou du SFC)
		  ○ 4=coupure secteur ou manipulation de la carte mémoire
		  ○ 5=arrêt sur défaut matériel
		  ○ 6=arrêt sur instruction HALT.
			-	OUI	OUI

%SW59 ADJDATETIME	Réglage de la date courante	Contient deux séries de 8 bits pour régler la date courante. L'action est toujours réalisée sur front montant du bit. Ce mot est validé par le bit %S59. Illustration :  	0	OUI	OUI

22. LES ECRANS D'EXPLOITATION

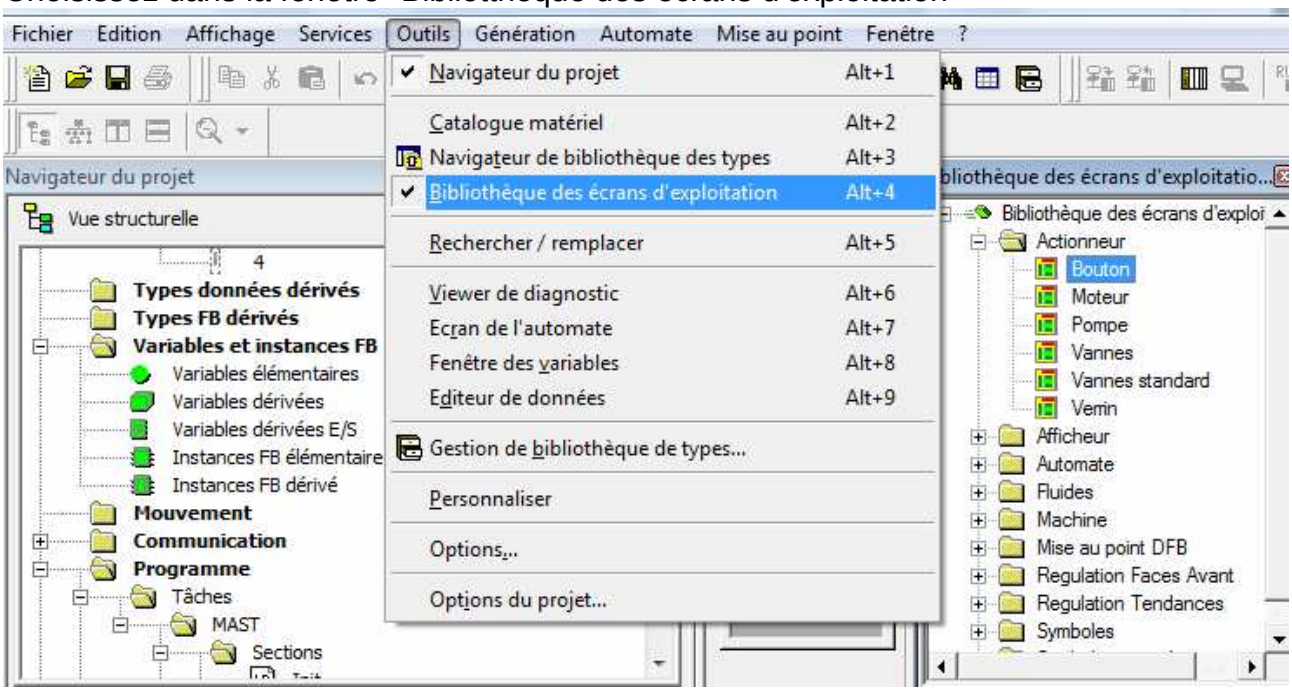
Les écrans d'exploitation sont très utiles lors des mises aux points mais aussi en maintenance, ils remplacent avantageusement les superviseurs, très facile à utiliser, on peut définir toutes sortes de formes, de textes, de boutons, les animer très facilements…
On peut aussi utiliser les objets de "outils" "bibliothéque des écrans d'exploitations"

Voyez un exemple d'écran d'exploitation

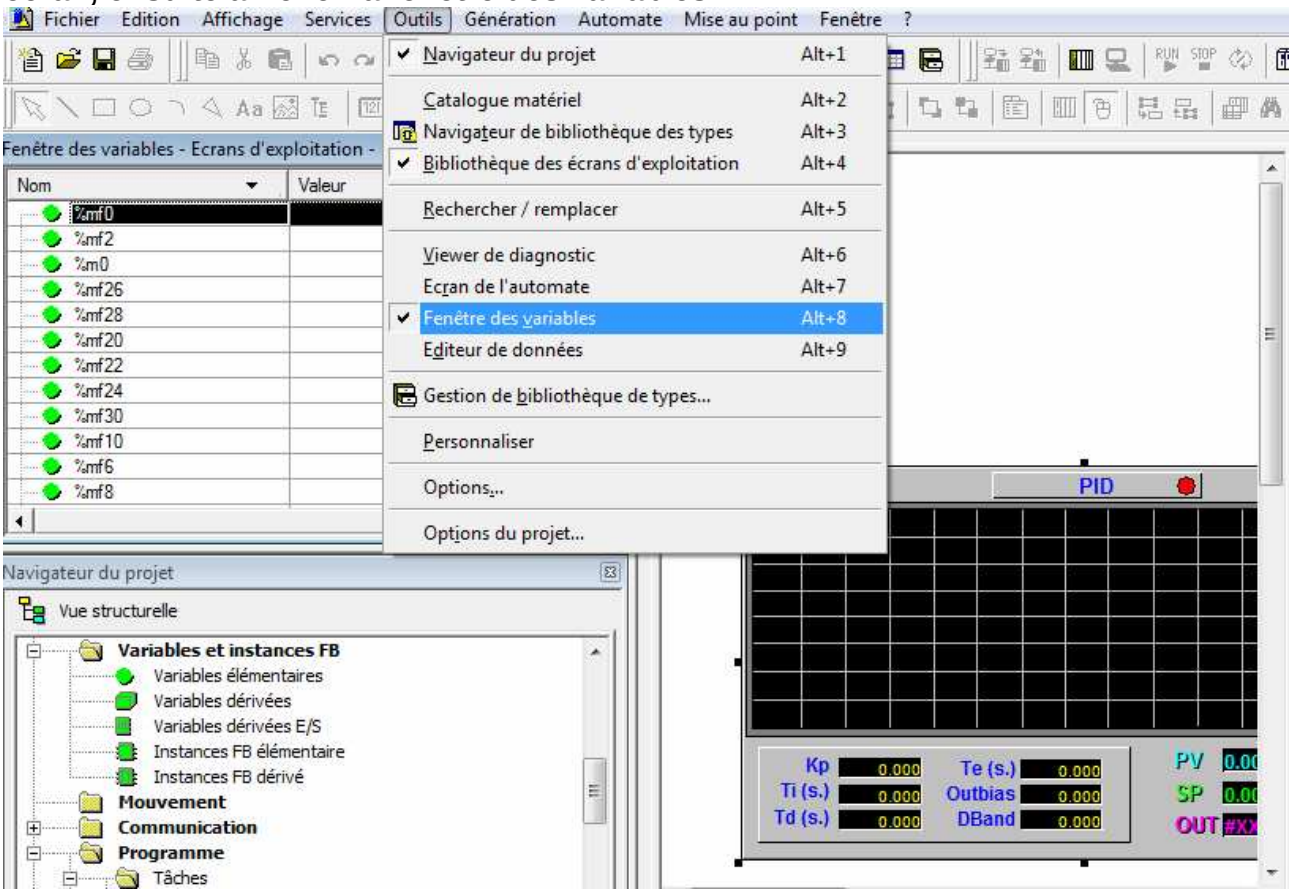


22.1. *Utiliser la bibliothéque d'objet prédéfinis*

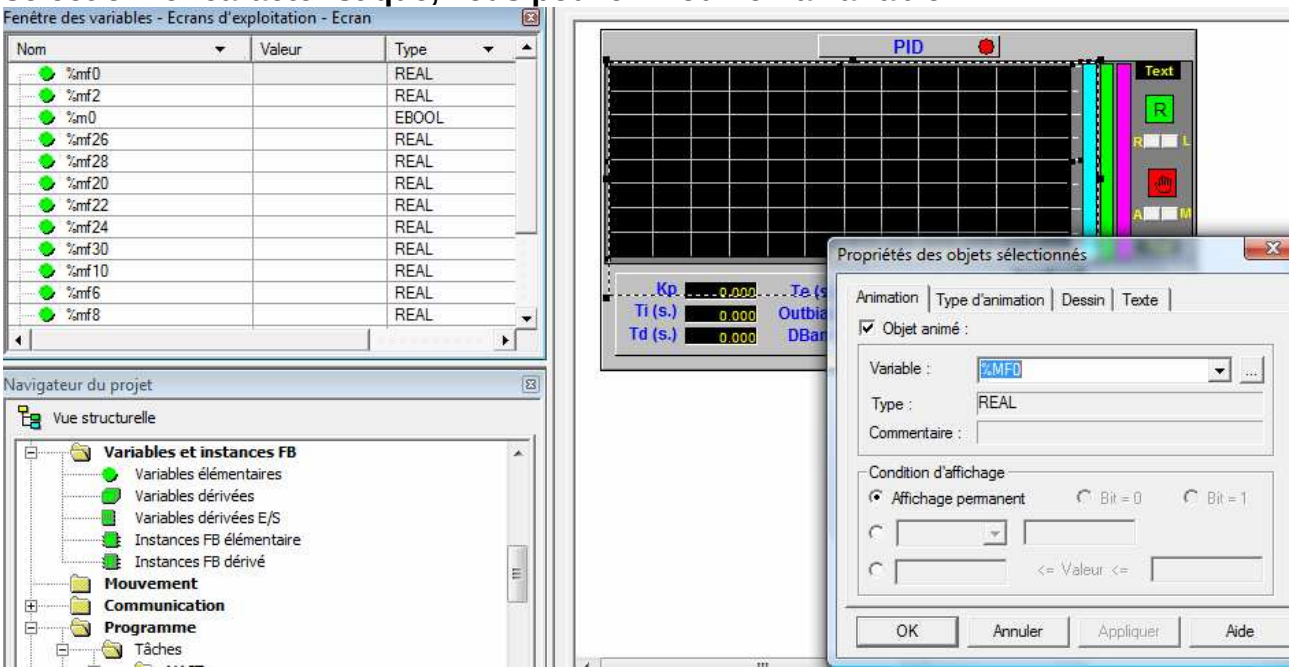
Choisissez dans la fenêtre "Bibliotthèque des écrans d'exploitation"



Sélectionner un objet, par exemple une vue d'un PID, copier le dans un nouvel écran, ensuite afficher la fenêtre des variables



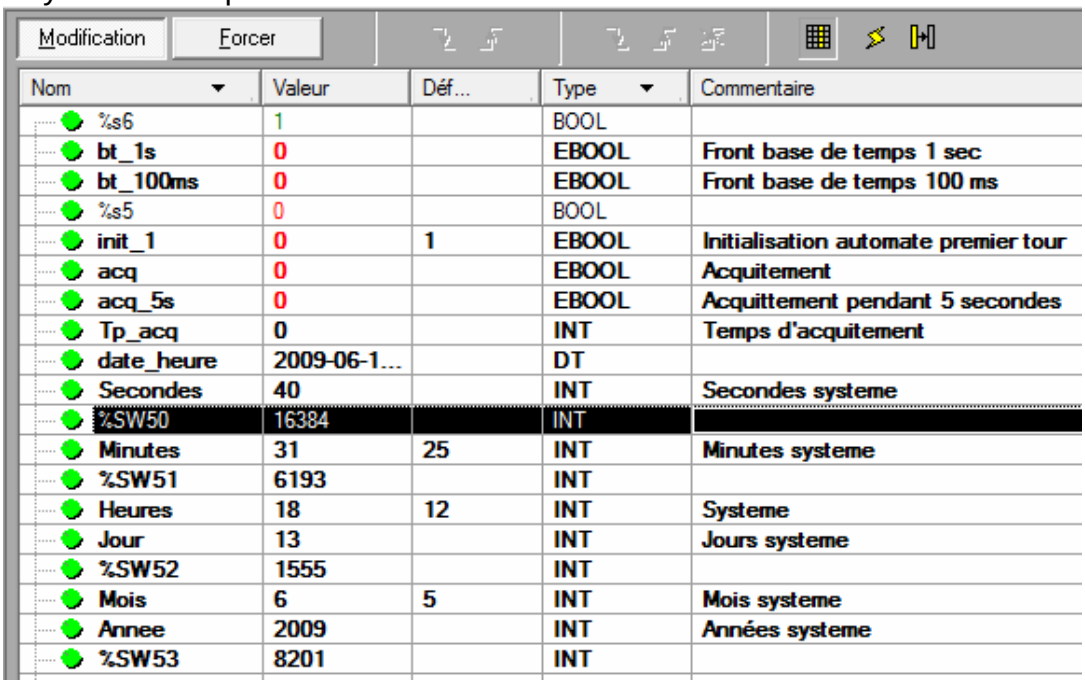
Double cliquer sur la variable que vous désirer modifier, par exemple %MF0, vous avez les objet sélectionné qui aparaissent, cliquer alors avec le bouton droite et selectionner caractéristique, vous pouvez modifier la variable.



23. LES TABLES D'ANIMATIONS

Voilà avec les écrans d'exploitations les deux outils indispensables pour faire des réglages, on peut enregistrer autant de tables d'animations que necessaire.

Voyez un exemple de table d'animation



Nom	Valeur	Déf...	Type	Commentaire
%s6	1		BOOL	
bt_1s	0		EBOOL	Front base de temps 1 sec
bt_100ms	0		EBOOL	Front base de temps 100 ms
%s5	0		BOOL	
init_1	0	1	EBOOL	Initialisation automate premier tour
acq	0		EBOOL	Acquitement
acq_5s	0		EBOOL	Acquittement pendant 5 secondes
Tp_acq	0		INT	Temps d'acquitement
date_heure	2009-06-1...		DT	
Secondes	40		INT	Secondes systeme
%SW50	16384		INT	
Minutes	31	25	INT	Minutes systeme
%SW51	6193		INT	
Heures	18	12	INT	Systeme
Jour	13		INT	Jours systeme
%SW52	1555		INT	
Mois	6	5	INT	Mois systeme
Annee	2009		INT	Années systeme
%SW53	8201		INT	