***Automate programmable***

# Fonction de calcul

## Création d’une fonction F

Créer la fonction de calcul suivante

Nom : fcCalculCoteOpose

Variable (var\_input) : lrAngle, lrHypotenuse

Résultat(fcCalculCoteOpose :=)  lrHypotenuse\*[sin](http://infosys.beckhoff.com/content/1033/tc3_plc_intro/136424075.html?id=15276)(lrAngle)

## Utilisation de la fonction

Tester le fonctionnement de votre fonction, déclaration d’une variable global pour recevoir le résultat

## Idem sous forme d’un programme P

Refaire l’exercice à l’aide d’un programme en lieu et place d’une fonction (utilisation de var\_output)

# Détection de flanc

## Programme P

Réaliser la programmation d’un détecteur de flanc montant à l’aide d’un programme pR\_Trig, voir la description de la version officielle [ici](http://infosys.beckhoff.com/content/1033/tcplclib_tc2_standard/74391563.html?id=15792).

Réaliser un programme de test qui inverse une variable xOut1 à chaque flanc du changement d’une variable globale xTouche

## Réutilisation (deuxième instance)

Réaliser une deuxième gestion de flanc pour xOut2 et xTouche2

Faire le test et expliquer ce qui ne va pas

## Réalisation d’un bloc de fonction FB

Refaire la fonction sous la forme d’un bloc de fonction FB et faire le test avec les deux utilisations.

Constatation

## Utilisation d’un Timer TON

Réaliser l’exercice de la variable qui s’inverse tous les 1seconde avec un timer

[**TON**](http://infosys.beckhoff.com/content/1033/tcplclib_tc2_standard/74406539.html?id=15802)**.**

# HMI (Human-Machine Interface), IHM interface homme-machine, MMI (Mensch-Maschine-Interface)

## Variable de l’exercice 2.4

Afficher la variable de xBool1 de l’exercice



## Variable de l’exercice 2.3

Réaliser une visualisation qui affiche

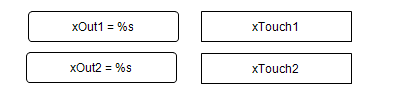
xOut1 et xOut2, la couleur change si la valeur est vrai, le texte indique la valeur (%s)

L’appui sur xTouch1 active xTouche1 tant que l’on presse la touche

L’appui sur xTouch2 inverse la variable xTocuhe2

La couleur de xTouche1 change si xTouche1=vrai

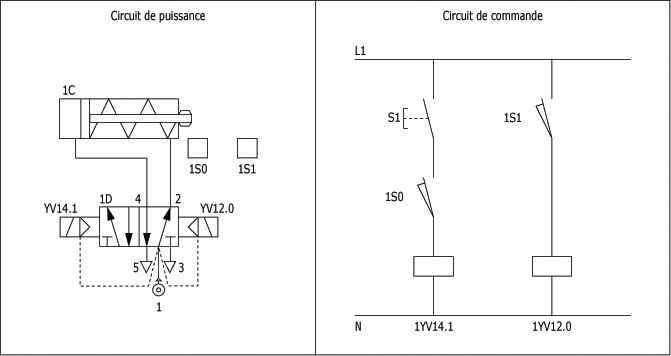
La couleur de xTouche2 change si xTouche1=vrai



# Exercice

## Du grafcet au TS

Réaliser le grafcet du pilotage de ce système, le vérin exécute continuellement un va et vient.



Les capteurs 1S0 et 1S1 sont simulés (IxFdcRentre et IxFdcSort)

Le pilotage se fait avec YV14.1 et YV12.0 (QxSort et QxRentre)

Réaliser le grafcet

Programmer le grafcet en TS avec la fonction case et la variable d’étape (état) « iStep :int ; »

## Séquence automatique temporisée

A l’aide de l’instruction « [case](http://infosys.beckhoff.com/content/1033/tcplccontrol/html/tcplcctrl_statement_case.htm?id=23155)» pour la variable d’étape (état) « iStep :int ; »

Réaliser une machine d’état pour réaliser les fonctions suivantes :

Le cycle tourne en boucle

L’étape 100 (iStep) doit durer 1s

L’étape 200 doit durer 0.5s

xLampe1 est vrai si l’étape=100

xLampe2 est vrai si l’étape=200

Une pause de 0.5s à l’étape 300

Réaliser le grafcet

Réaliser deux programmes selon les options suivantes

* Option 1

Réaliser le changement d’état dans l’étape, utilisation du TON également.

* Option 2

Réaliser le changement d’état en dehors de la séquence, après celle-ci.

Afficher dans une visualisation

* Les lampes
* La valeur de l’étape

Consigne :

* 1 seul TON autorisé

## Modification 1

On souhaite que la variable xLampe1 soit vrai iStep>=100 et iStep<=200

Modifier les deux programmes

* Option 1

Réaliser le changement d’état dans l’étape, utilisation du TON également.

* Option 2

Réaliser le changement d’état en dehors de la séquence, après celle-ci.

## Modification 2

On souhaite obtenir le diagramme suivant

xLampe1 est vrai durant 2s à partir de l’étape 100

xLampe2 est vrai après 0.5s de l’étape 100 et a une durée de 0.7s

Consigne :

* Utiliser deux timer TON

Réaliser le grafcet

Réaliser le programme

## Analyse

Remplir le tableau suivant

|  |  |  |
| --- | --- | --- |
| **Critères** | **Option 1** | **Option 2** |
| Utilisation mémoire |  |  |
| Ressource (charge de la CPU) |  |  |
| Reprise du code (modification) |  |  |
| Vitesse de réalisation |  |  |
| Utilisation sur plusieurs étapes |  |  |
|  |  |  |

## Option 1 de pilotage par activation du cycle

A l’aide la variable xEnable, le cycle est en fonctionnement normal, sinon l’étape doit être à 0

## Option 2 de pilotage par start, stop, reset

Création d’un bloc fonctionnel comportant la séquence automatique du point 3.1.

Un Flanc sur xReset : la variable iStep passe à 0

Un Flanc sur xStart : transition étape 0, mode de fonctionnement continu

Un Flanc sur xStop : le cycle se termine normalement

La sortie xBusy est à vrai si le cycle est actif

xBusy :bool ;

FB\_CycleAuto

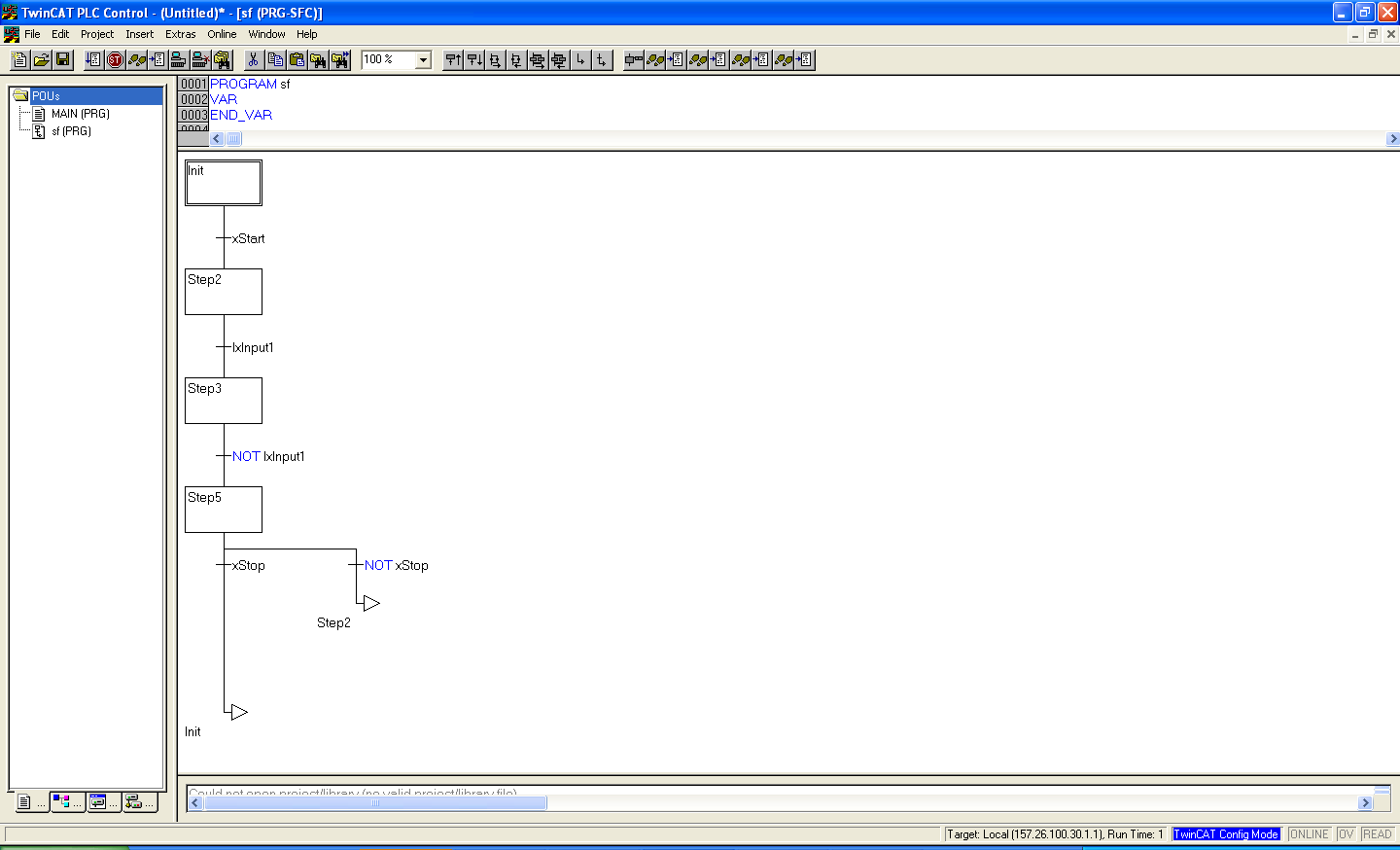
xStart :bool ;

xReset :bool ;

xStop :bool ;

# Configuration et liaison I/O

Réaliser le programme suivant et faire un test avec le coupleur EK1100



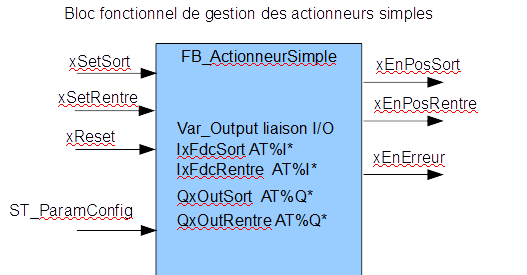
QxOutput2=vrai

QxOutput1=vrai

# Exercice

## Base du FB\_ActionneurSimple

Pour la gestion des actionneurs à deux positions, réaliser le bloc fonctionnel FB\_ActionneurSimple selon les consignes suivantes.



**Configuration :**

ST\_ParamActionneur :

tTimeOut : time ; //temps d’enveloppe maximale avant activation xEnErreur

**Fonctionnement :**

si l’entrée xReset=1

Les sorties QxOutSort et QxOutRentre sont forcées à 0

xEnErreur est initialisé à false

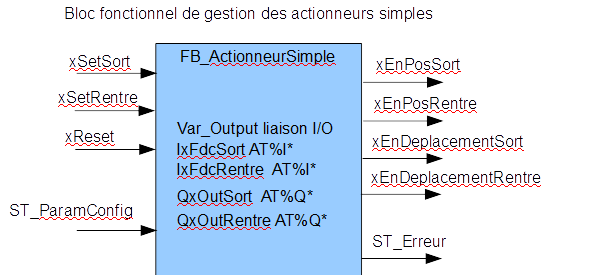
Un flanc montant sur xSetSort ou xSetRentre active la sortie respective QxOutSort ou QxOutRentre et attend le fin de course IxFdcPlus ou IxFdcMoins.

Les sorties du bloc de fonction xEnPosSort et xEnposRentre indiquent que le vérin est en position correct (QxOutPlus=1 et IxFdcPlus=1 ou QxOutRentre=1 et IxFdcRentre=1)

Si le fin de course n’est pas atteint avant le temps indiqué par tTimeOut , la sortie xEnErreur est active, cette erreur n’est pas fatal, si le fin de course revient la sortie xEnPosSort/Rentre devient vrai.

1. **Réaliser la programmation (langage selon la norme CEI-1131-3) interne pour ce bloc de fonction en langage ST**
2. **Test en mode online avec l’automate**

## Optimisation du bloc de fonction FB\_ActionneurSimple



Optimiser le FB\_ActionneurSimple avec les fonctionnalités suivantes :

ST\_ ParamActionneur :

**xAvecFDCSort** :bool ; //Activation de la présence d'un fin de course sort

**xAvecFDCRentre** :bool ; //Activation de la présence d'un fin de course rentre

**tTimeOut** : time ; //temps d’enveloppe avant activation xEnErreur

**tEnvStabSort** :time ; //temps d'enveloppe si xAvecFDCSort=0, temps de stabilisation si xAvecFDCSort=1

**tEnvStabRentre** :time ; //temps d'enveloppe si xAvecFDCSort=0, temps de stabilisation si xAvecFDCSort=1

Entrée de libération du déplacement :

**xAutoriseSort** :bool ; //Autorisation de déplacement sort

**xAutoriseRentre** :bool ; //Autorisation de déplacement rentre

Sortie complémentaire :

**ST\_Erreur**

xEnErreur :bool ; //vrai=> erreur active

iIdErreur : int ; //numéro de l’erreur active

sMessage : string ;//message d’erreur

**xEnDeplacementSort**:BOOL;//indique qu'il se déplace en position SORT

**xEnDeplacementRentre**:BOOL;//indique qu'il se déplace en position RENTRE

Liste des erreurs possibles :

* + Si un ordre de déplacement sort est donné et que l’entrée, xAutoriseSort n’est pas vrai, on indique une erreur
  + Idem pour xAutoriseRentre
  + Erreur de timeout sort
  + Erreur de timeout rentre
  + Les deux fins de course sont vraies en même temps
  + Double consignes, xSetSort=1 et xSetRentre=1

[Exemple gestion erreur pour MC\_MoveAbsolute](http://infosys.beckhoff.com/content/1040/tcplclibmc2/html/blocks/tcplclibmc_moveabsolute.htm?id=13501)

## Tester le FB\_ ActionneurSimple sur platine Festo

Réaliser le test du FB\_ActionneurSimple sur les platines Festo

# Exercice portique

**Utilisation d’un portique axe X et Y et un préhenseur (2 FDC par vérin)**



Fonctionnalité :

Système de portique pour transporter une pièce de A à B (en évitant l’obstacle...)

Si une pièce est présente en A, le système prends une pièce en positon A et la dépose en B.

Deux capteurs inductifs en A et B permettent de détecter si une pièce est présente.

Un capteur optique dans le préhenseur (Pince) permet de détecter la présence d’une pièce.

Matériel:

* Un pince (ouverte/ fermée) commandé par une vanne 5/2 monostable avec
* Un détecteur pince ouverte
* Un détecteur pince fermée
* Un vérin pneumatique vertical commandé par une vanne 5/2 monostable avec
* Un détecteur position haute,
* Un détecteur position basse
* Un vérin pneumatique horizontal commandé par une vanne 5/2 monostable avec
* Un détecteur position gauche
* Un détecteur position droite

## Cycle automatique

Réaliser le cycle automatique pour ce portique

Réaliser un HMI de simulation

## Initialisation

Réaliser le cycle d’initialisation pour ce portique

## Réaliser un mode pas à pas

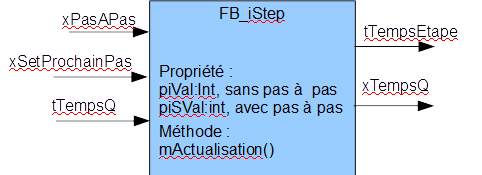
Ajouter un mode pas à pas aux deux cycles précédents.

L’astuce est d’agir sur la transition d’étape, qui est l’équivalent de l’écriture de la variable iStep :=xxxx.

Grâce à l’arrivé des objets dans la norme IEC 61131, nous pouvons maintenant utiliser une propriété.

On en profite également pour lui ajouter une fonctionnalité de gestion du temps d’étape avec activation d’une sortie si un temps donné est atteint.

Sur cette base, réaliser un objet fbiStep qui répond aux exigences suivantes :



Propriété :

piVal : valeur d’étape, fonction set sans surveillance de xPasApas

piSVal : valeur d’étape, fonction set avec surveillance de xPasAPas

Var\_Input :

xPasAPas:BOOL:=FALSE;//activation du mode pas à pas

xSetProchainPas:BOOL;//va au prochain pas sur flanc haut

tTempsQ :TIME;//temps pour l’activation de la sortie xTempsQ

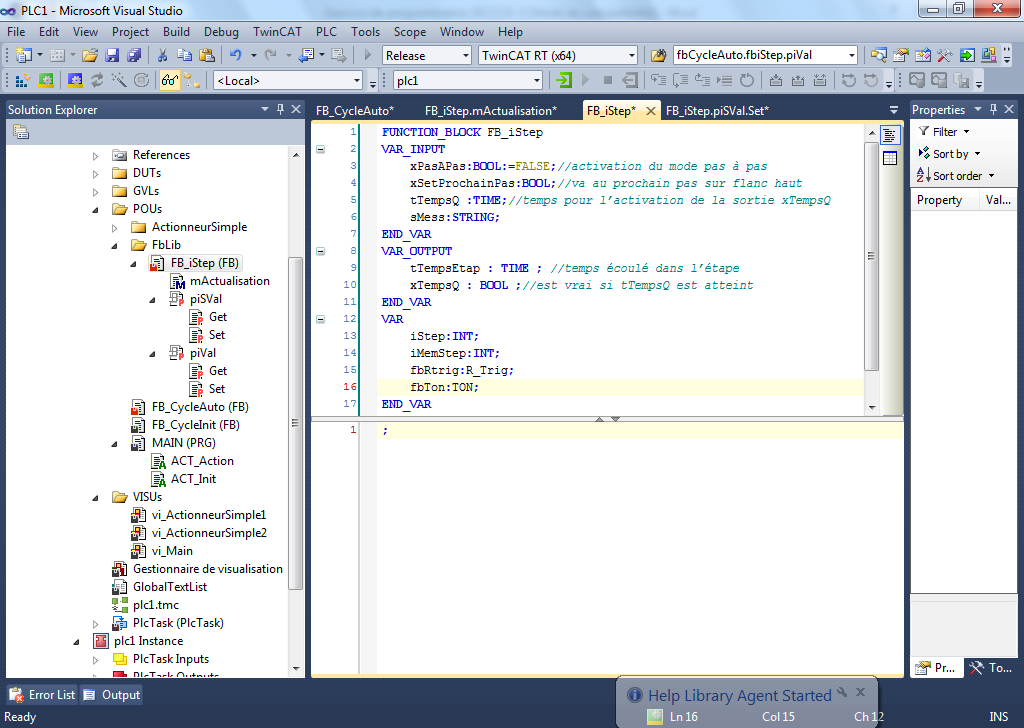
Var\_output :

tTempsEtap : time ; //temps écoulé dans l’étape

xTempsQ : bool ;//est vrai si tTempsQ est atteint

Méthode :

mActualisation() // actualisation des sorties tTempsEtap et xTempsQ et surveillance du trig pour activer le prochain pas



Dans Fb\_CycleAuto, on a :

VAR

fbiStep:FB\_iStep;

début de programme

fbiStep.mActualisation();

à chaque étape

fbiStep.piSVal:=xxx; si étape avec arrêt pas à pas

fbiStep.piVal:=xxx; sans arrêt pas à pas

fin du programme

iStep:=fbiStep.piVal;

Programmer cet objet et réaliser un test dans les cycles précédents.

## Ajouter une temporisation dans un cycle

Grâce à l’objet FB\_iStep, on peut maintenant utiliser des temporisations sans faire de déclaration dans le cycle.

Exemple :

//étape avant la surveillance du temps

100 : fbiStep. tTempsQ :=t#1s ;

FbiStep.piVal :=110 ;

//attendre la fin de la temporisation

110 : if fbiStep.xTempsQ then

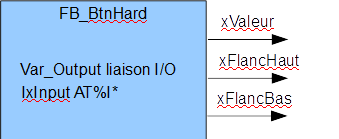
FbiStep.piSVal :=120 ;

End\_if

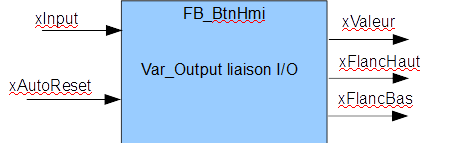
Réaliser un test dans les cycles précédents

# Réalisation objet Touche hard et soft

## Touche hardware (depuis une entrée digitale)

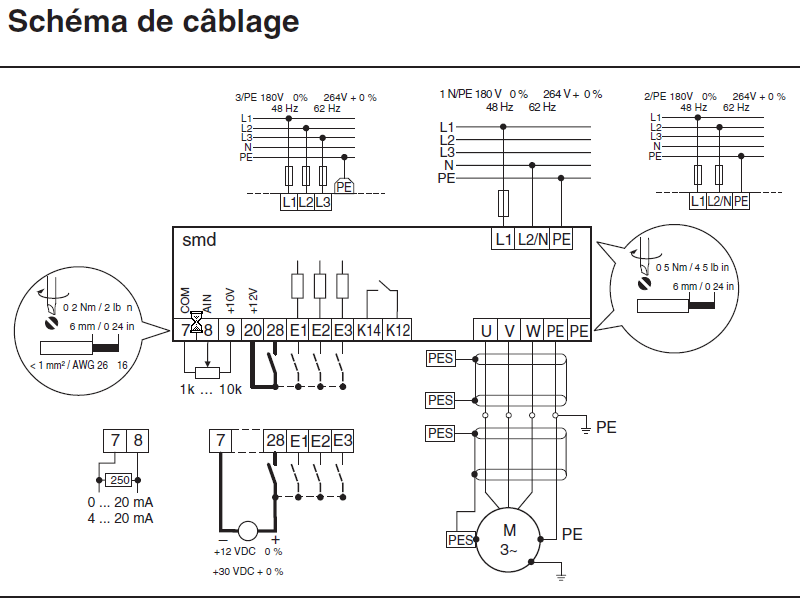


## Touche logiciel (depuis le HMI)



Option avec xAutoReset, implique que la valeur de l’entrée xInput est remise à false automatiquement (très utile pour ne pas surcharger la communication IHM<->API , (HMI<->PLC)).

# Gestion d’un variateur de vitesse



Description :

On souhaite piloter le convertisseur de fréquence Lenze à l’aide d’un automate.

La vitesse est indiqué par un champ de l’interface homme machine en tr/min.

La vitesse doit être modifiée depuis l’automate PIN8

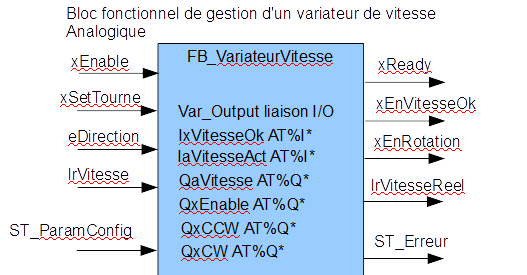
La libération de l’axe (Enable) est obtenue sur l’entrée PIN28

La mise en rotation en sens horaire est activée sur PINE1

La mise en rotation en sens antihoraire est activée sur PINE2

La sortie PINK12 nous indique lorsque la broche a atteint la vitesse de consigne

Programmer le bloc fonctionnel suivant



xEnable : libération de l’asservissement, mise en puissance

xSetTourne : ordre de mise en rotation

eDirection : sens de rotation

lrVitesse : vitesse de consigne

ST\_ParamConfig :

* + tTimeOut pour atteindre la vitesse de consigne
  + iNbreIncInAnalog : indiquer le nombre d’incrément de l’entrée analogique
  + iNbreIncOutAnalog : indiquer le nombre d’incrément de la sortie analogique
  + lrVitesseMax : vitesse [tr/min] si la tension de sortie = lrTensionAnOut
  + lrTensionAnOut : tension de sortie maximale
  + xSignedAnOut : signée = true
  + xAvecSignalVitesse : si vrai, on utilise l’entrée analogique IaVitesseAct pour connaitre la vitesse actuelle
  + tTempsEnv : temps d’enveloppe pour le calcul de la vitesse de sortie si xAvecSignalVitesse =false

ST\_Erreur :

* + xError : indique une erreur
  + iId : no d’erreur

xReady : on attend un ordre de rotation

xEnVitesseOk : la vitesse de consigne est atteinte

xEnRotation : indique que le moteur tourne

lrVitesseReel : sortie numérique de la vitesse actuelle selon l’entrée analogique IaVitesseAct si xAvecSignalVitesse =true ou calcul tTempsEnv

# Gestion des modes de marche et d’arrêt MMA

## GEMMA

Réaliser le GEMMA pour l’exercice 7.

## Programmation du cycle MMA

Intégration de la gestion des modes de marche et d’arrêt dans l’exercice 7.

Cycle d’initialisation, automatique et manuel

## Gestion des pannes simple (par exemple fin de course)

Réaliser la démonstration du principe de fonctionnement.

## Gestion de l’arrêt d’urgence

Réaliser la démonstration du principe de fonctionnement.

# Moteurs

## Terminologie

A l’aide d’internet et du support de cours [20\_Electrique et moteur](file:///\\Intra.he-arc.ch\org\ING\Formation\010-Bachelor\030-Niveau-3\020-ProfsAEtudiants\3268_Systemes_Automatises_II\Outils%20API%20Commun\20_Electrique%20et%20moteur), expliquer les terminologies suivantes :

1. Transmission (Mouvement et mécanique) :
   * Linéaire
     1. Entrainement direct
     2. Entrainement indirect
        1. Vis à bille
        2. Courroie
        3. Crémaillère
        4. Bielle
        5. Mécanisme de combinaison avec 2 moteurs et plus
   * Rotatif
   * Réducteur
     1. Planétaire (notion d’étage et de satellite)
     2. Harmonic drive
     3. Par courroie ou engrenage simple
2. Asservissement :
   * Boucle ouverte
   * Boucle fermée
   * Dual loop
   * En mode couple (courant)
   * En mode vitesse
   * En mode position
3. Trajectoires :
   * Profile du mouvement
   * Interpolation
   * Cinématique direct et inverse
4. Type de moteur :
   * Electrique à Courant Continu
     1. Avec balais
     2. Sans balais (brushless)
   * Electrique universel
   * Electrique Asynchrone
   * Electrique Synchrone
   * Electrique pas à pas
   * Electrique piézoélectrique
   * Electrique voice-coil (principe haut-parleur)
   * Pneumatique
   * Thermique
5. Mesure :
   * Tachymètre
   * Reslover
   * Codeur optique incrémental
   * Codeur optique absolue
     1. Simple tour
     2. Multi tours
   * Capteur hall
   * Codeur magnétique
6. Protocole/type signaux codeur :
   * Analogique sin/cos (1Vpp)
   * TTL
   * EnDat. Hiperface, BISS, SSI, …

# Exercice, moteur pas à pas

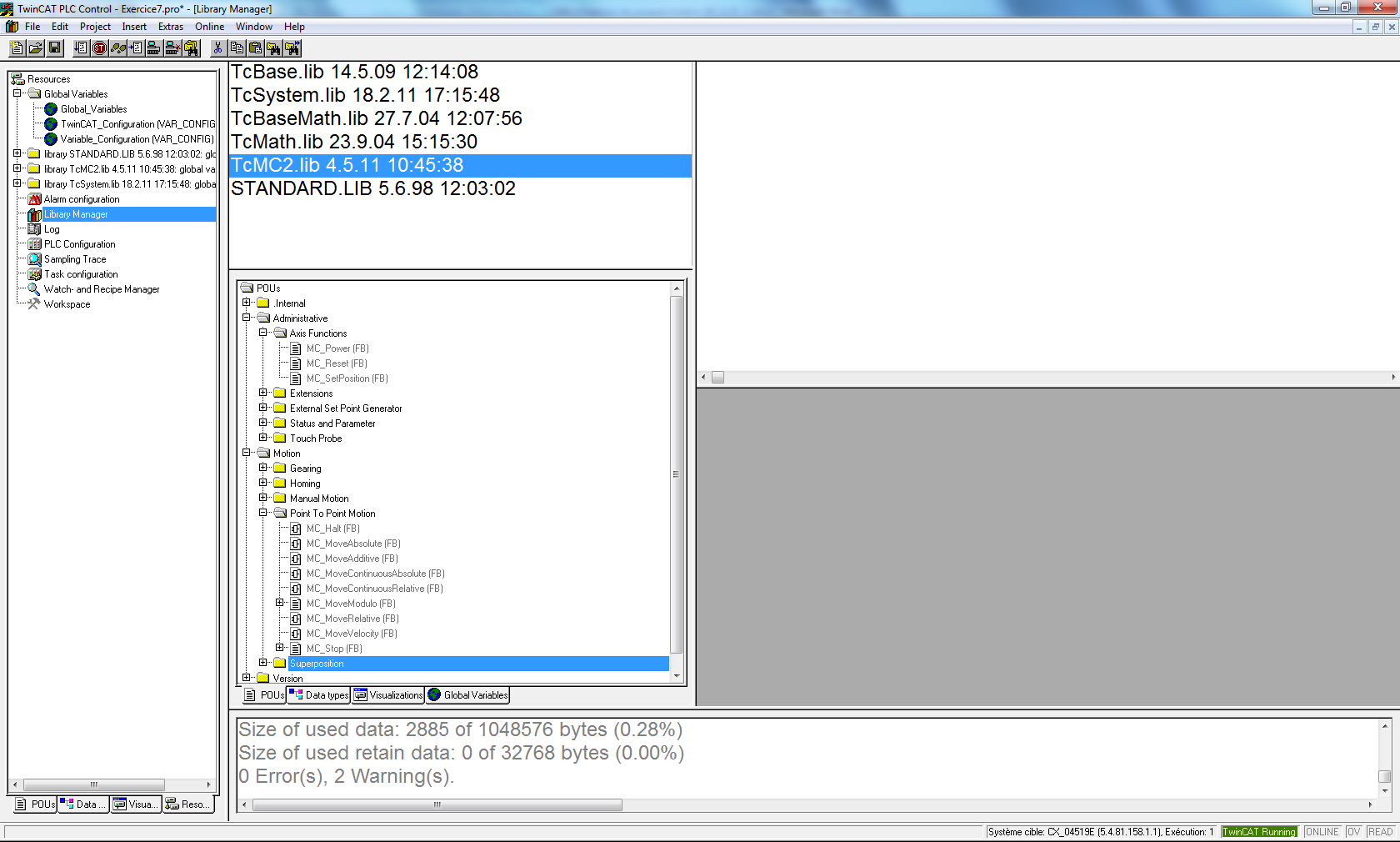
Réaliser l’exercice [Série2\_Exercice1\_Config\_Mot\_PP](file:///\\Intra.he-arc.ch\org\ING\Formation\010-Bachelor\030-Niveau-3\020-ProfsAEtudiants\3268_Systemes_Automatises_II\Outils%20API%20Commun\10_Exercice%20Pratiques%20sur%20API%20Beckhoff\Série2_Configuration_moteur\Série2_Exercice1_Config_Mot_PP.pdf)

Ensuite [Série2\_Exercice2\_Config\_Mot\_PP\_Simu](file:///\\Intra.he-arc.ch\org\ING\Formation\010-Bachelor\030-Niveau-3\020-ProfsAEtudiants\3268_Systemes_Automatises_II\Outils%20API%20Commun\10_Exercice%20Pratiques%20sur%20API%20Beckhoff\Série2_Configuration_moteur\Série2_Exercice2_Config_Mot_PP_Simu.pdf)

Et [Série2\_Exercice3\_Config\_Mot\_PP\_Simu\_Param](file:///\\Intra.he-arc.ch\org\ING\Formation\010-Bachelor\030-Niveau-3\020-ProfsAEtudiants\3268_Systemes_Automatises_II\Outils%20API%20Commun\10_Exercice%20Pratiques%20sur%20API%20Beckhoff\Série2_Configuration_moteur\Série2_Exercice3_Config_Mot_PP_Simu_Param.pdf)

# Exercice, utilisation MC Library 2

Dans le projet [PLC](file:///P:\Formation\010-Bachelor\020-Niveau-2\020-ProfsAEtudiants\2262-Systemes_automatises_I\2262.1%20Systemes_automatises_I%20INF2\Exercice%20API\Exercice7.pro), utiliser les blocs fonctionnels de la librairie « MC Library 2 »



MC\_Power, mise en puissance

MC\_MoveRelative, ordre de déplacement relatif

Dans la séquence FB\_CycleAuto, réaliser le cycle suivant,

* + étape 10, Mise en puissance du moteur fbMoteur1
  + étape 20, le moteur avance de 10mm dans le sens positif
  + étape 30, le moteur avance de 10mm dans le sens négatif
  + étape 90, si xStop on termine, sinon on recommence à l’étape 20

Charger le fichier suivant pour tester la simulation [SystemManger](file:///P:\Formation\010-Bachelor\020-Niveau-2\020-ProfsAEtudiants\2262-Systemes_automatises_I\2262.1%20Systemes_automatises_I%20INF2\Exercice%20API\Exercice7simulation%20moteur.tsm)

Aide en ligne <http://infosys.beckhoff.com>