*Tutoriel : **USER***

Sources :
STAUBLI : https://www.staubli.com/global/en/robotics

Mise en œuvre du Robot STAUBLI « Tx40 / CS8c »

Langage VAL 3

2023/04/03

Sommaire

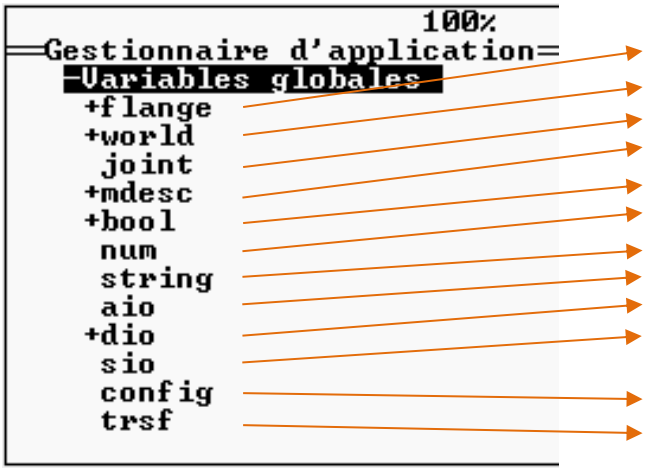
1 Langage VAL3

Le Codage VAL 3

Langage dédié à la robotique, le langage VAL 3 commande de puissants ensembles de fonctions robotiques. Une approche modulaire très flexible, permet de réutiliser et de capitaliser les connaissances accumulées. Une plus grande flexibilité est assurée par un large éventail de possibilités de connexion, y compris DIO, AIO et bus de terrain. L'accès au contrôleur et au robot par une interface simple facilite énormément la programmation.

1.1 STRUCTURE D'UNE APPLICATION

1.1.1 Branche variables globales



Définition des outils
Données cartésiennes : point et frame
Positions articulaires
Descripteur de mouvement (motion descript)
Variables booléennes (true / false)
Variables numériques
Variables chaînes de caractères
Variables entrées/sorties analogiques
Variables entrées/sorties digitales
Variables "sérielles" (port série, socket TCP/IP)
Configuration de bras
Transformations cartésiennes

screen (VAL3 V7+) → Écran utilisateur

1.1.2 Nom des variables

Libre de choix de donner des noms aux variables avec :

_ 15 caractères maxi : lettres (a…z A…….Z) chiffres (0….9) caractère souligné _

_ espace interdit ainsi que : ; , + - * / . ? « ! Etc……

_ 1er caractère est une lettre ou _

Convention interne STÄUBLI :

_ 1er caractère en minuscule désigne le type de la variable

_ 1ére lettre de chaque mot commence par une majuscule

Exemple : bool bStartProd num nCompteurPiece
string sMessageRecu point pPrise joint jDepart
tool tPince trsf trApproPrise etc…..

1.1.3 Variables globales

_ BOOL valeur true ou false bInit=true
_ NUM : valeur numérique réelle nCompteur= 20
_ STRING : Chaîne de caractères sMessage= "cycle en cours"

_ Opérations classiques sur variables numériques : +, -, /,*
_ priorité des opérateurs => utiliser des parenthèses nX=(nZ+nY)*nT

_ (bool) bRun=bStart = Opérateur d 'affectation
bRun=!bStart ! Opérateur NEGATION (NOT)

1.1.4 Tableaux

✓ 1 dimension à 1 élément :
nIndex=10

✓ 1 dimension avec plusieurs éléments :
nIndex[3]=10

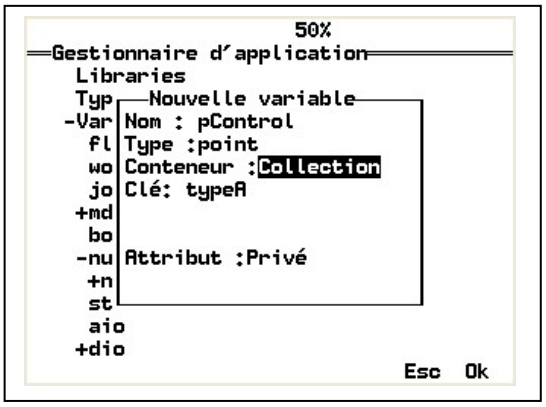
✓ Jusqu'à 3 dimensions avec plusieurs éléments (V7+) :
nIndex[1,0,2]=10

✓ ! Max 10000 éléments
(sur une même application)

1.1.5 Collections (V7+)

✓ Tableau ou l'identifiant des éléments n'est pas une valeur numérique mais une clé (chaîne de caractères) :

```
movel(pControl["typeA"],tPince, mLent)
```



✓ La clé peut être une variable String :

```
sRef="typeA"
movel(pControl[sRef],tPince, mLent)
```

✓ Les clés sont classées dans l'ordre alphabétique

✓ Instructions pour parcourir une collection : first , next, last, prev

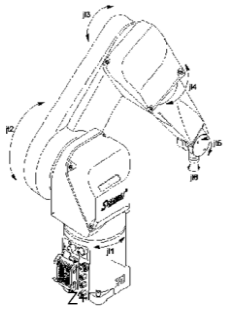
(exemple chapitre page utilisateur)

✓ Utile pour gérer plusieurs références de produits dans une même application

- Données géométriques

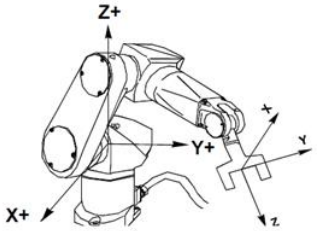
✓ **Positions articulaires** : Branche **JOINT**

→ Relatif au point articulaire Zéro du menu Calibrage
(0, 0, 0, 0, 0, 0)



✓ **Points Cartésiens** : Branche **WORLD**

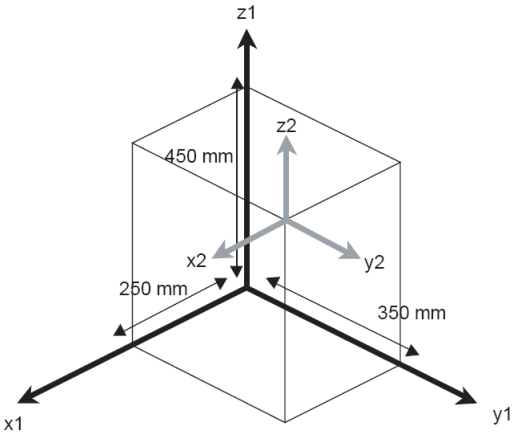
→ Liés à un frame (world ou autre)
→ X, Y, Z, Rx, Ry, Rz



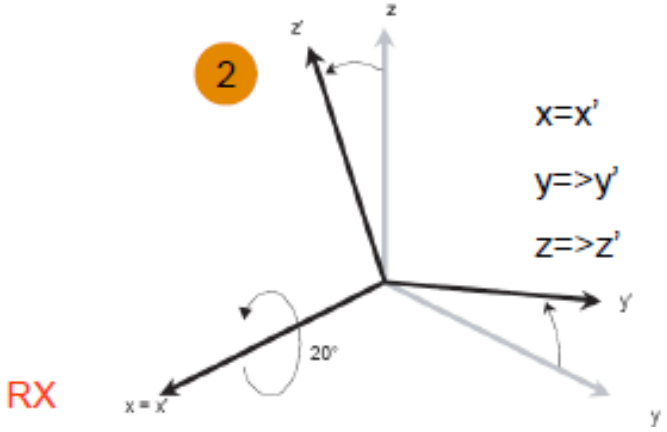
✓ Frames sont appris avec 3 points depuis l ’interface boîtier manuel
→ Sélection du Frame par la touche FRAME du boîtier
→ Déplacement au boîtier dans le FRAME sélectionné.

- trsf : Transformations {x, y, z, rx, ry, rz} = {250,350,450,20,10,30}

1

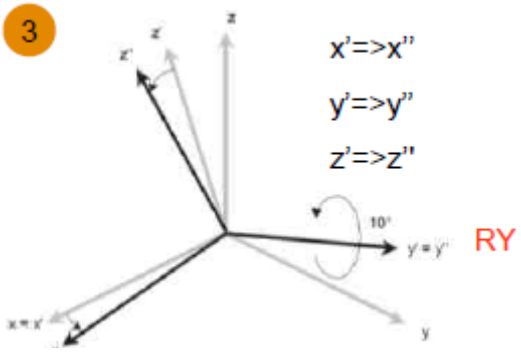


2

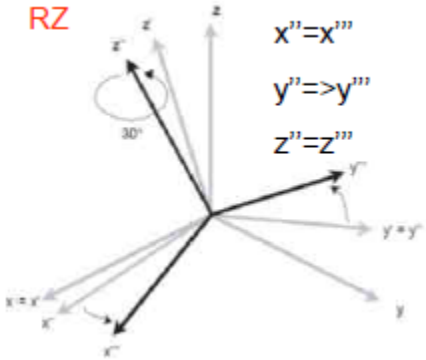


{250,350,450,0,0,0}
{250,350,450,20,0,0}

3

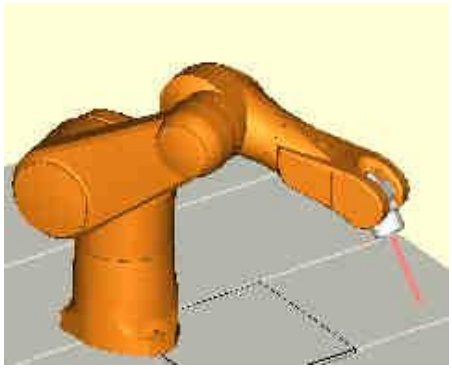
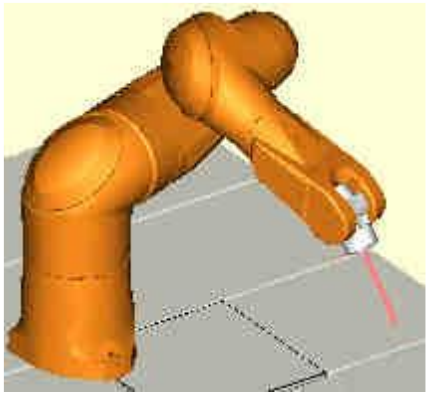
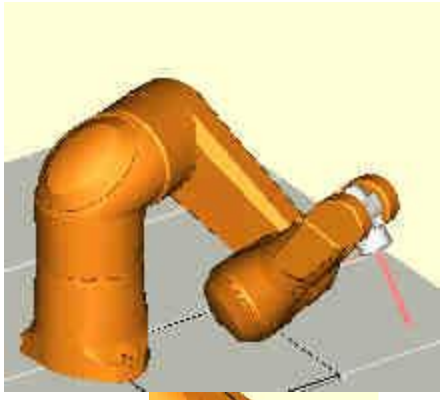


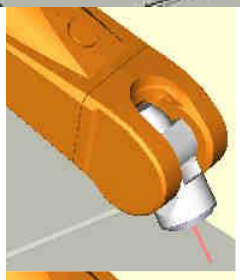
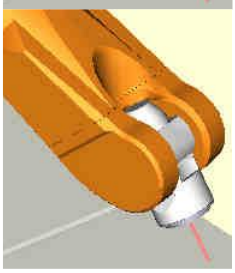
4



{250,350,450,20,10,0}
{250,350,450,20,10,30}

- **Jusqu'à 8 configurations sur points cartésiens**

<u>On choisit de définir la configuration du bras avec JOINT</u>

✓ Approche zone : JOINT passage par la position sans marquer l'arrêt

✓ Action (prise/pose piece, process) sur une position : POINT

✓ **Au minimum 1 position articulaire dans une application (ex : jDepart)**

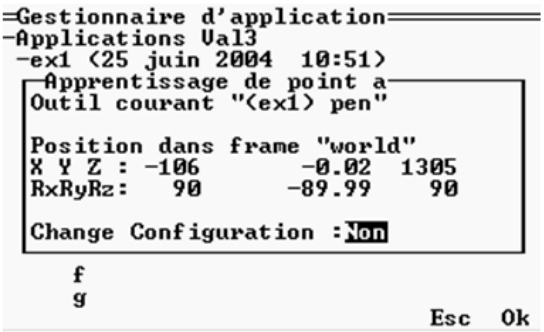
✓ 1 position JOINT pour chaque zone avec configuration bras différente

- MOUVEMENTS ÉVOLUÉS

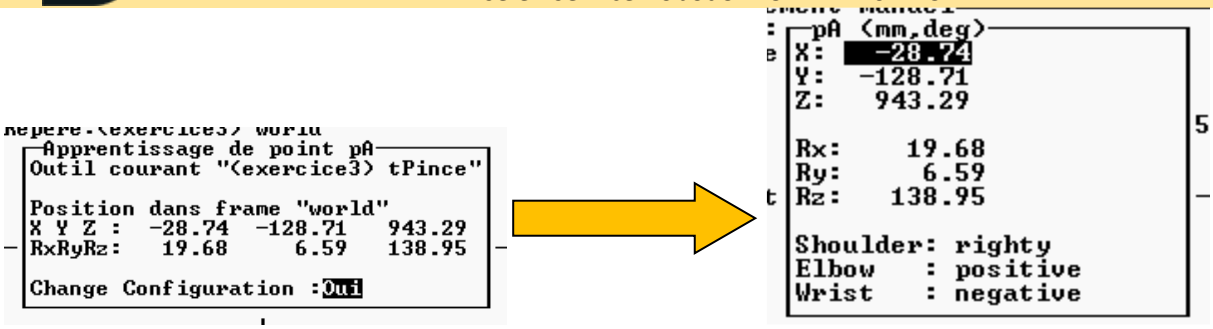
1. Points cartésiens

<u>Quand un point cartésien est appris :</u>

→ Change Configuration : Non : garde la même configuration que celle du mouvement précédent (= refus de changement de configuration = SAME)



→ Change Configuration : Oui : le bras aura toujours la même configuration que celle courante

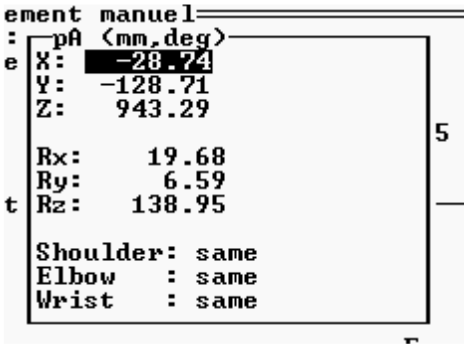


Utilisation risquée avec le OUI !!!

→ Avec SAME, plus de possibilités :

_ le bras garde la même configuration que le 1er point articulaire de la trajectoire.

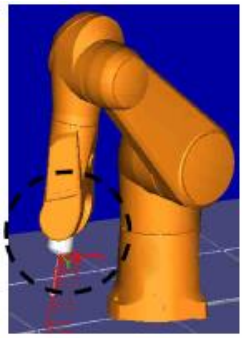
_ Pour modifier la configuration de toute la trajectoire, il suffit uniquement de réapprendre le 1er point



- Singularités robot 6 axes

Alignement d'axes sur mouvements en lignes droites :

- Poignet : J5 proche de 0 degré
- Robot plafond + poignet dans le prolongement du pied



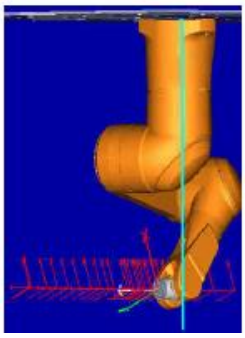
Animation 1

Animation 2

↳ Conséquence : fort ralentissement pénalisant pour le temps de cycle

↳ Solution :

- Utiliser MOVEJ « si possible »
- Modifier implantation cellule
- Modifier le préhenseur



Animation 3

⚠ Axes 4/5/6 alignés

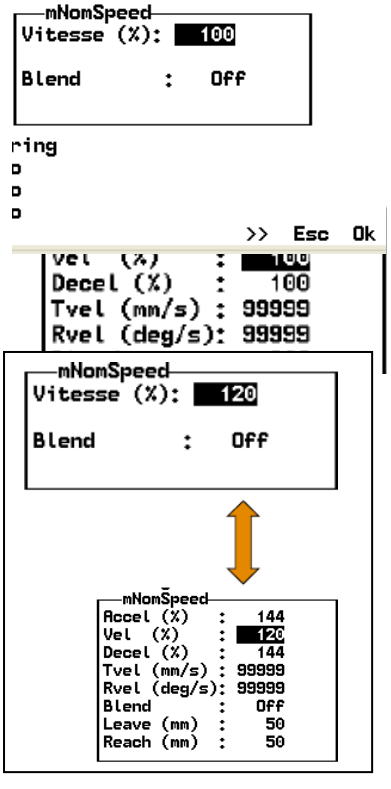
NB : si possible, travaillez avec un poignet cassé

- Descripteur de mouvement : mDESC

✓ vel, accel, decel : % des valeurs nominales sur articulations

✓ En complément : pour un contrôle type process :

↳ tvel : vitesse de translation max. en bout outil en mm/s, (ex : si on souhaite une passe à 50mm/s pour déposer de la colle nous avons réellement 50mm/s **si** vel=100%)



↳ rvel : vitesse de rotation max. en bout outil en degrés/s

↳ hors process : on garde les valeurs définies

✓ Au final : Valeur la plus restrictive sera utilisée

AJUSTEMENT AUTOMATIQUE

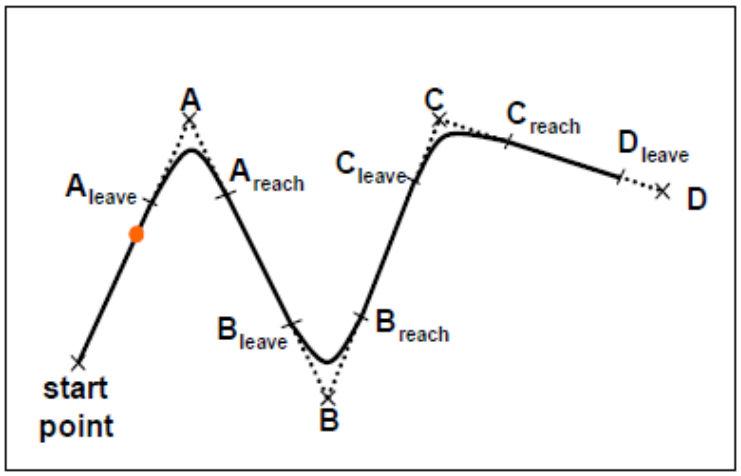
✓ Dans page édition simplifiée (<<) :

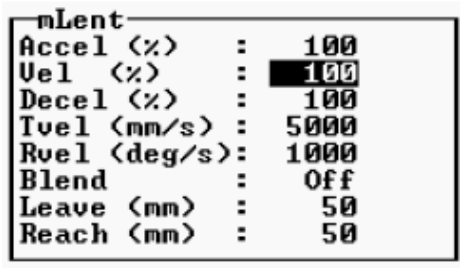
↳ Ajustement automatique des accel/decel

optimise le ratio vel/accel
_accel = vel ²
_ex : vel=120% (1,2)² =1,44
_Résultat : accel=decel= 144%

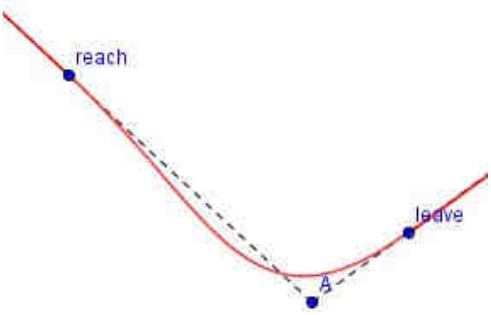
✓ Pour réglage autre, aller dans page édition évoluée (>>) par F6

- **Lissage : BLENDING**





✓ Lissage activé par blend = joint (calcul en mode articulaire



✓ Amplitude paramétrable avec :
_ leave : distance en mm quitte trajectoire.
_ reach : distance en mm rejoint trajectoire.

_ (V7+) Lissage cartésien : blend= cartesian

!!!! Leave et reach très différent →

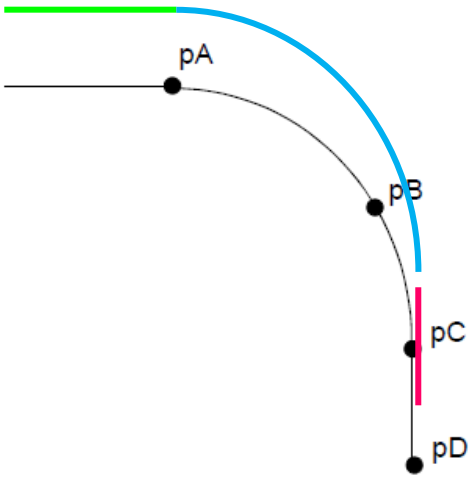
- **Mouvement : MOVEC**

Movec(point,point,tool,mdesc)

Movel(pA,tPince,mRapide)

Movec(pB,pC,tPince,mRapide)

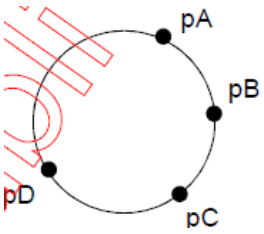
Movel(pD, tPince,mRapide)



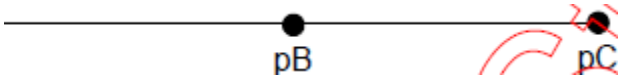
- Interpolation circulaire : mouvement en arc de cercle
- MDESC spécifie le lissage en fin d'arc

- Cercle réalisable avec 4 points :

movec(pB,pC,tPince,mRapide)
movec(pD,pA,tPince,mRapide)

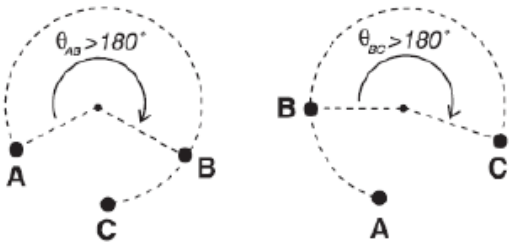


- ligne droite possible : movec(pB,pC,tPince,mRapide)

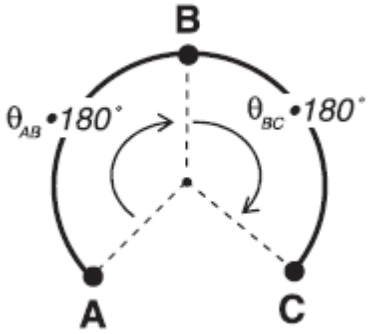


MOVEC : PARTICULARITES

Mouvements impossibles
(pas de solution unique)



Le point intermédiaire doit faire un angle inférieur à 180° avec le point de départ ou d'arrivée



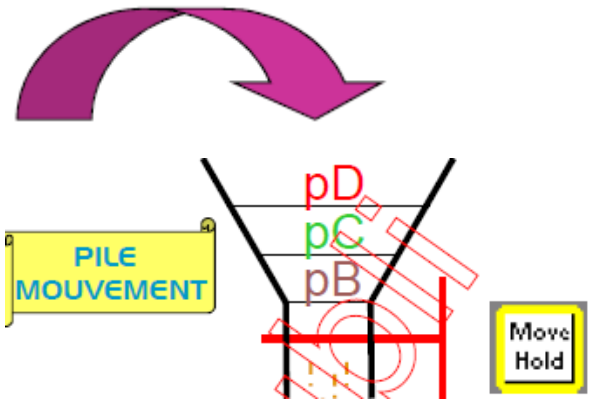
Voir Chapitre « Contrôle des mouvements » dans Manuel de référence VAL3

- **Application & mouvement**

movej(pA, ,)
movel(pB , ,)
movej(pC , ,)
movel(pD , ,)

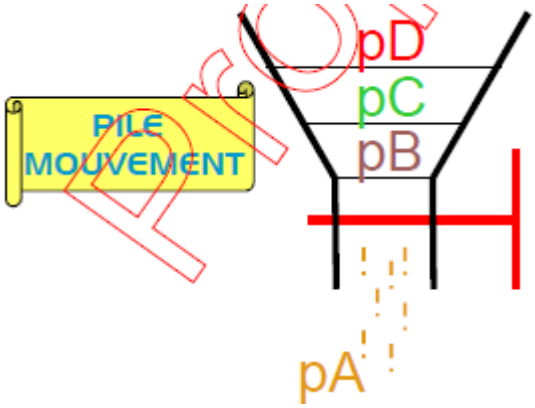
DEBOGUEUR

movej, movel, movej,...



!!! COMPLETE INDEPENDANCE ENTRE LES 2 NIVEAUX !!!

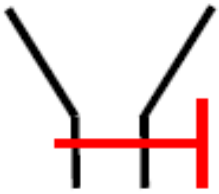
CONTRÔLE DES MOUVEMENTS 1



stopMove()

restartMove()

↳ Arrêt / Reprise des mouvements par programme



resetMotion()

↳ Supprime les mouvements de la pile de mouvement

NB : A effectuer au moins une seule fois dans le start du programme pour vider la pile mvt

CONTRÔLE DES MOUVEMENTS 2

resetMotion(jStart) où jStart est une variable joint

↳ vide la pile de mouvement et génère un mouvement de connexion vers jStart

Par défaut les mouvements de connexion en mode déporté s'exécutent par maintien de MOVE/HOLD

autoConnectMove(true) les mouvements de connexion sont effectués automatiquement à vitesse faible (utile si travail sans boîtier manuel)

- Approche sur points cartésiens

1. Variable de type : TRSF

⚠ avec repère WORLD

✓ Variable transformée **TRSF** permet de faire des opérations mathématiques sur les points cartésiens.
Ex : Approche sur point, décalage dans palette , ….

✓ 6 champs numériques : x, y, z, rx, ry, rz

✓ si trsf trDecal déclarée comme variable
↳ Écriture possible dans un programme:
trDecal ={0,0,-100,0,0,0}
↳ aussi trDecal.x=0 trDecal.y=0 trDecal.z=-100 trDecal.rx=0 ...

⚠ Impossible de faire des mouvements sur une TRSF
↳ !! Réservé QUE pour du calcul !!

Exemple : pA= {300,-10,+20,0,180,-10}

trDecal= {0, 0,-100,0, 0, 0}

pCalcul= {300,-10,-80, 0,180,-10}

- **Approche sur un point**

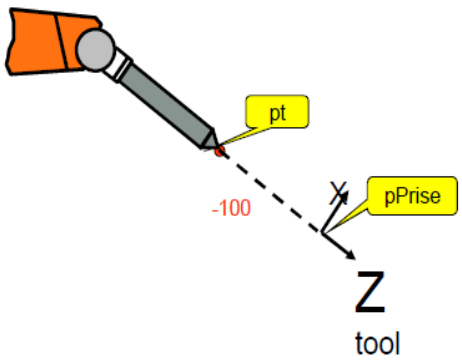
POINT <== appro(POINT,TRSF)

APPRO calcule un point cartésien relatif à un point auquel on applique une transformée

↳ ! effectue un décalage suivant le repère outil courant

POINT pt POINT pPrise TRSF trDecalz NUM nDistance=100 sont définis

```
trDecalz={0,0,-nDistance,0,0,0}
pt=appro(pPrise,trDecalz)
movej(pt,tPince,mRapide)
```



Autre écriture possible : *(écriture préférable)*

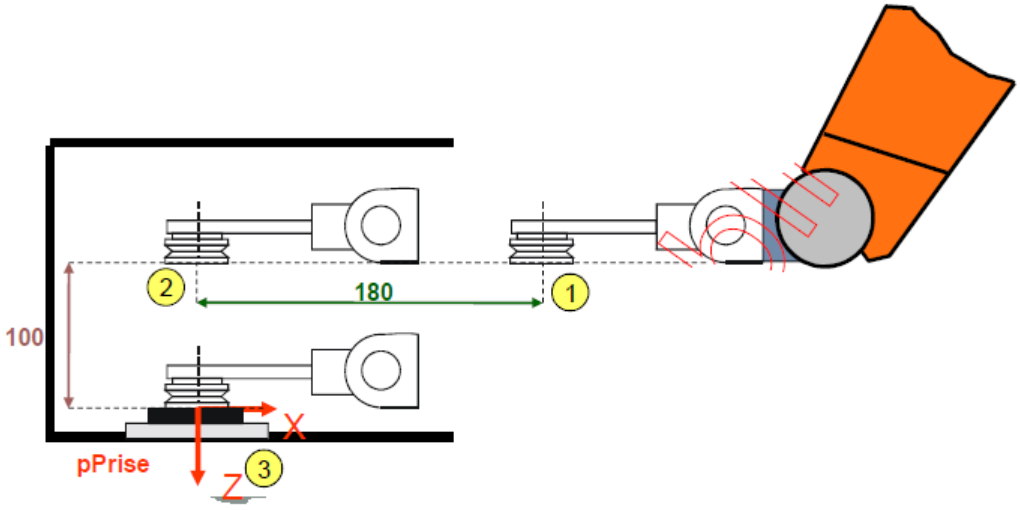
```
movej(appro(pPrise,trDecalz),tPince,mRapide)
```

Autre écriture possible : *(écriture préférable)*

```
movej(appro(pPrise,{0,0,-100,0,0,0}),tPince,mRapide)
```

APPROCHE COMPLEXE - 1 - :

```
movej(appro(pPrise,{180,0,-100,0,0,0}),tPince,mLent)        ①
movel(appro(pPrise,{0,0,-100,0,0,0}),tPince,mLent)          ②
movel(pPrise,tPince,mLent))                                 ③
```



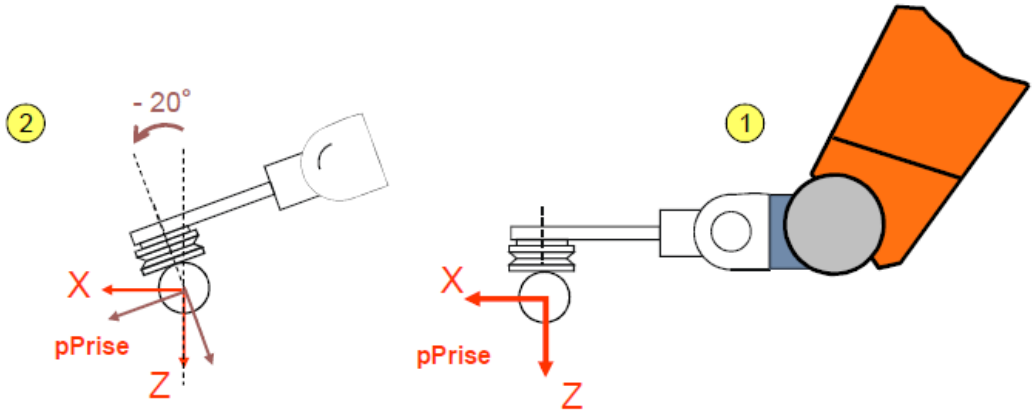
pPrise{0,0,0,0,0,0}

APPROCHE COMPLEXE - 2 –

movel(pPrise,tPince,mLent) ①

movel(appro(pPrise,{0,0,0,0,-20,0}),tPince,mLent) ②

(Blend =off)

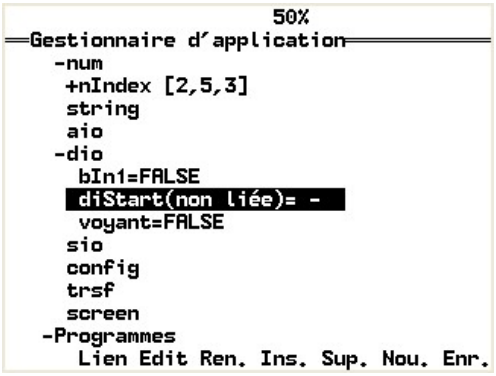


1.2 ENTREES SORTIES DIGITALES

1.2.1 Variables globales : DIO

DIO doit être liée à E/S physique pour être utilisable :

- DIO doit être liée manuellement, depuis le menu Gestionnaire d'Application ou depuis SRS
- DIO peut être liée à une autre DIO
  dioLink(dio1,dio2)



dans

Sinon erreur détectée uniquement à l'exécution

1.2.2 Lecture des entrées : ==

test diCapteur == true retourne true si actif
test diCapteur == false retourne true si inactif

== : test sur une entrée

Attente bloquante sur entrée : wait(diCapteur == true)

Attente avec un temps limite spécifié en secondes :

bOk=watch (diCapteur == true, 2)

- retourne true si condition arrive avant 2 secondes et false sinon

Test de différence : diCapteur1 != diCapteur2

1.2.3 Contrôle des sorties : =

- **doVerin=true** pour activer la sortie
- **doVerin=false** pour désactiver la sortie

- Affectation avec un booléen : **doVerin=bRun**

- Copie d'une entrée dans une sortie : **doVerin=(diCapteur==true)**

1.2.4 Synchronisation mouvements & sorties : waitEndMove()

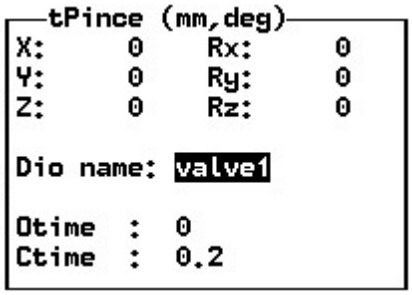
VAL3 : Anticipation exécution du programme par rapport au mouvement

↳ Pour change rl'état d'une sortie à une position robot donnée : **waitEndMove()**

En effet, ajouter « waitEndMove » à la fin du pg à cause de l'entonnoir)

```
movel(pControle ,tPince, mRapide)
waitEndMove()                    Annule le lissage sur mouvement précédent
doStartControle= true                  (en pas à pas peut générer un mouvement)
```

Cas des outils : Si outil défini : tPince



open(tPince) ou close(tPince)

inclus waitEndMove() + temporisations
ouverture ou fermeture

1.2.5 Temporisation : delay()

delay(num)

Permet de suspendre l'exécution d'un programme pendant un temps spécifié en secondes
Précision : 1 s

```
movel(pControle, tPince, mLent)
waitEndMove()                    signal de mesure à un point
doStartControle= true                  de contrôle pendant 2.5 secs
delay(2.5)
doStartControle=false
```

…

- PROGRAMMATION STRUCTURÉE

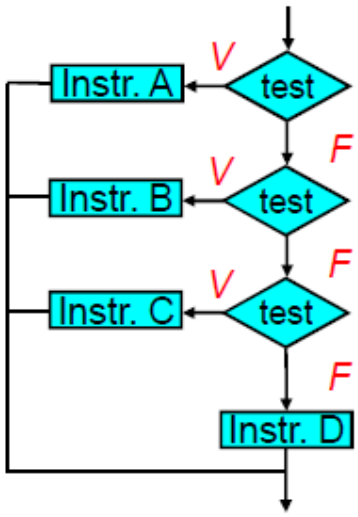
1. Opérateurs : conditions

Opérateurs : <, >, ==, !=, >=,<=, and, or, xor, !

Conditions : nX == 1 Vrai (true) ou Faux(false) nY != 3 nY différent de 3

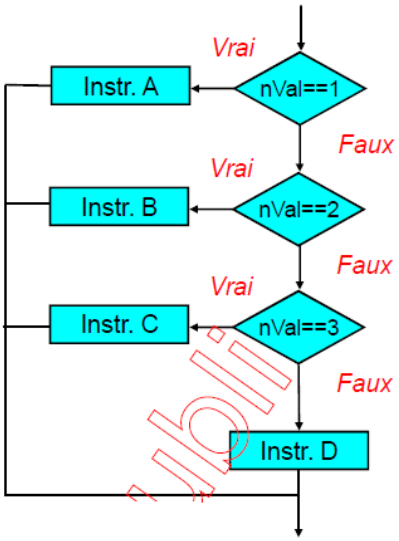
```
If nX==1
        Instructions VAL3
endIf
```

Forme simple :




```
if nY != 3
        |  Instructions A
elseif nX>2
        |  Instructions B
elseif (nY==5) or (nX==6)
        |  Instructions C
else
        |  Instructions D
endIf
```

- **Structure : SWITCH (case)**




```
switch nVal
case 1
        |  Instr. A
break
case 2
        |  Instr. B
break
case 3
        |  Instr. C
break
default
        |  Instr. D
break
endSwitch
```

Test plusieurs valeurs de nVal

break : sépare les tests

test la valeur « defaut »

NB : fonctionne avec des numériques ou des chaîne de caractères

case 1,2,3 : nVal == 1 ou 2 ou 3

- **Boucle : Nombre de passage dans la boucle connu (for)**

```
for nCompteur= 1 to 10
        <Instructions >
endFor
```

Ex1 :

```
for nCompteur = 0 to 49
        movel(pTraj[nCompteur],tPince,mRapide)
endFor
```

Ex2 :

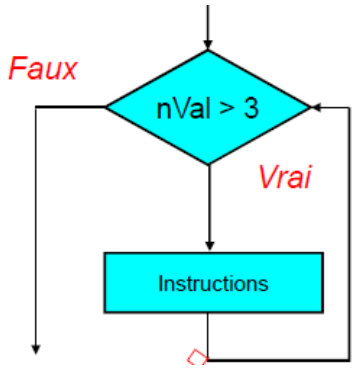
```
nDebut=100
nFin= 10
for nIndex = nDebut to nFin step -1          pas différent de 1
        nDecal[nIndex]= nCote*nIndex
endFor
```

Utilité : pour les trajectoires, les tableaux…

Exemple : avec fichier CAO, création de tableaux de points : traj

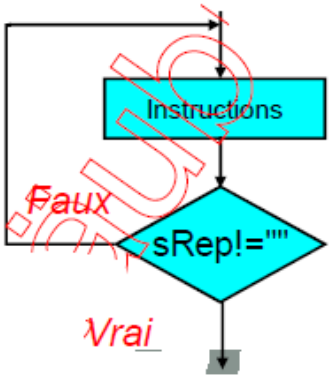
⚠ si on ne met rien, implicitement l'incrément sera de 1

- **Boucles conditionnelles (nombre de passage dans la boucle inconnu)**




```
while nVal > 3
        |
        | Instructions
endWhile
```

« tant que…faire »




```
do
        |
        | Instructions
until sRep !=" "
```

« faire…jusqu'à »

- **Sous-programmes**

programme appli()

….

….
CALL prise()
….
….
Le programme appli() appelle (call) le programme prise(), l'exécute et retourne dans le programme appli().

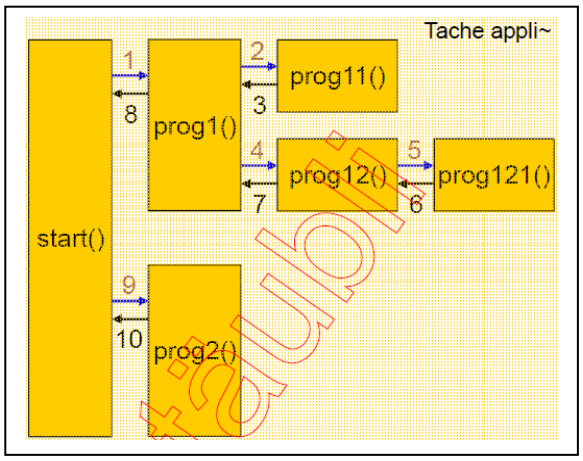
- **Séquence exécution des programmes**

- Démarrage « appli », système crée une tache nommée « appli~»
- Le programme start est exécuté dans cette tâche
- Lors de l'appel aux sous-programmes, on déroute le cycle d'exécution

Programme Start
call prog1()
call prog2()

Programme Prog1
call prog11()
call prog12()

Programme Prog12
call prog121()

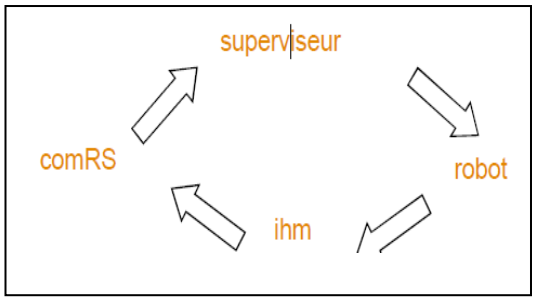


- **Multitâche**

⚠ 1 TACHE MVT ROBOT ET NON 2 TACHE EN MÊME TEMPS CAR 1 SEUL BRAS

- Système multi-tâches = plusieurs programmes peuvent s'exécuter en parallèle.
- Pour gérer plusieurs fonctions indépendantes les unes des autres.
- Temps processeur partagé mais basculement très rapide d'une tâche à une autre
- Communication/synchronisation entre tâches par Variables globales + E/S

Exemple :
- Tache gestion mouvement du bras : « robot »
- Tache gestion du système (mode de marche cellule, traitement erreur, vérif.conditions initiales,…) : « superviseur »
- Tache interface utilisateur : « ihm »
- Tache communication avec système externe, vision, RS232, socket Eth, … : « comRS »



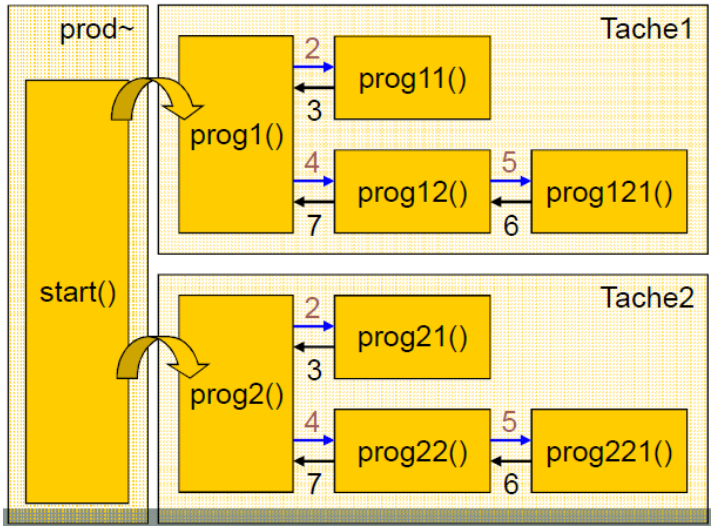
- Chaque tâche est identifiée par un nom et a une priorité
- Priorité : nb de lignes max. à exécuter avant de passer à la tache suivante

Ou instruction bloquante : delay, wait, watch, waitEndMove, open, close, get,…
↳ système passe d'une tâche à une autre

taskCreate “securite”,100,prog(params) : crée nouvelle tâche

Nom Tâche — Priorité 1 à 100 — Nom du prog. à lancer — Paramètres (opt.)

• Autres instructions gestion tâche : taskKill, taskSuspend, taskResume, askStatus
↳ vérifie l’état d’une tâche avant d’en faire une autre
• Démarrage « prod » , système crée une tache nommée « prod~»
• Le programme start est exécuté dans cette tâche
• start crée tache1 et tache2 puis la « prod~ » disparaît (fin de start)
• tache1 et tache2 s’exécute indépendamment l’une de l’autre



tache

APPLICATION PROD

Programme Start
taskCreate "tache1", 100, prog1()
taskCreate "tache2", 100, prog2()
// fin de la tache prod~

Programme Prog1
call prog11()
call prog12()

Programme Prog2
call prog21()
call prog22()

- FENÊTRE UTILISATEUR

1. Historique

_ popUpMsg(string) Affiche une fenêtre avec un message qui doit être acquitté par l’opérateur

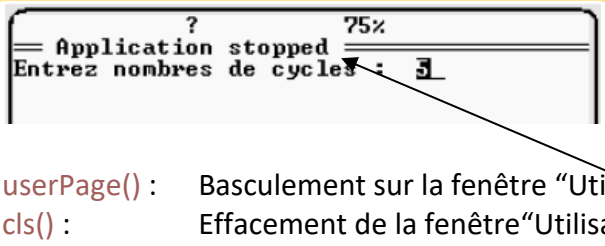
_ logMsg(string) permet d’écrire des message applicatifs dans « l’historique des évènements » , la date et l’heure sont automatiquement ajoutés au message.

_ logMsg("Texte a ecrire dans historique")

=> Ligne ajoutée dans fichier « errors.log » :
« [01/01/06 00:00:01] USR:Texte a écrire dans historique »

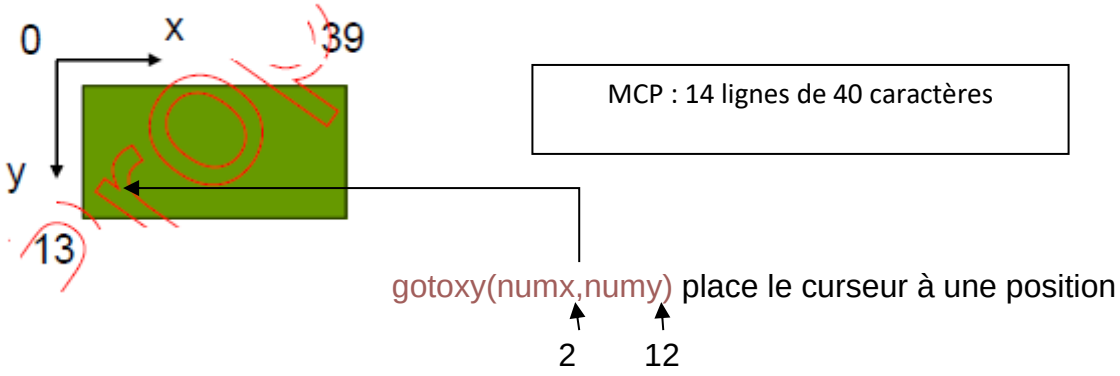
- **Fenêtre utilisateur**

La fenêtre utilisateur est disponible pour afficher des messages ou réaliser la saisie de données par l’opérateur



userPage() : Basculement sur la fenêtre “Utilisateur”
cls() : Effacement de la fenêtre“Utilisateur”
title(string) : Changement du titre de la fenêtre “Utilisateur”

- **Fenêtre utilisateur: AFFICHAGE**



put(num) / put(string) : affiche un message ou une donnée sans retour à la ligne

putln(num) / putln(string) : identique mais avec retour à la ligne

cls()

```
put("nb de pieces produites : ")
putln(nCompteur)
sMessage="cycle en cours …"
putln(sMessage)
```

- **Fenêtre utilisateur : SAISIE**

nTouche=get(num) ou nTouche=get(string)

Attente de la saisie d'une donnée jusqu'à validation par les touches "ENTER" ou "ESC" ou une touche de menu.

```
put(" Entrez le nombre de cycles : ")
get(nNbCycles)                    saisie
put(nNbCycles)                    afficher le nombre
```

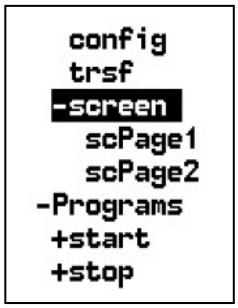
"?" est affiché dans la barre d'état quand une saisie est en attente dans la fenêtre "UTILISATEUR"

nTouche = get() attend et retourne le code de la prochaine touche enfoncée : instruction blocante
nCode=getKey() sans attente => si aucune touche nCode = -1 pas d'instruction blocante

- Multifenêtre : VARIABLE SCREEN (V7+)

• Variable SCREEN permet de gérer plusieurs pages utilisateur
• Mêmes instructions :

userPage(screen) _ affiche la page définie par la variable
cls(screen)
title(screen,string)
gotoxy(screen,num,num)
put / putln(screen,num)
get(screen,num) ou get(screen,string)



exemple :

```
userPage(scPage1)
title(scPage1, " Menu")
putln(scPage1,"En cycle …")
title(scPage2, " Réglages")
…
```

- Timer : CLOCK()

nVal =clock()

Retourne la valeur courante de l'horloge interne en secondes.
Précision = ms
Valeur initialisée à 0 au démarrage de la baie
Roll Over au bout d'environ 49 jours

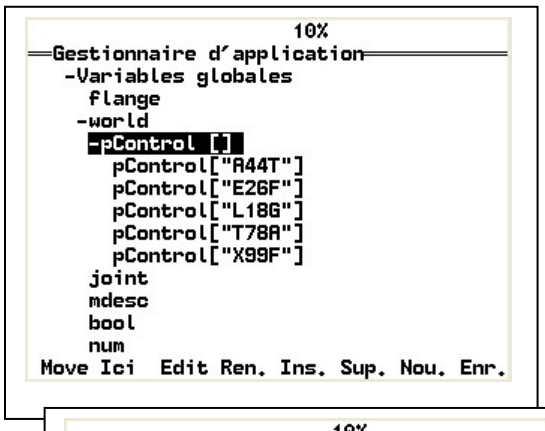
Calcul de temps de cycle :

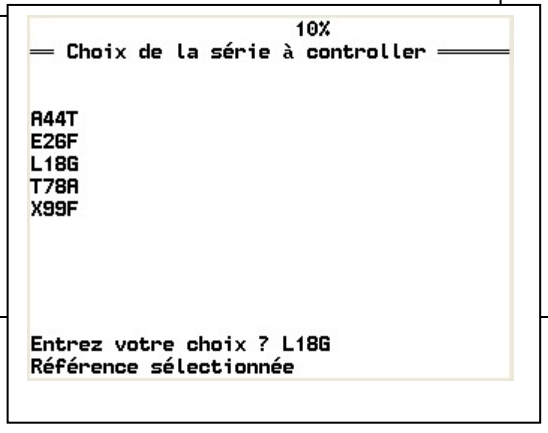
```
nDebut=clock()
// mouvements robot
nTemps=clock()-nDebut
```

- Exemple de page utilisateur

```
//Affichage de la liste des points d'une collection

userPage()
cls()
title("Choix de la série à controler")
gotoxy(0,2)
sElt=first(pControl)
while sElt!=""
        putln(sElt)
        sElt=next(pControl[sElt])
endWhile
gotoxy(0,12)
put("Entrez votre choix ? ")
get(sRef)
putln(sRef)
if isDefined(pControl[sRef])
        put("Référence sélectionnée")
else
        put("Référence invalide")
```





```
        return
endIf
...
```

NB : « collection » = tableau dont l'indice est une clé

- Variables locales et paramètres

Variables globales :

Communes à tous les programmes

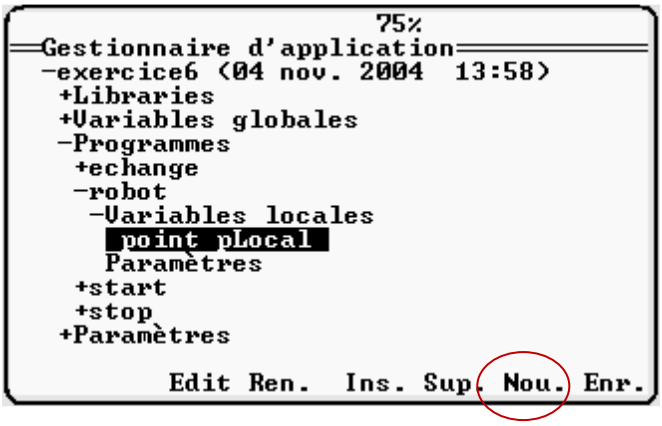
⚠ Risque de corruption de variables

Variables locales :

Indépendantes d 'un programme à un autre

⚠ nécessite le passage de paramètres pour partager des données.

1. Variables locales



Les variables locales doivent être initialisées dans le programme où elles sont utilisées

• Copie à partir d'une variable existante

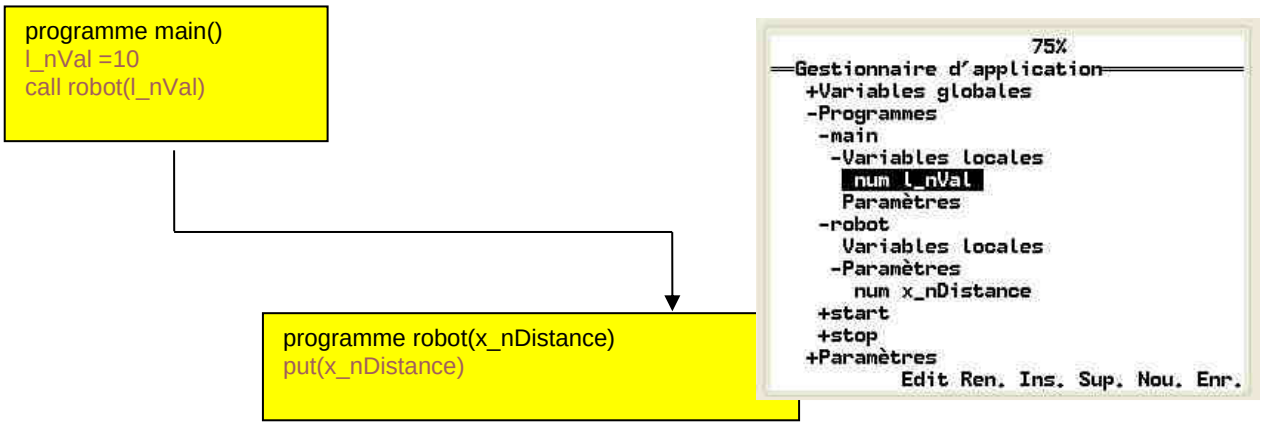
```
l_pLocal= pPrise
```

• Généré par le résultat d'un calcul ou d'une fonction

```
l_nVal = clock()
l_pLocal=here(tPince, world)        → enregistre le point robot
l_jLocal=herej()
```

- Paramètres

- Passage de paramètres :



⇨ affiche 10

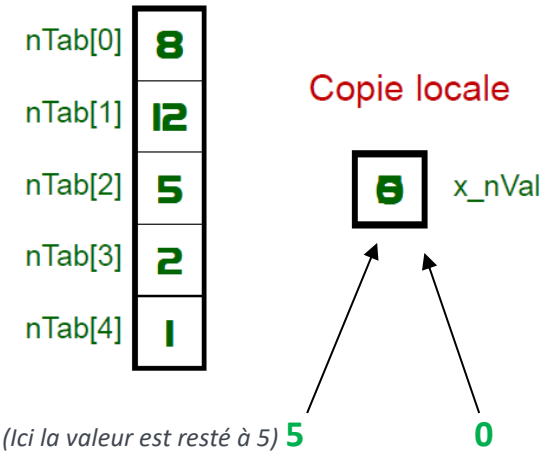
Autant de paramètres que nécessaire (X=paramètre)

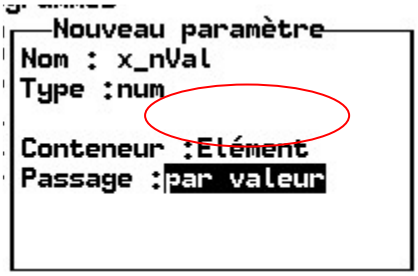
! Attention ! à l'ordre des paramètres et leur nombre et leur type

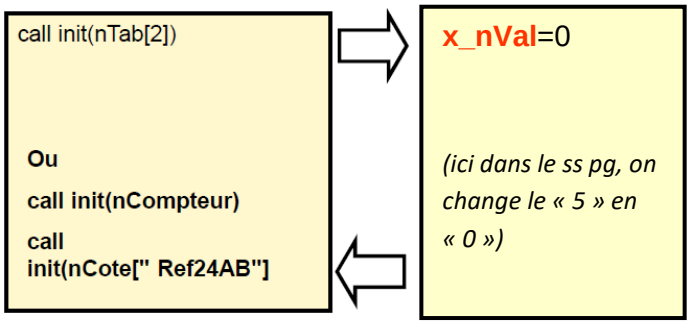
↳ Prise(X_Position, X_approche, X_tool)
Point trsf tool

Utilité : **pour des actions répétées, on utilise les paramètres** (voir ex2)

- Passage d'éléments par valeur

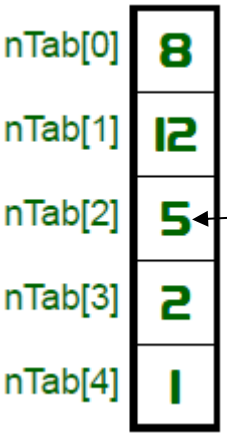






⚠ : On travail sur une copie mais **on ne change pas** l'originale avec un passage par valeur

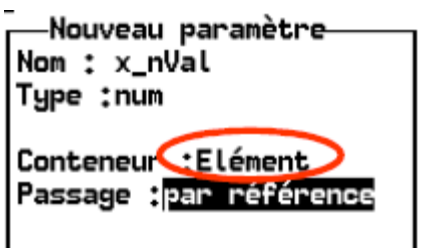
- Passage d'éléments par référence

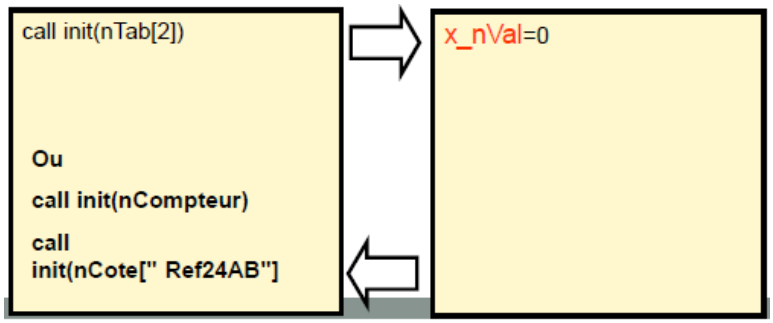


Mémoire commune

x_nVal

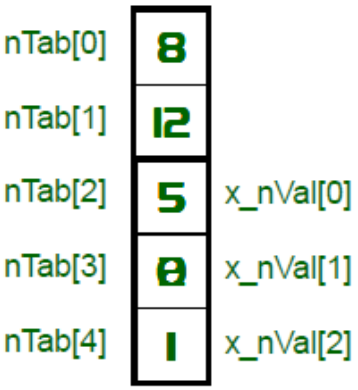
ici « 5 » sera changé en «0 »

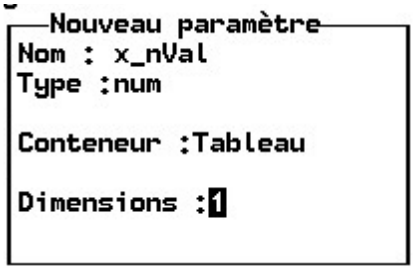


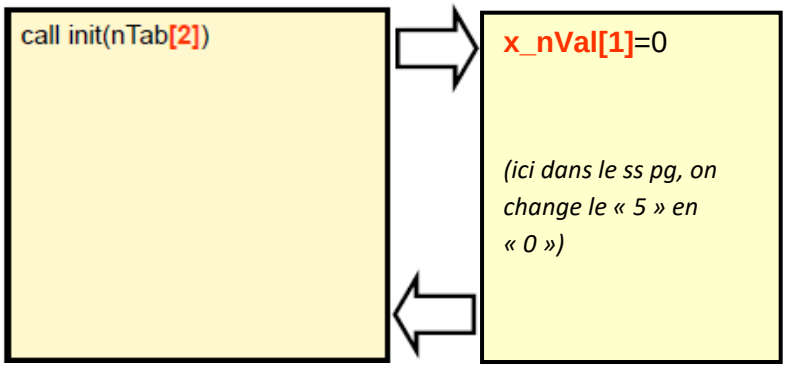


: ici **on modifie** la valeur d'origine

- Passage de tableaux



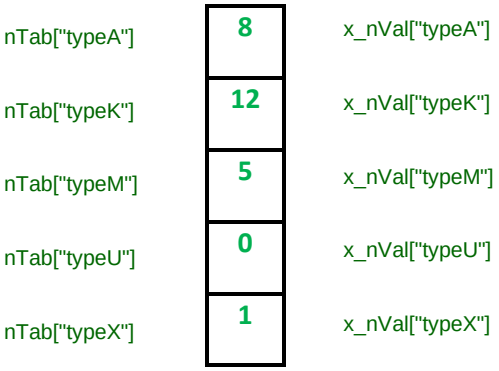


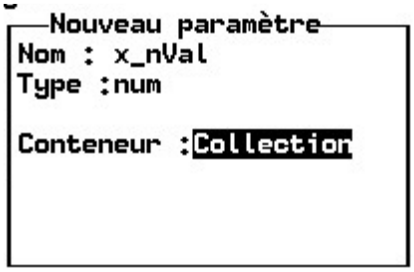


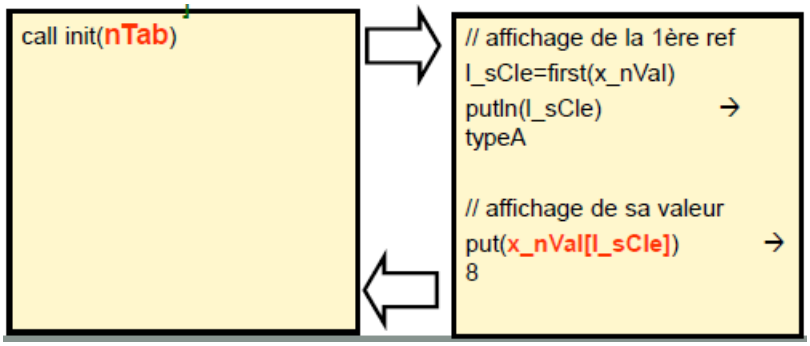
Dim 2 _ x_nVal[?,?]
Dim 3 _ x_nVal[?,?,?]

⚠ Par « tableau » le passage se fait ***Toujours par référence***
↳ : ici **on modifie** la valeur d'origine

- Passage de collections







⚠ Toute la collection
↳Toujours par référence
↳ : ici **on modifie** la valeur d'origine

NB : « collection » = tableau dont l'indice est une clé

- FRAME

1. Pourquoi un frame

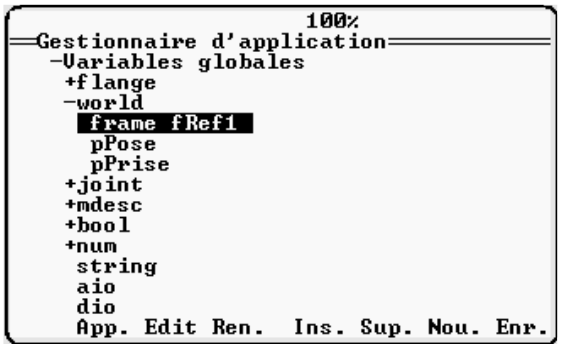
Mon robot est en production, l'application tourne à plein régime mais ….Marcel passe par là avec son chariot et …!!!! CATASTROPHE !!!!!

… une journée d'apprentissage de point… ⇨ Sauf si ...

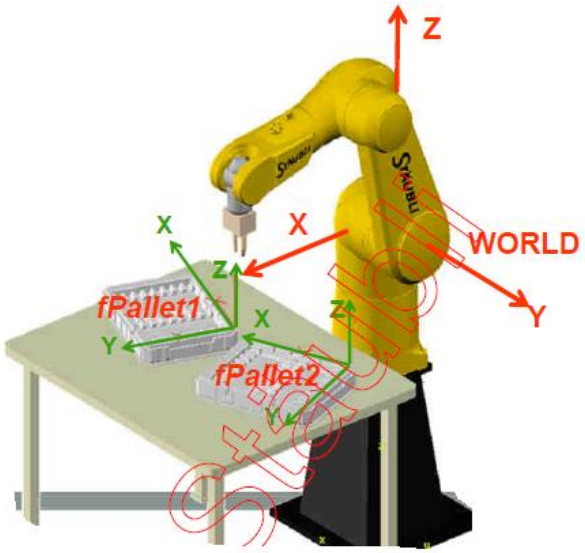
- Utilisation du frame

Repère de travail local :

- Pour faciliter le ré apprentissage des points
- Utiliser pour dupliquer des points
- Décalage de points dans palette



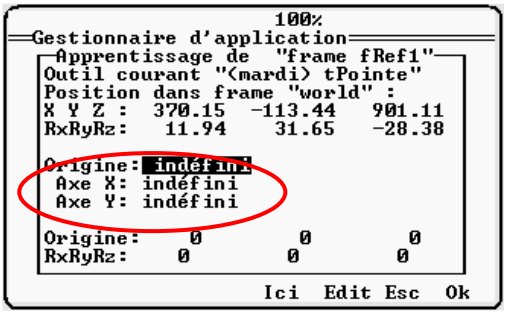
Créer un nouveau repère

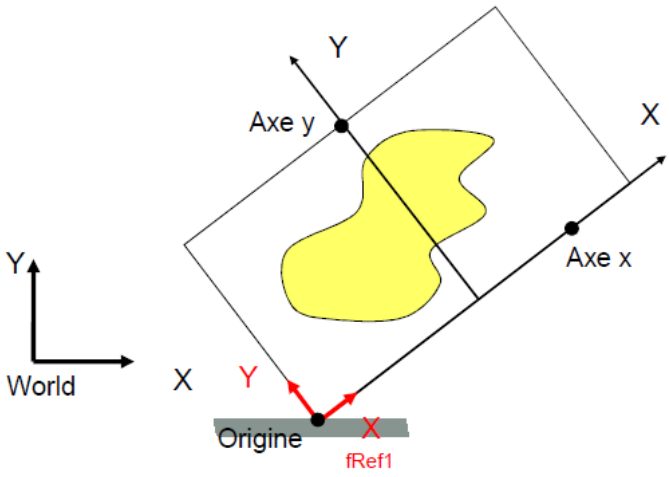


- Apprentissage du frame

Défini par 3 points à apprendre :

- Utiliser un outil précis : pointe
- Définir cet outil comme courant
- Apprendre les points les plus écartés possible (+ de précision)





⚠ Avec quel outil j'apprends le repère ?

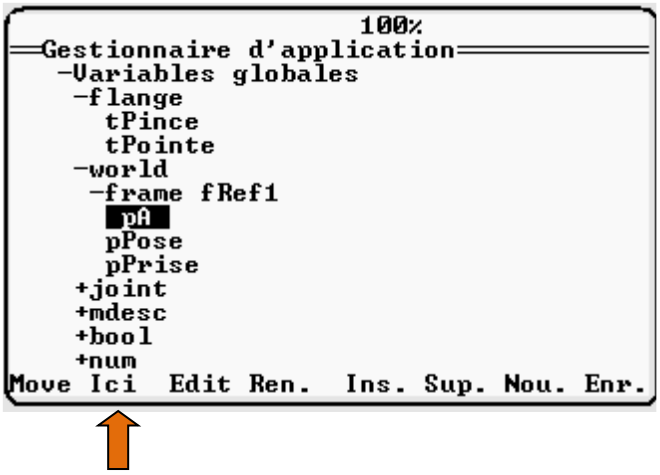
↳ Conseil :
Utiliser la pièce dans la pince pour reproduire l'apprentissage au mieux

Apprendre : - point d'origine
- Axe X
- Axe Y

- Points dans un frame

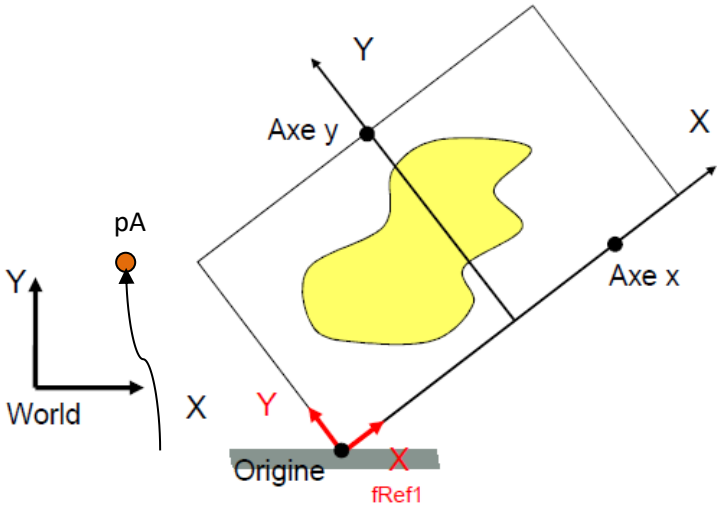
• Créer dans l'arbre du frame les points de l'application:

Lors de l'apprentissage du point, les coordonnées sont exprimées dans le repère du frame



• Pour le mouvement, il n'est pas nécessaire de spécifier le frame :

movej(pA ,tPince, mRapide)



Utilité : si la palette tombe, nous avons juste les points X, Y et Z à réapprendre, car les points du repère « fRef1 » sont enregistrés.

- Calcul d 'un frame par programme

nErreur = setFrame(pOrigine, pX,pY, fRepere)

Code d 'erreur de calcul :

3 points O, X, Y

Frame à calculer

0 : pas d 'erreur -1 : ptX trop proche de ptOrigine -2 : les 3 points sont trop alignés

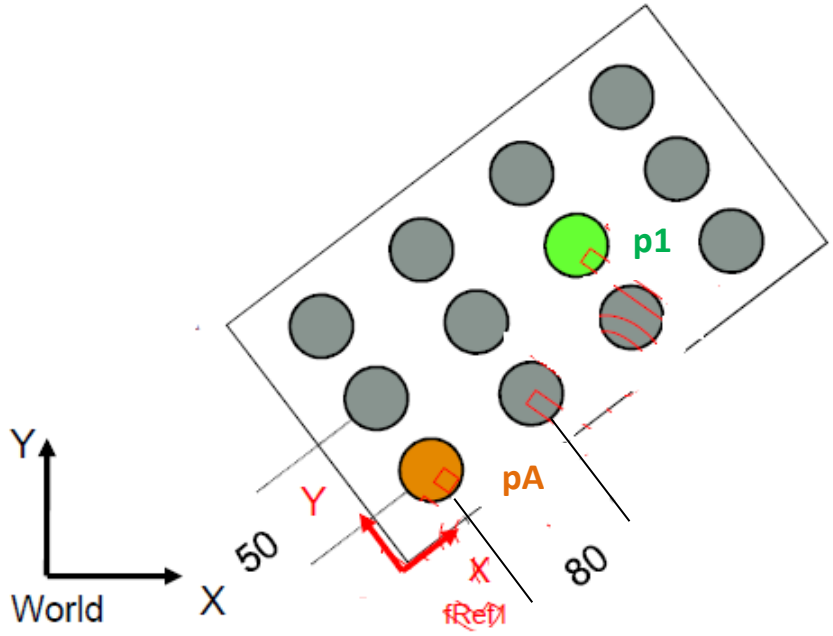
Utilité : interface pour changer d'outil

- Palettisation dans un frame

Compose(point,frame,trsf) : calcule un point décalé de trsf suivant le repère frame

p1=compose(pA,fRef1,{160,50,0,0,0,0}) ⇨ *On lui apprend le point A puis les axes X et Y du repère fRef1)*

movel(p1 ,tPince,mLent)



Ou : **movel(compose(pA,fRef1,{160,50,0,0,0,0}),tPince,mLent)**

compose est un décalage par rapport au repère frame courant

Ou : **movel(appro(pA,fRef1,{160,50,0,0,0,0}),tPince,mLent)**

appro est un décalage par rapport à l'outil courant

Application : Ex4

NB : Val3 7+: LINK pour récupérer pA dans un autre repère

- Librairies

But :

_ Utilisation de programmes/données dans de multiples applications (pour une application avec de nombreuses références génériques)

_ exemple :

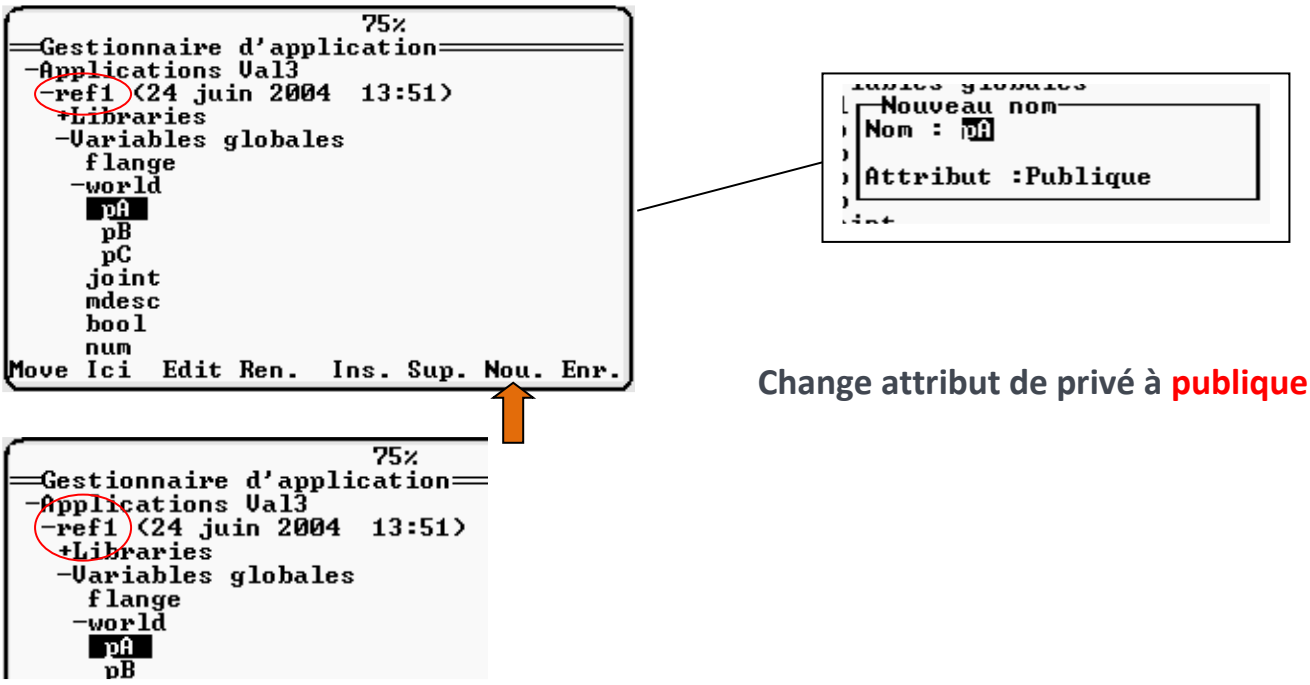
- Librairies de programmes pour réutiliser votre code source.
- Librairies PIECES pour utiliser une application unique avec plusieurs références de pièces
  => 1 application unique + 1 librairie de points pour chaque référence de pièce. (chaussures…)

_ Une librairie est une application normale, crée comme une simple application : depuis contrôleur CS8 ou émulateur CS8.

_ MAIS il faut déclarer les données/programmes qui seront exportés et utilisables depuis d'autres applications.

Utilité : Librairies sur serveur et le CS8 va chercher la librairies utile au pg ⇨lié le PC avec l'onglet Ftp
(en bas de l'écran du MCP : Edit. Sup. Ftp. Esc. Ok)

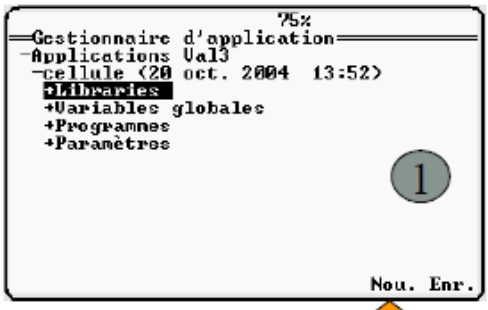
EXPORT DE DONNEES DE LA LIBRAIRIE

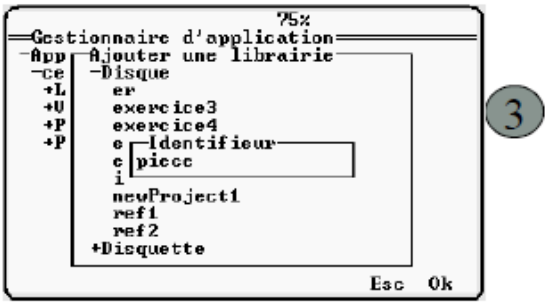


Exemple : Le point pA est public de la librairie ref1

IMPORT DE LIBRAIRIES

⚠ Les librairies utilisées dans une application doivent être déclarées.





Sélection de la librairie

Identifiant utilisé dans l'application qui importe la librairie

Exemple : La librairie **ref1** est utilisable dans l'application « **cellule** » avec l'identifiant « **piece** »

UTILISATION DES DONNEES

Utilisation d'un point pA défini publique dans la librairie REF1 :

pA de la librairie Ref1

```
movej(piece:pA, tPince, mRapide)
```

identifiant de la librairie

Travail sur une variable numérique nX définie publique dans la librairie REF1 :

```
piece:nX=10
```

Appel à un programme INIT défini publique dans la librairie REF1 :

```
call piece:init()
```

A l'édition du programme, le système garantit la cohérence des données et la validité de la syntaxe

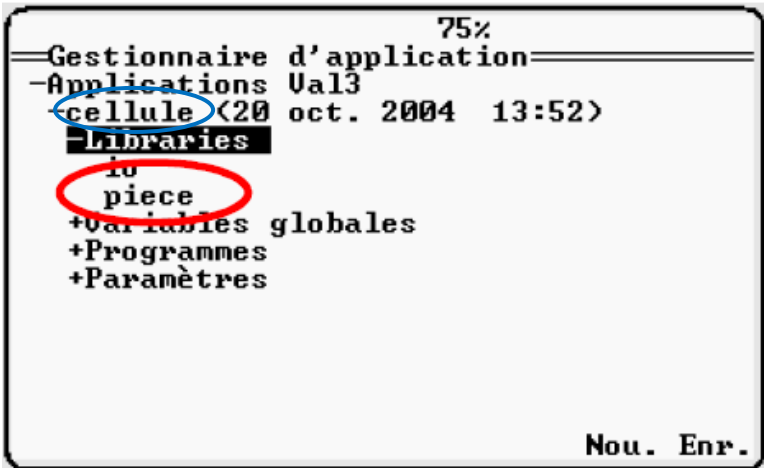
pA de Ref1 est différent de pA de Ref2

CHARGEMENT DE LIBRAIRIES

Exemple :

1 application avec plusieurs librairies de points pA est défini public dans ref1 et dans ref2 :

```
nErr=piece:libload("ref2")        charge ref2 a la place de ref1
movej(piece:pA,flange,nom_speed)  utilise pA de ref2
nErr=piece:libload("ref1")        charge ref1 a la place de ref2
movej(piece:pA,flange,nom_speed)  utilise pA de ref1
```



1 seul identifiant déclaré pour plusieurs librairies avec les mêmes données exportées mais avec des valeurs différentes

Exemple : « ref2 » a été créé en exportant « ref1 » sous « ref2 » puis « pta » de « ref2 » a été appris

NB : pas de limites en identifiant, ni en librairies

: si un pg est publique, toutes les variables du pg passerons publiques