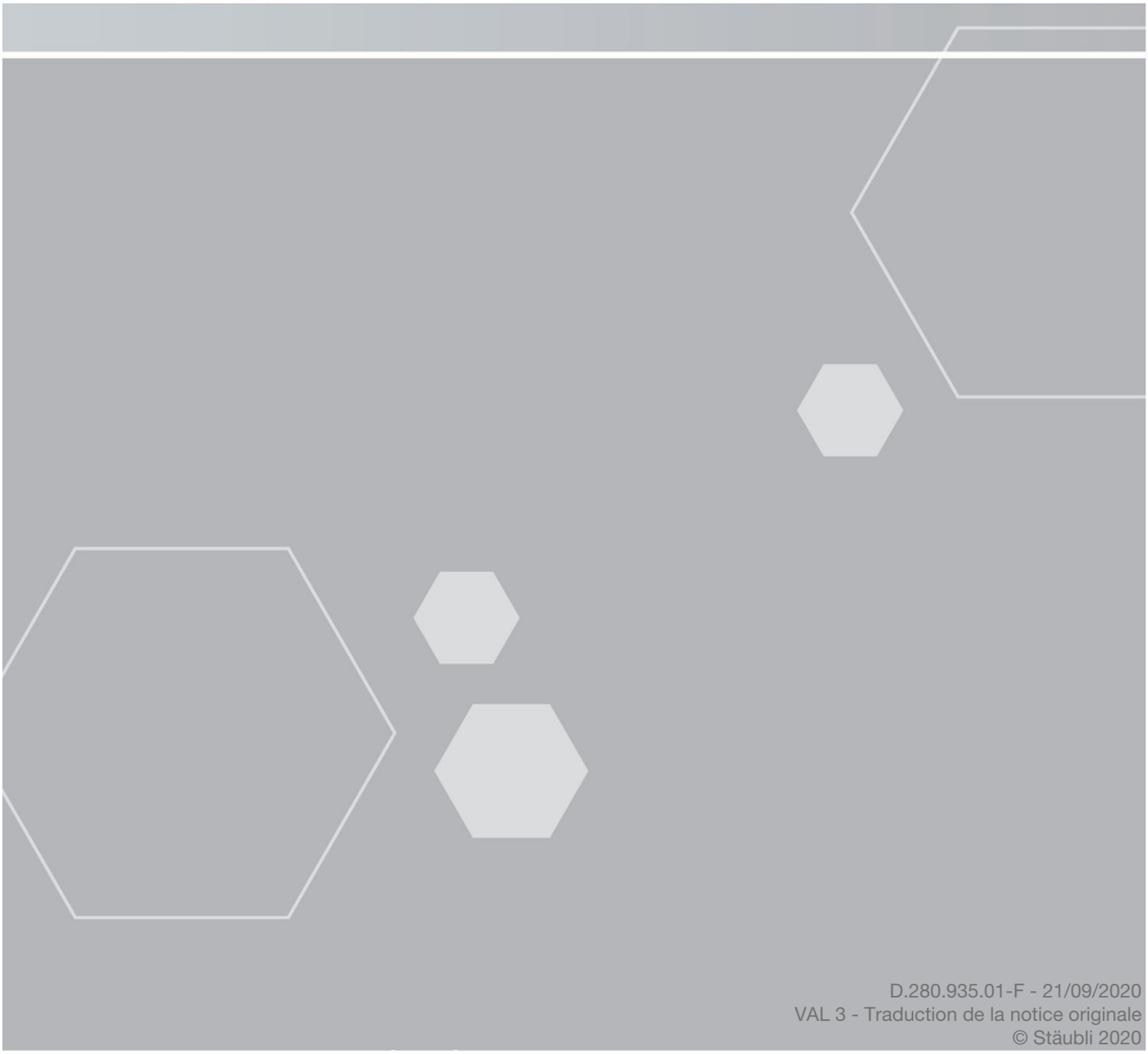
FAST MOVING TECHNOLOGY

STÄUBLI

Manuel de référence VAL 3

Version 8



D.280.935.01-F - 21/09/2020
VAL 3 - Traduction de la notice originale

Un document "readme.pdf" peut être livré avec le DVD du robot. Il contient des ajouts et errata, à la documentation.

TABLE DES MATIÈRES

STÄUBLI

M0005468.1

1 - INTRODUCTION

VAL 3 est un langage de programmation de haut niveau, destiné à la commande des robots Stäubli dans toutes sortes d'applications.

Le langage VAL 3 combine les caractéristiques de base d'un langage informatique temps réel standard de haut niveau et les fonctionnalités spécifiques du contrôle d'une cellule de robot industriel :

1) outils de contrôle du robot

2) outils de modélisation géométrique

3) outils de contrôle d'entrées sorties

Ce manuel de référence explique les notions indispensables à la programmation d'un robot, et détaille les instructions du langage VAL 3, classées suivant les catégories suivantes :

- Eléments du langage
- Types simples
- Interface utilisateur
- Tâches
- Librairies
- Types d'utilisateurs
- Contrôle du robot
- Position du bras
- Contrôle des mouvements

Chaque instruction, avec sa syntaxe, apparaît dans la table des matières pour une consultation rapide.

Les exemples de VAL 3 avec l'interface utilisateur graphique sont basés sur le modèle VAL3help_example fourni avec le SRC.

2 - ÉLÉMENTS DU LANGAGE VAL 3

M0005469.1

Le langage de programmation VAL 3 se compose d'applications. Une application VAL 3 contient à la fois des programmes et des données. Une application VAL 3 peut aussi faire référence à d'autres applications utilisées soit comme librairies, soit comme définitions de types d'utilisateurs.

2.1 - APPLICATIONS

M0005470.1

2.1.1 - DÉFINITION

M0005471.1

Une application VAL 3 est un logiciel autonome destiné à commander les robots et les entrées-sorties associées à un contrôleur.

Une application VAL 3 est constituée des éléments suivants :

- un ensemble de programmes : les instructions VAL 3 à exécuter
- un ensemble de données globales : les données partagées par tous les programmes de l'application
- un ensemble de librairies : applications externes utilisées pour partager des programmes et/ou des données
- un ensemble types d'utilisateurs : applications externes utilisées comme modèles pour définir des données structurées dans l'application
- un ensemble de MCP pages utilisateur : l'interface graphique de l'application à afficher sur le MCP

Lorsqu'une application est en cours d'exécution, elle contient également :

- un ensemble de tâches : les programmes exécutés simultanément

2.1.2 - CONTENU PAR DÉFAUT

M0005472.1

Pour créer une nouvelle application VAL 3, il faut copier le contenu d'une application prédéfinie servant de modèle. Il est possible de créer de nouveaux modèles personnalisés. Ceux-ci se composent simplement d'une application VAL 3 standard, placée dans un répertoire dédié dans le contrôleur.

Une application VAL 3 ne peut être lancée que si elle contient un programme **start**() et un programme **stop**(). Sans les programmes **start**() et **stop**(), une application VAL 3 ne peut être utilisée que comme librairie ou comme définition de type d'utilisateur. Il est possible de définir des applications contenant seulement des données ou seulement des programmes.

2.1.3 - DÉMARRAGE ET ARRÊT

M0005473.1

Le lancement d'une application VAL 3 est géré par le contrôleur. Il peut soit être demandé par l'utilisateur sur l'interface utilisateur du MCP, soit être automatique dans le cadre du processus de démarrage.

Une seule application VAL 3 peut être lancée à la fois. Elle peut cependant utiliser plusieurs autres applications en même temps (par exemple des librairies) et lancer plusieurs tâches d'exécution différentes.

lorsqu'une application VAL 3 est lancée, son programme **start**() s'exécute.

Une application VAL 3 s'arrête d'elle-même lorsque la dernière de ses tâches se termine : le programme **stop**() est alors exécuté. Toutes les tâches crées par des librairies, s'il en reste, sont détruites dans l'ordre inverse de leur création.

Si l'arrêt d'une application VAL 3 est provoqué depuis l'interface utilisateur du contrôleur MCP, la tâche de démarrage, si elle existe encore, est immédiatement détruite. Le programme **stop**() est ensuite exécuté, puis toutes les tâches de l'application, s'il en reste, sont détruites dans l'ordre inverse de l'ordre de leur création.

2.1.4 - PARAMÈTRES D'APPLICATION M0005474.1

Une application VAL 3 peut être configurée par les paramètres suivants :

- l'unité de longueur
- quantité de mémoire d'exécution

Ces paramètres sont accessibles à l'aide d'une instruction VAL 3 et ne peuvent être modifiés qu'à l'aide de l'interface utilisateur du MCP ou de VAL 3 Studio dans la Stäubli Robotics Suite.

2.1.4.1 - Unité de longueur M0005475.1

Dans les applications VAL 3, l'unité de longueur est le millimètre ou le pouce. Elle est utilisée par les types de donnée géométriques du VAL 3 : repère, point, joint (pour les axes linéaires), transformation, outil et lissage de trajectoire.

L'unité de longueur d'une application est définie lors de sa création par l'unité de longueur courante du système, et ne peut plus être modifiée ensuite.

L'unité de longueur (mm ou pouces) est une propriété de chaque module VAL 3 (applications, librairies, définitions de type d'utilisateur). L'unité de longueur, définie dans les librairies et les types d'utilisateur, utilisée dans une application doit correspondre à l'unité de longueur définie dans cette application.

En cas d'incohérence, un message avertit l'utilisateur que l'application a été ouverte avec une unité de longueur incohérente.

Lorsque l'incohérence est détectée par la fonction VAL 3 libLoad, la librairie est ouverte mais le code d'avertissement 1 est affiché.

Dans les deux cas, l'historique des événements fournit des informations concernant l'incohérence détectée.

2.1.4.2 - Quantité de mémoire d'exécution M0005811.1

Chaque tâche de VAL 3 a besoin de mémoire pour enregistrer :

- La pile d'exécution (liste des appels du programme exécutés pendant cette tâche)
- Les paramètres de chaque programme de la pile d'exécution
- Les variables locales pour chaque programme de la pile d'exécution

Par défaut, chaque tâche dispose de **20000** octets pour la mémoire d'exécution. Il n'est habituellement pas nécessaire de modifier ce paramètre.

Cette mémoire peut toutefois ne pas être suffisante pour les applications contenant de grands tableaux de variables locales ou des algorithmes récursifs :

elle doit alors être augmentée dans l'interface utilisateur MCP ou à l'aide de Stäubli Robotics Suite, ou bien l'application doit être optimisée en réduisant le nombre de programmes dans la pile d'appels ou en utilisant des variables globales au lieu de variables locales.

2.1.4.3 - Interface utilisateur graphique de l'application (pages utilisateur) M0005476.1

Pour pouvoir afficher et saisir des informations, une page graphique doit être définie. SRS doit être utilisé pour créer des pages utilisateur et pour lier des éléments graphiques aux variables VAL 3.

Consulter la documentation de SRS pour la procédure de liaison.

2.2 - PROGRAMMES

M0005477.1

2.2.1 - DÉFINITION

M0005478.1

Un programme est une séquence d'instructions VAL 3 à exécuter.

Un programme est constitué des éléments suivants :

- La séquence d'instructions : les instructions VAL 3 à exécuter
- Un ensemble de variables locales : les données internes au programme
- Un ensemble de paramètres : les données fournies au programme lors de son appel

Les programmes permettent de regrouper des séquences d'instructions susceptibles d'être utilisées à plusieurs endroits dans une application. Outre qu'ils font gagner du temps de programmation, ils simplifient aussi la structure des applications, facilitent la programmation et l'entretien et améliorent la lisibilité.

Le nombre d'instructions d'un programme est limité seulement par la place mémoire disponible dans le système.

Le nombre de variables locales et de paramètres n'est limité que par la taille de la mémoire d'exécution pour l'application (voir chapitre 2.1.4.2).

2.2.2 - RÉENTRANCE

M0005479.1

Les programmes sont réentrants, c'est-à-dire qu'ils peuvent s'appeler de façon récursive (instruction **call**) ou être appelés simultanément par plusieurs tâches. Chaque appel d'un programme utilise ses propres variables et paramètres locaux spécifiques. Aucune interaction n'est possible entre deux appels différents du même programme.

2.2.3 - PROGRAMME START()

M0005480.1

Le programme **start**() est appelé lorsque l'application VAL 3 est lancée. Il ne peut avoir de paramètres.

Ce programme contient habituellement toutes les opérations requises pour exécuter l'application : initialisation des variables globales et des sorties, création des tâches d'application, etc..

L'application ne s'arrête pas à la fin du programme **start**() si d'autres tâches de l'application sont encore en cours d'exécution.

Il est possible d'appeler le programme **start**() dans un programme (instruction **call**) comme n'importe quel autre programme.

2.2.4 - PROGRAMME STOP()

M0005481.1

Le programme **stop**() est le programme appelé lors de l'arrêt de l'application VAL 3. Il ne peut avoir de paramètres.

On trouvera typiquement dans ce programme toutes les opérations nécessaires pour terminer proprement l'application : réinitialisation des sorties et arrêt des tâches de l'application dans un ordre adéquat, etc.

Le programme **stop**() peut être appelé depuis un programme (instruction **call**) de la même manière que n'importe quel autre programme. L'appel du programme **stop**() n'arrête cependant pas l'application.

2.2.5 - INSTRUCTIONS DE CONTRÔLE DU PROGRAMME

M0005482.1

Comment //

M0005483.1

Syntaxe

// <String>

Fonction

Une ligne commençant par « // » n'est pas exécutée ; l'exécution reprend à la ligne suivante. Il ne faut pas utiliser « // » au milieu d'une ligne, il doit former les premiers caractères de la ligne.

Exemples

```
// This is an example of a comment
```

call program

M0005484.1

Syntaxe

call program([parameter1][,parameter2])

Fonction

L'instruction d'appel exécute un programme personnalisé. Le nombre et le type d'expressions après le nom du programme doivent concorder avec l'interface du programme. Les expressions définies comme des paramètres sont d'abord exécutées dans leur ordre de spécification. Les variables locales sont ensuite initialisées et l'exécution du programme commence par son instruction de début.

L'exécution d'un appel est terminée quand le programme exécute une instruction de retour ou de fin.

Exemples

```
// Calls the pick() and place() programs for i,j between 1 and 10
for i = 1 to 10
 for j = 1 to 10
   call pick(pPallet1[i,j])
   call place(pPallet2[i,j])
 endFor
endFor
```

return

M0005485.1

Syntaxe

return

Fonction

L'instruction de retour interrompt immédiatement l'exécution du programme en cours. Si ce programme a été appelé par un **call**, son exécution reprend après le **call** dans le programme d'appel. Sinon (si le programme est le programme **start**() ou le point de lancement d'une tâche), la tâche courante se termine. L'instruction de retour a exactement le même effet que l'instruction de fin à la fin du programme.

Un programme est souvent plus facile à comprendre et à entretenir quand son exécution se termine toujours par l'instruction de fin. L'utilisation d'une instruction de retour au milieu d'un programme n'est donc pas souhaitable.

if control instruction

M0005486.1

Syntaxe

if <bool bCondition>
** <instructions>**
[elseIf <bool bAlternateCondition1>
** <instructions>]**
../..
[elseIf <bool bAlternateConditionN>
** <instructions>]**
[else
** <instructions>]**
endIf

Fonction

La séquence **if...elseIf...else...endIf** évalue successivement les expressions booléennes marquées par les mots-clés **if** ou **elseIf** jusqu'à ce qu'une expression soit vraie. Les instructions suivant l'expression booléenne sont ensuite exécutées jusqu'au mot-clé **elseIf**, **else** ou **endIf** suivant. Le programme reprend finalement après le mot-clé **endIf**. Si toutes les expressions booléennes marquées par **if** ou **elseIf** sont fausses, les instructions comprises entre les mots-clés **else** et **endIf** sont exécutées (si le mot-clé **else** est présent). Le programme reprend ensuite après le mot-clé **endIf**.

Il n'y a pas de limites au nombre d'expressions **elseIf** dans une séquence **if...endIf**.

La séquence **if...elseIf...else...endIf** peut être remplacée par la séquence **switch...case...default...endSwitch** pour tester les différentes valeurs possibles d'une même expression.

Exemples

Ce programme convertit un jour écrit dans une string (**sDay**) en un num (**nDay**).

```
sOutput="Enter a day"
sInput=sDay
if sDay=="Monday"
 nDay=1
elseIf sDay=="Tuesday"
 nDay=2
elseIf sDay=="Wednesday"
 nDay=3
elseIf sDay=="Thursday"
 nDay=4
elseIf sDay=="Friday"
 nDay=5
else
 // Weekend !
 nDay=0
endIf
```

Voir aussi

switch

while control instruction

M0005487.1

Syntaxe

while <bool bCondition>
** <instructions>**
endWhile

Fonction

Les instructions comprises entre **while** et **endWhile** sont exécutées tant que l'expression booléenne **bCondition** est (**true**).

Si l'expression booléenne **bCondition** n'est pas vraie lors de la première évaluation, les instructions comprises entre **while** et **endWhile** ne sont pas exécutées.

Paramètres

bool bCondition	expression booléenne à évaluer

Exemples

```
// This simple program makes a signal flash as long as the robot is moving
diLamp = false
while (isSettled()==false)
 //Inverses the value of the diLamp: true false
 diLamp = !diLamp
 //Waits ½ s
 delay(0.5)
endWhile
diLamp = false
```

do ... until control instruction M0005488.1

Syntaxe

```
do
  <instructions>
until <bool bCondition>
```

Fonction

Les instructions comprises entre **do** et **until** sont exécutées jusqu'à ce que l'instruction booléenne **bCondition** soit (**true**).

Les instructions comprises entre **do** et **until** sont exécutées une fois si l'expression booléenne **bCondition** est vraie lors de sa première évaluation.

Paramètres

bool bCondition	expression booléenne à évaluer

for control instruction M0005489.1

Syntaxe

```
for <num nCounter> = <num nBeginning> to <num nEnd> [step <num nStep>]
  <instructions>
endFor
```

Fonction

Les instructions comprises entre **for** et **endFor** sont exécutées jusqu'à ce que le **nCounter** dépasse la valeur spécifiée pour **nEnd**.

Le **nCounter** est initialisé par la valeur **nBeginning**. Si le **nBeginning** dépasse **nEnd**, les instructions comprises entre **for** et **endFor** ne sont pas exécutées. A chaque itération, le **nCounter** est incrémenté de la valeur de **nStep**, et les instructions entre **for** et **endFor** sont de nouveau exécutées si le **nCounter** ne dépasse pas **nEnd**.

Si le **nStep** est positif, la boucle **for** s'arrête quand le **nCounter** est supérieur à **nEnd**. Si le **nStep** est négatif, la boucle **for** s'arrête quand le **nCounter** est inférieur à **nEnd**.

Paramètres

num nCounter	variable de type num utilisée comme compteur
num nBeginning	expression numérique d'initialisation du compteur
num nEnd	expression numérique de test de fin de boucle
[num nStep]	expression numérique d'incrément du compteur

Exemples

```
// This program rotates axis 1 from -90° to +90° in -10° steps
for nPos = 90 to -90 step -10
 jDest.j1 = nPos
 movej(jDest, flange, mNomSpeed)
 waitEndMove()
endFor
```

Syntaxe

switch <expression>
case <value1> [, <value2>]
<instructions1-2>
break
[case <value3> [, <value4>]
<instructions3-4>
break]
[default
<Default Instructions>
break]
endSwitch

Fonction

La séquence **switch...case...default...endSwitch** évalue successivement les expressions signalées par le mot-clé **case** jusqu'à ce qu'une expression soit égale à l'expression initiale après le mot-clé **switch**.

Les instructions suivant l'expression sont ensuite exécutées, jusqu'au mot-clé **break**. Le programme reprend finalement après le mot-clé **endSwitch**.

Si aucune expression **case** n'est égale à l'expression **switch** initiale, les instructions comprises entre les mots-clés **default** et **endSwitch** sont exécutées (si le mot-clé **default** est présent).

Il n'y a pas de limites au nombre d'expressions **case** dans une séquence **switch...endSwitch**. Les expressions suivant le mot-clé **case** doivent être du même type que celles suivant le mot-clé **switch**.

La séquence **switch...case...default...endSwitch** est très semblable à la séquence **if...elseIf...else...endIf**.

Elle accepte non seulement les expressions booléennes, mais aussi tout type d'expressions acceptant l'opérateur standard "is equal to""==".

Exemples

Ce programme lit un num (**nMenu**) correspondant à une séquence de touches et modifie un string s en conséquence.

```
nMenu = sInput
switch nMenu
 case 271
  s = "Menu 1"
 break
 case 272
  s = "Menu 2"
 break
 case 273, 274, 275, 276, 277, 278
  s = "Menu 3 to 8"
 break
 default
  s = "this key is not a menu key"
 break
endSwitch
```

Ce programme convertit un jour écrit dans une string (**sDay**) en un num (**nDay**).

```
sOutput="Enter a day: "
 sDay=sInput
 switch sDay
   case "Monday"
     nDay=1
   break
   case "Tuesday"
     nDay=2
   break
   case "Wednesday"
     nDay=3
   break
   case "Thursday"
     nDay=4
   break
   case "Friday"
     nDay=5
   break
   default
     // Not a week day !
     nDay=0
   break
 endSwitch
```

2.3 - VARIABLES

M0005490.1

2.3.1 - DÉFINITION

M0005491.1

Une variable est un ensemble de valeurs à utiliser comme paramètre ou résultat d'instructions de VAL 3.

Les variables se composent des éléments suivants :

- un ensemble de valeurs
- un type définissant les valeurs possibles et les opérations autorisées sur les données. Les types de données les plus simples sont Booléen, Numérique et String
- un conteneur définissant la manière dont les valeurs sont stockées dans les données. Les conteneurs de données possibles dans VAL 3 sont Elément, Tableau et Collection

2.3.2 - TYPES SIMPLES

M0005492.1

Le langage VAL 3 supporte les types simples suivants :

- Type bool : pour les valeurs booléennes (vrai / faux)
- Type num : pour les valeurs numériques (nombres entiers ou à virgule flottante)
- Type string : pour les chaînes de caractères (caractères Unicode)
- Type dio : pour les entrées-sorties numériques
- Type aio : pour les entrées-sorties numériques (analogiques ou digitales)
- Type sio : pour les ports d'entrée/sortie et SocketsEthernet

Le type de variable est indiqué dans la documentation par les premières lettres de son nom en minuscule :

- **bVariable** est une variable de type bool
- **nVariable** est une variable de type num
- **sVariable** est une variable de type string
- **diVariable** est une variable de type dio
- **aiVariable** est une variable de type aio
- **siVariable** est une variable de type sio

2.3.3 - TYPES STRUCTURÉS

M0005493.1

Un type structuré combine plusieurs types plus simples dans un nouveau type de niveau plus élevé. Chaque sous-type reçoit un nom et devient accessible individuellement en tant que champ de la structure. Les types adéquats dans une application organisent les données de façon à faciliter les calculs et les évolutions du programme.

Le langage VAL 3 prend en charge les types structurés suivants, composés de types simples :

- Type trsf : pour les transformations géométriques cartésiennes
- Type frame : pour les repères géométriques cartésiens
- Type tool : pour les outils montés sur un robot
- Type point : pour les positions cartésiennes d'un outil
- Type joint : pour les positions articulaires du robot
- Type config : pour les configurations du robot
- Type mdesc : pour les paramètres de déplacement du robot

Le langage VAL 3 prend aussi en charge des types 'Utilisateur' combinant des types VAL 3 simples, structurés ou même d'autres types 'Utilisateur' dans un nouveau type. Un type 'Utilisateur' peut être utilisé dans une application de la même manière qu'un type standard.

Le type de variable est indiqué dans la documentation par les premières lettres de son nom en minuscule :

- **trVariable** est une variable de type trsf
- **fVariable** est une variable de type frame
- **tVariable** est une variable de type tool
- **pVariable** est une variable de type point
- **jVariable** est une variable de type joint
- **cVariable** est une variable de type config
- **mVariable** est une variable de type mdesc

2.3.4 - CONTENEURS

M0005494.1

Le conteneur de données définit la manière dont les valeurs sont stockées dans les données :

- Un conteneur "élément" se compose d'une valeur unique. True (booléen), 0 (numérique) ou 'text' (string) ont un conteneur de ce type.
- Un conteneur "tableau" se compose d'un ensemble de valeurs identifiées par 1, 2 ou 3 indices entiers. L'indice initial dans les tableaux est toujours 0.
- Un conteneur "collection" se compose d'un ensemble de valeurs identifié par une clé représentée par une chaîne de caractères. Une chaîne de caractères non vide peut être utilisée comme identifiant de valeur.

Un conteneur tableau à une dimension avec une seule valeur (index 0) est considéré comme un conteneur élément.

Lorsque cela est nécessaire dans la documentation, le conteneur de la variable est identifié avec le nom de celle-ci :

- **s1dArray** est un tableau à une dimension du type string
- **s2dArray** est un tableau à deux dimensions du type string
- **s3dArray** est un tableau à trois dimensions du type string
- **sColl** est une collection du type string

Certaines instructions (pour la gestion des tableaux ou des collections) sont indifférentes au type des données. Le type est alors remplacé par un astérisque dans la documentation : '*'.

2.4 - INITIALISATION DES DONNÉES M0005495.1

2.4.1 - DONNÉES DE TYPE SIMPLE M0005496.1

La syntaxe pour l'initialisation d'une donnée de type simple est indiquée dans le chapitre décrivant chaque type simple. Un tableau ou une collection doit être initialisé élément par élément. La valeur d'initialisation est **false** pour un bool, 0 pour un num et "" (chaîne vide) pour une chaîne.

Exemples

Dans cet exemple, **bBool** est une variable booléenne, **nPi** une variable numérique et **sString** une variable de chaîne.

```
bBool = true
nPi = 3.141592653
sString = "this is a string"
```

2.4.2 - DONNÉES DE TYPE STRUCTURÉ M0005497.1

La valeur d'une donnée de type structurée est définie par la séquence des valeurs des champs entre parenthèses {}, séparées par des virgules. Les valeurs des champs vides sont remplacées par 0. L'ordre de la séquence est spécifié dans le chapitre détaillant chaque type structuré. La valeur d'une structure peut inclure des valeurs ou d'autres sous-structures imbriquées. Un tableau ou une collection d'un type structuré doit être initialisé élément par élément.

Exemples

Le type point est fait d'un trsf et d'un type config. Une variable point peut être initialisée de la manière représentée :

```
pPosition = {{100, -50, 200, 0, 0, 0}, {sfree, efree, wfree}}
```

La transformation du point pourrait aussi être initialisée avec :

```
pPosition.trsf = {100, -50, 200,,,}
```

2.5 - VARIABLES M0005498.1

2.5.1 - DÉFINITION M0005499.1

Une variable est une donnée référencée par son nom dans une application ou un programme.

Une variable est caractérisée par :

- un nom : une chaîne de caractères
- une portée : là où la variable est accessible (dans un seul programme, partagée par des programmes dans une application ou partagée entre des applications)
- un ensemble de valeurs
- un type de données (type simple ou structuré)
- un conteneur de variables (élément, tableau ou collection)

Un nom de variable est une chaîne de caractères sélectionnés parmi **"a..zA..Z0..9_"** et commençant par une lettre.

2.5.2 - PORTÉE D'UNE VARIABLE M0005500.1

La portée d'une variable peut être :

- globale : tous les programmes de l'application peuvent utiliser la variable, ou
- locale : la variable n'est accessible qu'au programme dans lequel elle est déclarée

Lorsqu'une variable globale et une variable locale ont le même nom, le programme où la variable locale est déclarée utilisera la variable locale, et n'aura pas accès à la variable globale.

Quand une application est utilisée comme librairie, chaque variable globale peut être déclarée comme publique ou privée. Une variable publique est accessible aux applications utilisant la librairie ; une variable privée n'est accessible qu'à l'intérieur de la librairie.

2.5.3 - ACCÈS À LA VALEUR D'UNE VARIABLE

M0005501.1

L'accès à la valeur d'une variable dépend de son conteneur :

- la valeur d'un conteneur élément est accessible par le nom de la variable (sans crochets) : nVariable.
- une valeur dans un tableau est accessible par ses indices numériques entre crochets carrés après le nom de la variable : n1dArray[nIndex], n2dArray[nIndex1, nIndex2], n3dArray[nIndex1, nIndex2, nIndex3].
- une valeur d'une collection est accessible par sa clé entre crochets après le nom de la variable : nCollection[sKey]

Un conteneur tableau à une dimension avec une seule valeur (index 0) est considéré comme un conteneur élément. Il est possible d'accéder à sa valeur sans crochets : n1dArray est équivalent à n1dArray[0].

Les indices numériques utilisés pour accéder à une valeur dans un tableau sont arrondis au nombre entier le plus proche : n2dArray[5.01, 6.99] est équivalent à n2dArray[5, 7]

L'indice utilisé pour accéder à une valeur dans un tableau est compris entre 0 et la taille de la dimension moins 1.

Les champs d'une variable de type structuré sont accessibles à l'aide d'un '.' suivi du nom du champ : pPoint.trsf.x renvoie à la valeur du champ 'x' ou du champ 'trsf' de la donnée point pPoint.

Exemples

Initialisation de variables de type simple avec différents conteneurs :

```
nPi = 3.141592653
sMonth[0] = "January"
sProductName["D 243 064 40 A"] = " CdRom"
```

2.5.4 - INSTRUCTIONS VALABLES POUR TOUTES LES VARIABLES

M0005502.1

num **size**(*)

M0005503.1

Fonction

Cette instruction renvoie le nombre de valeurs accessibles avec la variable :

- la taille d'une variable de conteneur élément est 1.
- la taille d'un tableau à une dimension est le nombre d'éléments dans le tableau.
- la taille d'une collection est le nombre d'éléments qu'elle contient.

Pour les tableaux à deux et trois dimensions, l'instruction de taille nécessite un deuxième paramètre pour spécifier la dimension de taille. Pour un tableau à une dimension, size(**s1dArray**) est équivalent à size(**s1dArray**, 1).

La taille d'un paramètre de tableau à une dimension donnée par une référence de tableau dépend de l'index spécifié lors de l'appel du programme. Seule la partie du tableau qui commence à partir de l'index spécifié est accessible dans le sous-programme : size(**s1dArray**[**nIndex**]) = size(**s1dArray**) - **nIndex**.

Paramètres

variable	variable de type quelconque

Exemples

La variable de taille doit être spécifiée sans crochets :

```
// Ok
nNbElements=size(sCollection)
// compilation error: unexpected key
nNbElements=size(sCollection[sKey])
```

Pour un tableau à une dimension, un index indique le début d'un sous-tableau : size(**s1dArray**[**nIndex**]) est la taille du sous-tableau qui commence à l'index **nIndex**.

bool **isDefined**(*) M0005823.1

Fonction

Cette instruction renvoie **true** si l'élément spécifié est défini dans un tableau ou une collection, ou **false** si l'élément n'est pas défini.

Elle peut être utilisée pour tester si un élément est défini dans une collection ; elle peut aussi être utilisée pour tester si une librairie définit une variable de son interface ou pas. Cette fonction est utile pour gérer l'évolution de l'interface d'une librairie et adapter son utilisation selon s'il s'agit d'une version récente ou ancienne de l'interface.

Exemples

Cet exemple ajoute une nouvelle clé d'article dans une collection.

```
// Get reference name from user input
sReference=sInput
if isDefined(sReferenceColl[sReference])==true
 sOutput="Error: reference already defined"
else
 // Add new article in the collection
 insert(sReferenceColl[sReference])
endIf
```

L'exemple suivant teste l'interface d'une librairie.

```
// Load part library
nLoadCode = part:libLoad(sPartPath)
// part:sVersion was not defined in the first version of the library
// Test if this library defines it
if (nLoadCode==0) or (nLoadCode==11)
  if (isDefined(part:sVersion)==false)
   // initial version
   sLibVersion = "v1.0"
  else
   sLibVersion = part:sVersion
  endIf
endIf
```

Fonction

Cette instruction crée une nouvelle valeur du type de la variable et l'enregistre dans le conteneur de variables. La nouvelle valeur est initialisée avec la valeur par défaut du type. La taille de la variable est augmentée d'un.

Pour un tableau à une dimension, la nouvelle valeur est insérée dans la position d'index spécifiée. La position d'index peut être égale à la taille du tableau : l'insertion est alors effectuée en fin de tableau. "insert(**s1dArray**[size(**s1dArray**)])" est équivalent à “append(**s1dArray**)*?.

Pour les collections, l'insertion n'est acceptée que si la clé n'est pas déjà utilisée dans la collection. La nouvelle valeur est associée à la clé spécifiée et la fonction renvoie **true**. L'instruction est sans effet et renvoie **false** si la clé était déjà utilisée. L'instruction isDefined() peut être utilisée pour vérifier si une clé est ou non utilisée dans une collection.

Cette instruction n'est pas utilisable pour les tableaux à deux et trois dimensions (utiliser à la place l'instruction resize()) ni pour les tableaux de variables locales. La taille d'une variable est limitée à 9999 valeurs. Une erreur d'exécution est générée quand la taille de la variable dépasse cette limite.

L'instruction d'insertion attribue de la mémoire système. La performance de l'attribution de mémoire n'est pas garantie. Il vaut donc mieux éviter l'usage fréquent d'insert() dans les applications VAL 3 où la performance est critique.

Exemples

Cet exemple ajoute un nouvel article à une liste.

```
// sInput in the format articleName,position
nPos=find(sInput,",")
// Define new article name
sArticleName=left(sInput,nPos)
// Define the position in the list
toNum(left(sInput,nPos), nIndex, bOk)
if (nIndex<0) or (nIndex>size(sArticleList))
 sOutput="Error: invalid position"
else
 // Add new article in the list
 insert(sArticleList[nIndex])
 sArticleList[nIndex] = sArticleName
endIf
```

Cet exemple ajoute une nouvelle clé d'article dans une collection.

```
// Ask for a new article name
sArticleName=sInput
if isDefined(sArticleColl[sArticleName])==true
 sOutput="Error: reference already defined"
else
 // Ask new article in the collection
 insert(sArticleColl[sArticleName])
endIf
```


Fonction

Cette instruction supprime la valeur spécifiée du conteneur de la variable. La taille de la variable est réduite d'un.

Une erreur d'exécution est générée si l'index ou la clé spécifiés dépassent les limites. L'instruction isDefined() peut être utilisée pour vérifier si une clé est ou non utilisée dans une collection.

La taille d'une collection peut être nulle, mais une variable de tableau doit toujours avoir au moins un élément. Une erreur d'exécution est générée si l'on essaie de supprimer le dernier élément d'un tableau.

Cette instruction n'est pas utilisable pour les tableaux à deux et trois dimensions (utiliser à la place l'instruction resize()) ni pour les tableaux de variables locales.

L'instruction delete() libère de la mémoire système. Les performances du recueil des données effacées ne sont pas garanties. Il vaut donc mieux éviter l'usage fréquent d'delete() dans les applications VAL 3 où la performance est critique.

Exemples

Cet exemple supprime un article dans une collection.

```
// Define article to delete
sArticleName=sInput
if isDefined(sArticleColl[sArticleName])==true
 // remove the article from the collection
 delete(sArticleColl[sArticleName])
else
 sOutput="Error: article not defined"
endIf
```

num **getData**(string sDataName, *) M0005506.1

Fonction

Cette instruction copie la valeur des données publiques globales, indiquée par leur nom **sDataName**, dans la variable spécifiée. Si la donnée et la variable sont toutes deux des tableaux à une dimension, l'instruction getData() copie toutes les entrées du tableau jusqu'à la fin de celui-ci. L'instruction renvoie le nombre d'entrées copiées dans la variable.

Le nom des données doit avoir le format suivant : "library:name[index]", où "library:" et "[index]" sont facultatifs :

- "name" est le nom de la donnée
- "library" est le nom de l'identifiant de librairie où la donnée est définie
- "index" est la valeur numérique de l'index auquel il faut accéder quand la donnée est un tableau à une dimension

L'instruction renvoie un code d'erreur lorsque les données n'ont pas pu être copiées :

Valeur renvoyée	Description
n > 0	La variable a bien été actualisée avec n entrées copiées
-1	Les données n'existent pas ou ne sont pas des données publiques globales
-2	L'identifiant de librairie n'existe pas
-3	L'index est hors limites
-4	Le type de donnée ne correspond pas au type de variable
-5	La librairie n'est pas chargée

Exemples

Ce programme fusionne 2 tableaux de points pApproach[] et pTrajectory[] provenant d'une librairie en un seul tableau local pPath[].

```
// Copy approach points in path
i = getData("Part:pApproach", pPath)
if(i > 0)
 nPoints = i
 // Append trajectory points in path
 i = getData("Part:pTrajectory", pPath[nPoints])
 if(i >0)
 nPoints=nPoints+i
 endIf
endIf
```

2.5.5 - INSTRUCTIONS SPÉCIFIQUES AUX TABLEAUX DE VARIABLES M0005507.1

void **append**(*) M0005508.1

Fonction

Cette instruction crée une nouvelle valeur du type de la variable et l'enregistre à la fin du tableau de variable à une dimension. La nouvelle valeur est initialisée avec la valeur par défaut du type. La taille de la variable est augmentée d'un.

Cette instruction n'est pas utilisable pour les tableaux à deux et trois dimensions ni pour les tableaux de variables locales. La taille d'une variable est limitée à 9999 valeurs. Une erreur d'exécution est générée quand la taille de la variable dépasse cette limite.

L'instruction d'ajout affecte de la mémoire système. La performance de l'attribution de mémoire n'est pas garantie. Il vaut donc mieux éviter l'usage fréquent d'append() dans les applications VAL 3 où la performance est critique.

Exemples

Cet exemple joint un nouvel article dans une liste.

```
// Ask for a new article name
sArticle= sInput
append(sArticleList)
sArticleList[size(sArticleList)-1] = sArticle
```

num **size**(*, num nDimension) M0005509.1

Fonction

Cette instruction renvoie la taille de la dimension spécifiée dans le tableau. Si nDimension dépasse les dimensions du tableau, la fonction renvoie 0. Pour un tableau à une dimension, size(**s1dArray**, 1) est équivalent à size(**s1dArray**).

Exemples

Le tableau de variable doit être indiquée sans crochets :

```
//Ok
nNbElements=size(s3dArray,3)
// Compilation error: unexpected indices
nNbElements=size(s3dArray[1,2,3],3)
```

void **resize**(*, num nDimension, num nSize) M0005510.1

Fonction

Cette instruction crée ou supprime des valeurs dans un tableau de telle façon que la taille de la dimension spécifiée corresponde à la valeur nSize. La création ou la suppression de la valeur s'effectue à la fin du tableau. Le cas échéant, les nouvelles valeurs sont initialisées avec la valeur par défaut du type.

Cette instruction n'est pas utilisable pour les tableaux de variables locales. La taille d'une variable est limitée à 9999 valeurs. Une erreur d'exécution est générée quand la taille de la variable dépasse cette limite.

L'instruction resize() affecte ou libère de la mémoire système. La performance de gestion de la mémoire n'est pas garantie. Il vaut donc mieux éviter l'usage fréquent d'resize() dans les applications VAL 3 où la performance est critique.

Exemples

La variable doit être définie sans index. L'instruction suivante modifie s2dArray de telle façon que sa seconde dimension soit 5.

```
resize(s2dArray, 2, 5)
```

2.5.6 - INSTRUCTIONS SPÉCIFIQUES AUX COLLECTIONS DE VARIABLE M0005511.1

string **first**(*) M0005512.1

Fonction

Cette instruction renvoie la première clé d'une collection dans l'ordre alphabétique. Si la collection est vide, l'instruction renvoie une chaîne vide "".

string **next**(*) M0005513.1

Fonction

Cette instruction renvoie la clé suivante d'une collection dans l'ordre alphabétique. Si la clé spécifiée est la dernière de la collection, l'instruction renvoie la chaîne vide "".

Exemples

Cet exemple affiche tous les éléments d'une collection sur la page utilisateur dans l'ordre alphabétique :

```
sKey = first(sCollection)
while sKey != ""
 sKey = next(sCollection[sKey])
 sOutput=sKey
endWhile
```

string **last**(*) M0005514.1

Fonction

Cette instruction renvoie la dernière clé d'une collection dans l'ordre alphabétique. Si la collection est vide, l'instruction renvoie une chaîne vide "".

Fonction

Cette instruction renvoie la clé précédente d'une collection dans l'ordre alphabétique. Si la clé spécifiée est la première de la collection, l'instruction renvoie une chaîne vide "".

Exemples

Cet exemple affiche tous les éléments d'une collection sur la page utilisateur dans l'ordre alphabétique inverse des clés :

```
sKey = last(sCollection)
while sKey != ""
 sKey = prev(sCollection[sKey])
 sOutput=sKey
endWhile
```

2.6 - PARAMÈTRES DU PROGRAMME M0005516.1

Les paramètres de sous-programme sont des données transmises d'un programme qui appellent un sous-programme avec l'instruction d'appel. Dans le sous-programme, les paramètres sont comme des variables locales initialisées automatiquement quand le sous-programme est lancé.

Il existe différentes manières de faire passer une variable à un sous-programme :

- on peut ne faire passer qu'une valeur (un élément) de la variable ou le conteneur (tableau ou collection) de la variable dans son ensemble.
- on peut laisser le sous-programme modifier la valeur de la variable (en transmettant la variable "par référence") ou faire passer simplement une copie de la valeur au sous-programme, en laissant la variable inchangée (transmission d'une variable "par valeur").

Une variable peut être transmise comme paramètre :

- par valeur de l'élément.
- par référence à l'élément.
- par référence à un tableau ou à une collection.

Il n'est pas permis de faire passer un conteneur (tableau ou collection) par valeur.

Une référence est notée d'un symbole ’&’ après le type de données dans la définition de l'interface du programme :

num& nData est un paramètre num transmis par référence à l'élément.

num& n1dArray[] est un num transmis par référence au tableau (tableau à une dimension).

num& n2dArray[,] est un paramètre num transmis par référence au tableau (tableau à deux dimensions).

num& n3dArray[,,] est un paramètre num transmis par référence au tableau (tableau à trois dimensions).

num& nCollection[""] est un paramètre num transmis par référence à la collection.

La même notation est utilisée dans la documentation pour la description des instructions :

bool pointToJoint(tool tTool, joint jInitial, point pPosition, joint& jResult) est une instruction renvoyant une valeur booléenne, prenant un outil, une position articulaire et un point transmis par valeur de l'élément et une position articulaire transmise par référence à l'élément.

num fromBinary(num& nDataByte[], num nDataSize, string sDataFormat, num& nValue[]) est une instruction renvoyant une valeur numérique, prenant un tableau à une dimension comme premier paramètre (transmis par référence), une donnée numérique et une donnée de chaîne transmises par valeur d'élément et un tableau à une dimension comme dernier paramètre (transmis par référence).

2.6.1 - PARAMÈTRE PAR VALEUR D'ÉLÉMENT

M0005517.1

Quand un paramètre est défini par la valeur de l'élément, le système crée une variable locale et l'initialise avec la valeur de l'instruction VAL 3 fournie par le programme d'appel. Si l'instruction fournie est une variable, le paramètre est initialisé avec une copie de la valeur de la variable. Aucune modification apportée à la valeur du paramètre dans le sous-programme n'a d'effet sur celle de la variable dans le programme d'appel.

Exemples

Supposons que sendMessage(string sMessage) est un programme avec un seul paramètre transmis par la valeur de l'élément.

sendMessage() peut être utilisé avec une donnée constante ou le résultat du calcul :

```
call sendMessage("Waiting for signal StartCycle")
call sendMessage("Waiting for signal"+sSignalName)
```

sendMessage() peut être utilisé avec des valeurs d'éléments, de tableaux ou de collections :

```
call sendMessage(sMessage)
call sendMessage(sMessageArray[23])
call sendMessage(s2dArray[12,3])
call sendMessage(s3dArray[5,7,9])
call sendMessage(sMessageColl[sMessageName])
```

Après ces appels, la valeur de sMessage, sMessageArray[23], s2dArray[12,3], s3dArray[5,7,9], sMessageColl[sMessageName] n'a pas été modifiée par les instructions dans sendMessage().

2.6.2 - PARAMÈTRE PAR RÉFÉRENCE D'ÉLÉMENT

M0005518.1

Quand un paramètre est défini par une référence d'élément, le système crée une variable locale et l'initialise avec un lien vers la donnée fournie par le programme d'appel. La variable transmise par référence peut avoir un conteneur élément, tableau ou collection mais seule la valeur indiquée dans l'appel est transmise au sous-programme. Le conteneur du paramètre est toujours un élément. Toutes les modifications apportées à la valeur du paramètre dans le sous-programme affectent directement la valeur correspondante dans les données du programme d'appel. Il n'est pas possible de transmettre une constante VAL 3 ou le résultat d'une expression VAL 3 par référence d'élément.

Exemples

Supposons que sendMessage(string& sMessage) est un programme avec un seul paramètre transmis par référence d'élément.

sendMessage() ne peut pas être utilisé avec une constante ou le résultat d'un calcul :

```
// compilation errors: variable expected as parameter
call sendMessage("Waiting for signal StartCycle")
call sendMessage("Waiting for signal"+sSignalName)
```

sendMessage() peut être utilisé avec des valeurs d'éléments, de tableaux ou de collections :

```
call sendMessage(sMessage)
call sendMessage(sMessageArray[23])
call sendMessage(s2dArray[12,3])
call sendMessage(s3dArray[5,7,9])
call sendMessage(sMessageColl[sMessageName])
```

Après ces appels, la valeur de sMessage, sMessageArray[23], s2dArray[12,3], s3dArray[5,7,9], sMessageColl[sMessageName] peut avoir été modifiée par les instructions de sendMessage().

2.6.3 - PARAMÈTRE PAR RÉFÉRENCE DE TABLEAU OU DE COLLECTION

Quand un paramètre est défini par une référence de tableau ou de collection, le système crée une variable locale et l'initialise avec un lien vers la donnée fournie par le programme d'appel. Le conteneur du paramètre dans le sous-programme est le même que celui de la variable fournie : un tableau à une, deux ou trois dimensions ou une collection. Tous les changements apportés à une valeur du paramètre dans le sous-programme affectent directement la valeur correspondante de la donnée du programme qui l'appelle.

Avec les tableaux à une dimension, il est possible de transmettre seulement une partie du tableau au sous-programme, en indiquant le premier élément accessible. Pour les tableaux à deux et trois dimensions et les collections, il n'est pas possible de transmettre une partie seulement du tableau ou de la collection au sous-programme. La variable doit alors être transmise sans crochets [ ] indiquant un index ou une clé. Il n'est pas possible de transmettre une constante VAL 3 ou le résultat d'une expression VAL 3 par référence de tableau ou de collection.

Exemples

Supposons que send1dMessage(string& s1dArray[ ]) est un programme à un seul paramètre transmis par référence de tableau (une dimension).

Supposons que send2dMessage(string& s2dArray[ ]) est un programme à un seul paramètre transmis par référence de tableau (deux dimensions).

Supposons que send3dMessage(string& s3dArray[ ] est un tableau à un paramètre transmis par référence de tableau (trois dimensions).

Supposons que sendCollMessage(string& sMessageColl[""]) est un programme à un seul paramètre transmis par référence de collection.

Aucun de ces programmes ne peut être utilisé avec une constante ou le résultat d'un calcul :

```
// compilation errors: array variable expected as parameter
call send1dMessage("Waiting for signal StartCycle")
call send1dMessage("Waiting for signal"+l_sSignalName)
```

Le conteneur de la variable transmise doit corresponndre au conteneur déclaré du paramètre :

```
// compilation errors: 1d array variable expected
call send1dMessage(sMessageColl)
call send1dMessage(s2dArray[12,3])
// compilation error: collection variable expected
call sendCollMessage(sMessage)
```

Il n'est pas possible de transmettre une partie d'un tableau ou d'une collection, sauf pour les tableaux à une dimension. Les tableaux et les collections doivent être indiqués sans indice ni clé.

```
// correct parameter
call send2dMessage(s2dArray)
call send3dMessage(s3dArray)
call sendCollMessage(sMessageColl)
call send1dMessage(sMessageArray)
call send1dMessage(sMessageArray[23])
// compilation errors: unexpected indices for the array
call send2dMessage(s2dArray[12,3])
call send3dMessage(s3dArray[5,7,9])
// compilation error: unexpected indices for the collection
call sendCollMessage(sMessageColl[l_sMessageName])
```

Dans ce dernier cas, seule la partie du tableau sMessageArray commençant à l'index 23 est transmise au programme send1dMessage(). Les valeurs de sMessageArray ayant un index inférieur à 23 sont inaccessibles à send1dMessage().

3 - TYPES SIMPLES

M0005520.1

3.1 - TYPE BOOL

M0005521.1

3.1.1 - DÉFINITION

M0005522.1

Les valeurs possibles des variables ou constantes de type bool sont :

- **true** : valeur vraie
- **false** : valeur fausse

Par défaut, une variable de type bool est initialisée à la valeur **false**.

3.1.2 - OPÉRATEURS

M0005523.1

Par ordre de priorité croissant :

bool <bool& bVariable> = <bool bCondition>	Affecte la valeur de **bCondition** à la variable **bVariable** et renvoie la valeur de **bCondition**
bool <bool bCondition1> or <bool bCondition2>	Renvoie la valeur du OU logique entre **bCondition1** et **bCondition2**. **bCondition2** n'est évaluée que si **bCondition1** est **false**.
bool <bool bCondition1> and <bool bCondition2>	Renvoie la valeur du ET logique entre **bCondition1** et **bCondition2**. **bCondition2** n'est évaluée que si **bCondition1** est **true**.
bool <bool bCondition1> xor bool <bCondition2>	Renvoie la valeur du OU exclusif entre **bCondition1** et **bCondition2**
bool <bool bCondition1> != <bool bCondition2>	Teste l'égalité des valeurs de **bCondition1** et **bCondition2**. Renvoie **true** si les valeurs sont différentes, **false** sinon.
bool <bool bCondition1> == <bool bCondition2>	Teste l'égalité des valeurs de **bCondition1** et **bCondition2**. Renvoie **true** si les valeurs sont identiques, **false** sinon.
bool ! <bool bCondition>	Renvoie la négation de la valeur de **bCondition**

Afin d'éviter les confusions entre les opérateurs = et ==, l'opérateur = n'est pas autorisé dans les expressions de VAL 3 servant de paramètre d'instructions. **if(bCondition1=bCondition2)** serait interprété comme **bCondition1=bCondition2** ; **if(bCondition1==true)**. Cependant, l'intention était souvent d'écrire : **if(bCondition1==bCondition2)**, ce qui est très différent !

3.2 - TYPE NUM

M0005524.1

3.2.1 - DÉFINITION

M0005525.1

Le type num représente une valeur numérique comprenant environ **14** chiffres significatifs.

Chaque calcul numérique se fait donc avec une précision limitée par ces **14** chiffres significatifs.

Il faut en tenir compte lorsque l'on veut tester l'égalité de deux numériques : le plus souvent, il est nécessaire de tester dans un intervalle.

```
[-] <digits>[.<digits>][e[-]<digits>]
```

Les constantes de type numérique ont le format suivant :

Le ’e’ est un marqueur de notation scientfique numérique, qui remplace ’10^’ : 1e3 est égal à 1 x 10^3 (ou 1000), 1e-2 est égal à 1 x 10^(-2) (ou 0.01).

Les variables de type num sont initialisées par défaut à la valeur 0.

Exemples

Le test du résultat du calcul numérique doit tenir compte de l'inexactitude numérique des calculs.

Il vaut mieux remplacer if cos(**nAngle**)==0 par if abs(cos(**nAngle**))<1e-10.

Voici quelques nombres constants :

```
1
0.2
-3.141592653
6.02214179e23
1.054571628e-34
```

3.2.2 - OPÉRATEURS

M0005526.1

Par ordre de priorité croissant :

Outil **num <num& nVariable> = <num nValue>**	Affecte **nValue** à la variable **nVariable** et renvoie **nValue**.
bool <num nValue1> != <num nValue2>	Renvoie **true** si **nValue1** n'est pas égal à **nValue2**, **false** sinon.
bool <num nValue1> == <num nValue2>	Renvoie **true** si **nValue1** est égal à **nValue2**, **false** sinon.
bool <num nValue1> >= <num nValue2>	Renvoie **true** si **nValue1** est supérieur ou égal à **nValue2**, **false** sinon.
bool <num nValue1> > <num nValue2>	Renvoie **true** si **nValue1** est strictement supérieur à **nValue2**, **false** sinon.
bool <num nValue1> <= <num nValue2>	Renvoie **true** si **nValue1** est inférieur ou égal à **nValue2**, **false** sinon.
bool <num nValue1> < <num nValue2>	Renvoie **true** si **nValue1** est strictement inférieur à **nValue2**, **false** sinon.
num <num nValue1> - <num nValue2>	Renvoie la différence entre **nValue1** et **nValue2**.
num <num nValue1> + <num nValue2>	Renvoie la somme de **nValue1** et **nValue2**.
num <num nValue1> % <num nValue2>	Opération modulo : Renvoie le reste de la division entière de **nValue1** par **nValue2**. Une erreur d'exécution est générée si **nValue2** est 0. Le signe du reste est le signe de **nValue1**.
num <num nValue1> / <num nValue2>	Renvoie le quotient de **nValue1** par **nValue2**. Une erreur d'exécution est générée si **nValue2** est 0.
num <num nValue1> * <num nValue2>	Renvoie le produit de **nValue1** et **nValue2**.
num - <num nValue>	Renvoie l'opposé de **nValue**.

Afin d'éviter les confusions entre les opérateurs = et ==, l'opérateur = n'est pas autorisé dans les expressions de VAL 3 servant de paramètre d'instructions. **nCos**=cos(**nAngle**=30) doit être remplacé par **nAngle**=30 ; **nCos**=cos(**nAngle**).

3.2.3 - INSTRUCTIONS

M0005527.1

num **sin**(num nAngle)

M0005528.1

Fonction

Retourne le sinus de **nAngle**.

Paramètres

num nAngle	angle en degrés

Exemples

```
sin(30) returns 0.5
```

num **asin**(num nValue) M0005529.1

Fonction

Retourne le sinus inverse de **nValue**, en degrés. Le résultat est compris entre **-90** et **+90** degrés.

Une erreur d'exécution est générée si **nValue** est supérieur à **1** ou inférieur à **-1**.

Exemples

```
asin(0.5) returns 30
```

num **cos**(num nAngle) M0005530.1

Fonction

Retourne le cosinus de **Angle**.

Paramètres

num nAngle	angle en degrés

Exemples

```
cos(60) returns 0.5
```

num **acos**(num nValue) M0005531.1

Fonction

Retourne le cosinus inverse de **nValue**, en degrés. Le résultat est compris entre **0** et **180** degrés.

Une erreur d'exécution est générée si **nValue** est supérieur à **1** ou inférieur à **-1**.

Exemples

```
acos(0.5) returns 60
```

num **tan**(num nAngle) M0005532.1

Fonction

Retourne la tangente de **Angle**.

Paramètres

num nAngle	angle en degrés

Exemples

```
tan(45) returns 1.0
```

num **atan**(num nValue) M0005533.1

Fonction

Retourne la tangente inverse de **nValue**, en degrés. Le résultat est compris entre **-90** et **+90** degrés.

Exemples

```
atan(1) returns 45
```

num **atan2**(num nX, num nY) M0005888.1

Fonction

Retourne la tangente d'arc en degrés de **nX/nY** à partir des signes des deux valeurs pour déterminer le quadrant correct.

atan2(0,0) renvoie une erreur d'exécution 70 (valeur du paramètre incorrecte).

Exemples

```
atan2(1,0) = 90
atan2(0,-1) = 180
atan2(-1,0) = -90
atan2(0,1) = 0
```

num **abs**(num nValue) M0005534.1

Fonction

Retourne la valeur absolue de **nValue**.

Exemples

Le test du résultat du calcul numérique doit tenir compte de l'inexactitude numérique des calculs :

Il vaut mieux remplacer if cos(**nAngle**)==0 par if abs(cos(**nAngle**))<1e-10.

```
abs(3.1415) returns 3.1415
abs(-3.1415) returns 3.1415
```

num **sqrt**(num nValue) M0005535.1

Fonction

Retourne la racine carré de **nValue**.

Une erreur d'exécution est générée si **nValue** est négatif.

Exemples

```
sqrt(9) returns 3
```

num **exp**(num nValue) M0005536.1

Fonction

Retourne l'exponentielle de **nValue**.

Une erreur d'exécution est générée si **nValue** est trop grand.

Exemples

```
exp(1) returns 2.71828
```

num **power**(num nX, num nY) M0005537.1

Fonction

Renvoie **nX** à la puissance **nY** : nX^{nY}

Une erreur d'exécution est générée dans le cas suivant :

- si **nX** est nul et **nY** négatif.
- si **nX** est négatif et **nY** n'est pas entier.
- si le résultat est trop grand.

Exemples

Ce programme calcule de 2 manières différentes 5 à la puissance 7.

```
// First way:  power instruction
nResult = power(5,7)
// Second way: power(x,y)=exp(y*ln(x)) (with numerical inaccuracy)
nResult = exp(7*ln(5))
```

num **ln**(num nValue) M0005538.1

Fonction

Retourne le logarithme népérien de **nValue**.

Une erreur d'exécution est générée si **nValue** est négatif ou nul.

Exemples

```
ln(2.718281828) returns 0.99999999983113
```

num **log**(num nValue) M0005539.1

Fonction

Retourne le logarithme décimal de **nValue**.

Une erreur d'exécution est générée si **nValue** est négatif ou nul.

Exemples

```
log(1000) returns 3
```

num **roundUp**(num nValue) M0005540.1

Fonction

Retourne **nValue** arrondi à l'entier immédiatement supérieur.

Exemples

```
roundUp(7.8) returns 8
roundUp(-7.8) returns -7
```

num **roundDown**(num nValue) M0005541.1

Fonction

Retourne **nValue** arrondi à l'entier immédiatement inférieur.

Exemples

```
roundDown(7.8) returns 7
roundDown(-7.8) returns -8
```

num **round**(num nValue) M0005542.1

Fonction

Retourne **nValue** arrondi à l'entier le plus proche.

```
Example
round(7.8) returns 8
round(-7.8)  returns -8
round(0.5)  returns 1
round(-0.5) returns 0
```

num **min**(num nX, num nY) M0005543.1

Fonction

Retourne le minimum de **nX** et **nY**.

Exemples

```
min(-1,10) returns -1
```

num **max**(num nX, num nY) M0005544.1

Fonction

Retourne le maximum de **nX** et **nY**.

Exemples

```
max(-1,10) returns 10
```

num **limit**(num nValue, num nMin, num nMax) M0005545.1

Fonction

Renvoie **nValue** borné par **nMin** et **nMax**.

Exemples

```
limit(30,-90,90) returns 30
limit(100,-90,90) returns 90
limit(-100,-90,90) returns -90
```

num **sel**(bool bCondition, num nValue1, num nValue2) M0005546.1

Fonction

Renvoie **nValue1** si **bCondition** est **true**, sinon **nValue2**.

Exemples

```
sel(true,-90, 90) returns -90
sel(false,-90, 90) returns 90
```

3.3 - TYPE DE CHAMP DE BITS M0005547.1

3.3.1 - DÉFINITION M0005548.1

Un champ de bits est un moyen de stocker et d'échanger sous forme compacte une série de bits (valeurs booléennes ou entrées/sorties numériques). VAL 3 ne donne pas un type de données spécifique pour gérer les champs de bits mais réutilise le type num pour stocker un champ de bits de 32 bits sous la forme d'un nombre entier positif dans la plage [0, 2^32-1].

Toute valeur numérique de VAL 3 peut être considérée comme un champ de bits de 32 bits ; les instructions de traitement du champ de bits arrondissent automatiquement une valeur numérique à un nombre entier positif de 32 bits, qui est alors traité comme un champ de bits de 32 bits.

3.3.2 - OPÉRATEURS M0005549.1

Les opérateurs standard du type num s'appliquent dans un champ de bits : '=', '==', '!='.

3.3.3 - INSTRUCTIONS M0005550.1

num **bNot**(num nBitField) M0005551.1

Fonction

Cette instruction renvoie l'opération logique binaire "NOT" (négation) dans un champ de bits de 32 bits. (Le i ème bit du résultat est initialisé à 1 si le i ème bit de l'entrée est à 0). On obtient ainsi un nombre entier positif dans la plage [0, 2^32-1].

L'entrée numérique est d'abord arrondie à un nombre entier positif dans la plage [0, 2^32-1] avant que l'opération binaire soit appliquée.

Exemples

Ce programme réinitialise les bits i à j d'un champ de bits **nBitField** à l'aide d'un masque **nMask**.

```
// Compute a bit mask with bits i to j set to 1 (see bOr for explanations)
nMask=(power(2,j-i+1)-1)*power(2,i)
// Invert the mask to have all bits to 1 except bit i to j
nMask=bNot(nMask)
// Reset bits i to j using the bitwise 'and'
nBitField=bAnd(nBitField, nMask)
```

num **bAnd**(num nBitField1, num nBitField2) M0005552.1

Fonction

Cette instruction renvoie l'opération logique binaire "ET" sur deux champs de bits de 32 bits. (Le i ème bit du résultat est initialisé à 1 si les i èmes bits des deux entrées sont initialisés à 1). On obtient ainsi un nombre entier positif dans la plage [0, 2^32-1].

Les entrées numériques sont d'abord arrondies à un nombre entier positif dans la plage [0, 2^32-1] avant que l'opération binaire soit appliquée.

Exemples

Ce programme affiche un champ de bits **nBitField** de 32 bits à l'écran en testant successivement chaque bit :

```
for i=31 to 0 step -1
  // Compute the mask for the i th bit
  nMask=power(2,i)
  if bAnd(nBitField, nMask)==nMask
   sOutput="1"
  else
   sOutput="0"
  endIf
endFor
```

num **bOr**(num nBitField1, num nBitField2) M0005553.1

Fonction

Cette instruction renvoie l'opération logique binaire "OU" sur deux champs de bits de 32 bits. (Le i ème bit du résultat est initialisé à 1 si le i ème bit d'au moins une entrée est initialisé à 1). On obtient ainsi un nombre entier positif dans la plage [0, 2^32-1].

Les entrées numériques sont d'abord arrondies à un nombre entier positif dans la plage [0, 2^32-1] avant que l'opération binaire soit appliquée.

Exemples

Ce programme calcule de deux manières différentes un masque de champ de bits dans lequel les i ème à j ème bits sont activés.

```
// First way: logical 'or' on bits i to j
nBitField=0
for k=i to j
 nBitField=bOr(nBitField, power(2,k))
endFor
// Second way: compute a bit mask of (j-i) bits
nBitField=(power(2,j-i+1)-1)
// Then shift the bit mask by i bits
nBitField=nBitField*power(2,i)
```

num **bXor**(num nBitField1, num nBitField2) M0005554.1

Fonction

Cette instruction renvoie l'opération logique binaire "XOR" (ou exclusif) sur deux champs de bits de 32 bits. (Le i ème bit du résultat est initialisé à 1 si les i èmes bits des deux entrées sont différents). On obtient ainsi un nombre entier positif dans la plage [0, 2^32-1].

Les entrées numériques sont d'abord arrondies à un nombre entier positif dans la plage [0, 2^32-1] avant que l'opération binaire soit appliquée.

Exemples

Ce programme inverse les bits i à j du champ de bits **nBitField** :

```
// Compute mask for bits i to j (see bOr example)
nMask=(power(2,j-i+1)-1)*power(2,i)
// Invert bits i to j using the mask
nBitField=bXor(nBitField,nMask)
```

num **toBinary**(num **nValue**[], num **nValueSize**, string **sDataFormat**, num& **nDataByte**[])

num **fromBinary**(num **nDataByte**[], num **nDataSize**, string **sDataFormat**, num& **nValue**[])

Fonction

L'instruction toBinary/fromBinary a pour rôle de permettre l'échange de valeurs numériques entre deux appareils à l'aide d'une connexion série ou réseau. Les valeurs numériques sont tout d'abord codées sous la forme d'un flux d'octets. Les octets sont ensuite envoyés à l'appareil correspondant. Enfin, l'appareil correspondant décode les octets pour récupérer les valeurs numériques initiales. Différents codages binaires de valeurs numériques sont possibles.

L'instruction toBinary code les valeurs numériques sous forme de tableau d'octets (champ de bits de 8 bits, nombre entier positif dans la plage [0, 255]) selon les spécifications du format de données **sDataFormat**. Le nombre de valeurs numériques **nValue** à coder est donné par le paramètre **nValueSize**. Le résultat est enregistré dans le tableau **nDataByte** et le tableau renvoie le nombre d'octets codés dans ce tableau.

Une erreur d'exécution est générée si le nombre de valeurs à coder **nValueSize** dépasse la taille de **nValue**, si le format spécifié n'est pas pris en charge ou si le tableau de résultats **nDataByte** n'est pas assez grand pour coder toutes les données d'entrée.

L'instruction fromBinary décode un tableau d'octets en valeurs numériques **nValue**, selon les spécifications du format de données **sDataFormat**. Le nombre d'octets à décoder est donné par le paramètre **nDataSize**. Le résultat est enregistré dans le tableau **nValue** et l'instruction renvoie le nombre de valeurs dans ce tableau. Si certaines données binaires sont altérées (octets en dehors de la plage [0, 255] ou codage de la virgule flottante invalide), l'instruction renvoie l'opposé du nombre de valeurs correctement décodées (valeur négative).

Une erreur d'exécution se produit si le nombre d'octets à décoder **nDataSize** dépasse la taille de **nDataByte**, si le format spécifié n'est pas pris en charge ou si le tableau de résultat **nValue** n'est pas assez grand pour décoder toutes les données d'entrée.

Les codages binaires pris en charge sont indiqués dans le tableau ci-dessous :

- Le signe "-" indique le codage d'un nombre entier signé (le dernier bit du champ de bits code le signe de la valeur).
- Le chiffre indique le nombre d'octets pour le codage de chaque valeur numérique.
- L'extension ".0" marque le codage des valeurs de virgule flottante (les codages à simple et double précision selon IEEE 754 sont pris en charge).
- La dernière lettre indique l'ordre des octets : "l" pour "little endian" (le codage commence par le bit de plus faible poids), "b" pour "big endian" (le codage commence par le bit de plus fort poids). Le codage "big endian" est standard pour les applications en réseau (TCP/IP).

"-1"	Octet signé
"1"	Octet non signé
"-2l"	Mot signé, little endian
"-2b"	Mot signé, big endian
"2l"	Mot non signé, little endian
"2b"	Mot non signé, big endian
"-4l"	Double mot signé, little endian
"-4b"	Double mot signé, big endian
"4l"	Double mot non signé, little endian
"4b"	Double mot non signé, big endian
"4.0l"	Valeur à virgule flottante simple précision, little endian

"4.0b"	Valeur à virgule flottante simple précision, big endian
"8.0l"	Valeur à virgule flottante double précision, little endian
"8.0b"	Valeur à virgule flottante double précision, big endian

Le format natif de VAL 3 pour les données numériques est le codage à double précision. Ce format doit être utilisé pour échanger des valeurs numériques sans perte de précision.

Exemples

Le premier programme code une donnée trsf **trShiftOut** sous la forme d'un tableau d'octets **nByteOut** et l'envoie par la liaison **siTcpClient**. Le deuxième programme lit les octets sur la connexion **siTcpServer** et les reconvertit en trsf **trShiftIn**.

```
// ---- Program to send a trsf ----
// Copy the trsf coordinates into a numerical buffer
nTrsfOut[0]=trShiftOut.x
nTrsfOut[1]=trShiftOut.y
nTrsfOut[2]=trShiftOut.z
nTrsfOut[3]=trShiftOut.rx
nTrsfOut[4]=trShiftOut.ry
nTrsfOut[5]=trShiftOut.rz
// Encode 6 numerical values (double precision floating point,
// therefore 8 bytes) into 6*8=48 bytes in nByteOut[48] array
toBinary(nTrsfOut, 6, "8.0b", nByteOut)
// Send nByte array (48 bytes) through tcpClient
sioSet(siTcpClient, nByteOut)
// ---- Program to read a trsf ----
nb=0
i=0
while (nb<48)
 nb=sioGet(siTcpServer, nByteIn[i])
 if(nb>0)
   i=i+nb
 else
  // Communication error
   return
 endIf
endWhile
if (fromBinary(nByteIn, 48, "8.0b", nTrsfIn) != 6)
  // Corrupted data
 return
else
   trShiftIn.x=nTrsfIn[0]
   trShiftIn.y=nTrsfIn[1]
   trShiftIn.z=nTrsfIn[2]
   trShiftIn.rx=nTrsfIn[3]
   trShiftIn.ry=nTrsfIn[4]
   trShiftIn.rz=nTrsfIn[5]
endIf
```

3.4 - TYPE STRING

M0005556.1

3.4.1 - DÉFINITION

M0005557.1

Les variables de type chaîne de caractères permettent de stocker des textes. Le type chaîne permet d'utiliser l'ensemble de caractères standard Unicode. On notera que l'affichage correct d'un caractère Unicode dépend des polices de caractères installées sur l'appareil d'affichage.

Une chaîne est mémorisée dans 128 octets ; le nombre maximum de caractères dans une chaîne dépend des caractères utilisés, le codage interne (Unicode UTF8) pouvant utiliser entre 1 octet (pour les caractères ASCII) et 4 octets pour les caractères (3 pour les caractères chinois).

La longueur maximale d'une chaîne ASCII est donc de 128 caractères, celle d'une chaîne en chinois de 42 caractères.

La valeur d'initialisation par défaut des variables de type chaîne est "" (chaîne vide).

3.4.2 - OPÉRATEURS

M0005558.1

Par ordre de priorité croissant :

string <string& sVariable> = <string sString>	Affecte **sString** à la variable **sVariable** et renvoie **sString**.
bool <string sString1> != <string sString2>	Renvoie **true** si **sString1** et **sString2** ne sont pas identiques, sinon **false**.
bool <string sString1> == <string sString2>	Renvoie **true** si **sString1** et **sString2** sont identiques, sinon **false**.
string <string sString1> + <string <sString2>	Renvoie les premiers caractères (limité à **128** octets) de **sString1** concaténés avec **sString2**.

Afin d'éviter les confusions entre les opérateurs = et ==, l'opérateur = n'est pas autorisé dans les expressions de VAL 3 servant de paramètre d'instructions. **nLen**=len(**sString**="hello wild world") doit être remplacé par **sString**="hello wild world" ; **nLen**=len(**sString**).

3.4.3 - INSTRUCTIONS

M0005559.1

string **toString**(string sFormat, num nValue)

M0005826.1

Fonction

Cette instruction renvoie une chaîne de caractères représentant **nValue** selon le format d'affichage **sFormat**.

Le format est **"size.precision"**, où **size** est la taille minimale du résultat (des espaces sont ajoutés en début de chaîne au besoin), et **precision** est le nombre de chiffres significatifs après la virgule (les **0** en fin de chaîne sont remplacés par des espaces). Par défaut, **size** et **precision** valent **0**. La partie entière de la valeur n'est jamais tronquée, même si sa longueur d'affichage est plus grande que **size**. **size** ne peut pas dépasser la longueur de chaîne VAL 3 définie dans le chapitre 3.4.1.

Exemples

renvoie

```
nPi = 3.141592654
toString(".4", nPi) returns "3.1416"
toString("8", nPi) returns "       3" (7 spaces before the '3'
toString("8.4", nPi) returns "  3.1416" (2 spaces before the '3')
toString("8.4", 2.70001) returns "  2.7   " (2 spaces before the '2', 3 spaces after the '7')
toString("", nPi) returns "3"
toString("1.2", 1234.1234) returns "1234.12"
```

Voir aussi

chr

toNum

string **toNum**(string sString, num& nValue, bool& bReport) M0005767.1

Fonction

Cette instruction trouve le **nValue** numérique représenté au début de **sString** spécifié et renvoie **sString** dans lequel tous les caractères ont été supprimés jusqu'à la représentation suivante d'une **value** numérique.

Si le début de **sString** ne représente pas une valeur numérique, **bReport** vaut **false** et **nValue** n'est pas modifié ; sinon, **bReport** vaut **true**.

Exemples

```
toNum("10 20 30", nVal, bOk) returns "20 30", nVal equals 10, bOk equals true
toNum("a10 20 30", nVal, bOk) returns "a10 20 30", nVal is unchanged, bOk equals false
toNum("10 end", nVal, bOk) returns "", nVal equals 10, bOk equals true

This program displays successively 90, 0, -7.6, 17.3
sBuffer = "+90 0.0 -7.6 17.3"
do
 sBuffer = toNum(sBuffer, nVal, bOk)
 sOutput = toString("", nVal)
until (sBuffer=="") or (bOk != true)
```

Voir aussi

toString

string **chr**(num nCodePoint) M0005768.1

Fonction

Cette instruction renvoie la chaîne faite du caractère de code de point Unicode, s'il s'agit d'un code de point Unicode valide. Dans le cas contraire, il renvoie une chaîne vide.

Le tableau ci-dessous indique les codes Unicode inférieurs à **128** (il correspond au tableau de caractères **ASCII**). Les caractères dans les cases grises sont des codes de contrôle qui peuvent être remplacés par un point d'interrogation quand la chaîne est affichée.

Tous les points de code Unicode valides sont pris en charge par le type de chaîne VAL 3. L'affichage du caractère dépend toutefois des polices de caractères installées sur l'appareil d'affichage. La liste complète des caractères Unicode est accessible à l'adresse http://www.unicode.org (chercher dans les "Code Charts").

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
NUL	SOH	STX	ETX	EOT	ENQ	ACQ	BEL	BS	HT	LF	VT	FF	CR	SO	SI
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
" "	!	"	#"	$	%	&	'	(	)	*	+	,	-	.	/
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
p	q	r	s	t	u	v	w	x	y	z	(	|	)	~	DEL

Exemples

```
chr(65) returns "A"
```

Voir aussi

asc

num **asc**(string sText, num nPosition) M0005789.1

Fonction

Cette instruction renvoie le code de point Unicode du caractère d'index **nPosition**.

Elle renvoie -1 si **nPosition** est négatif ou supérieur à la longueur de texte spécifiée.

Exemples

```
asc("A",0) returns 65
```

Voir aussi

chr

string **left**(string sText, num nSize) M0005560.1

Fonction

Cette instruction renvoie les **nSize** premiers caractères de **sText**. Si **nSize** est plus grand que la longueur de **sText**, l'instruction renvoie **sText**.

Une erreur d'exécution est générée si **nSize** est négatif.

Exemples

```
left("hello world",5) returns "hello"
left("hello world",20) returns "hello world"
```

string **right**(string sText, num nSize) M0005561.1

Fonction

Cette instruction renvoie les **nSize** derniers caractères de **sText**. Si le nombre spécifié est plus grand que la longueur de **sText**, l'instruction renvoie **sText**.

Une erreur d'exécution est générée si **nSize** est négatif.

Exemples

```
right("hello world",5) returns "world"
right("hello world",20) returns "hello world"
```

string **mid**(string sText, num nSize, num nPosition) M0005562.1

Fonction

Renvoie **nSize** caractères de **sText** à partir du caractère d'index **nPosition**, en s'arrêtant à la fin de **sText**.

Une erreur d'exécution est générée si **nPosition** ou **sText** est négatif.

Exemples

```
mid("hello wild world",4,6) returns "wild"
mid("hello wild world",20,6) returns "wild world"
```

string **insert**(string sText, string sInsertion, num nPosition) M0005563.1

Fonction

Cette instruciton renvoie **sText** où **sInsertion** est inséré après le caractère d'index **nPosition**. Si **nPosition** est plus grand que la taille de **sText**, **sInsertion** est ajoutée en fin de **sText**. Le résultat est tronqué s'il dépasse 128 octets.

Une erreur d'exécution est générée si nPosition est négatif.

Exemples

```
insert("hello world","wild",6))returns "hello wild world"
```

string **delete**(string sText, num nSize, num nPosition) M0005564.1

Fonction

Cette instruction renvoie **sText** où **nSize** a été supprimé du caractère d'index **nPosition**. Si **nPosition** est supérieur à la longueur de **sText**, l'instruction renvoie **sText**.

Une erreur d'exécution est générée si **nSize** ou **nPosition** est négatif.

Exemples

```
delete("hello wild world",5,6) returns "hello world"
```

string **replace**(string sText, string sReplacement, num nSize, num nPosition) M0005565.1

Fonction

Cette instruction renvoie **sText** où **nSize** caractères ont été remplacés dans le caractère d'index **nPosition** par **sReplacement**. Si **nPosition** est supérieur à la longueur de **sText**, l'instruction renvoie **sText**.

Une erreur d'exécution est générée si **nSize** ou **nPosition** est négatif.

Exemples

```
replace("hello ? world","wild",1,6) returns "hello wild world"
```

num **find**(string sText1, string sText2) M0005566.1

Fonction

Cette instruction renvoie l'index (entre **0** et **127**) du premier caractère dans la première occurrence de **sText2** dans **sText1**. Si **sText2** n'apparaît pas dans **sText1**, l'instruction renvoie **-1**.

Exemples

```
find("hello wild world", "wild") returns 6
```

Fonction

Cette instruction renvoie le nombre de caractères dans **sText**.

Exemples

```
len("hello wild world") returns 16
```

3.5 - TYPE DIO M0005567.1

3.5.1 - DÉFINITION M0005568.1

Les variables de type dio servent à interfacer une application VAL 3 avec les entrées et sorties digitales du système. Une variable dio enregistre un lien vers une entrée ou sortie digitale du système, ou "adresse physique".

Toutes les instructions utilisant une variable de type dio non liée à une entrée/sortie déclarée dans le système génèrent une erreur d'exécution. La valeur d'initialisation par défaut des variables de type dio est un lien indéfini. Le lien d'une variable dio peut être initialisé à partir d'une autre variable dio, du MCP du robot, ou à l'aide de VAL 3 Studio dans la Stäubli Robotics Suite.

3.5.2 - OPÉRATEURS M0005569.1

Par ordre de priorité croissant :

bool <dio diOutput> = <bool bCondition>	Affecte **bCondition** à l'état de **diOutput**, et renvoie **bCondition**. Une erreur d'exécution est générée si **diOutput** n'est pas liée à une sortie système.
bool <dio diInput1> != <bool bInput2>	Renvoie **true** si **diInput1** et **bInput2** ne sont pas dans le même état, sinon renvoie **false**.
bool <dio diInput> != <bool bCondition>	Renvoie **true** si l'état de **diInput** n'est pas égal à **bCondition**, sinon renvoie **false**.
bool <dio diInput> == <bool bCondition>	Renvoie **true** si l'état de **diInput** est égal à **bCondition**, sinon renvoie **false**.
bool <dio diInput1> == <dio diInput2>	Renvoie **true** si **diInput1** et **diInput2** sont dans le même état, sinon renvoie **false**.

Afin d'éviter les confusions entre les opérateurs = et ==, l'opérateur = n'est pas autorisé dans les expressions de VAL 3 servant de paramètre d'instructions. **if(diOutput=diInput)** serait interprété comme : **diOutput=diInput** ; **if(diOutput==true)**. Cependant, l'intention était souvent d'écrire : **if(diOutput==diInput)**, ce qui est très différent !

DANGER
L'opérateur '=' entre les deux variables dio n'existe plus avec VAL 3 s7 pour des raisons de cohérence avec les autres opérateurs '=' (voir la définition de l'opérateur '=' pour les types d'utilisateur). Il peut facilement être remplacé par l'opérateur '=' entre un dio et un bool : le **diOut = diIn** des versions antérieures de VAL 3 peut être remplacé par **diOut = (diIn==true)**.

void **dioLink**(dio& diVariable, dio diSource) M0005819.1

Fonction

Cette instruction lie **diVariable** à l'entrée/sortie à laquelle **diSource** est lié.

Exemples

Cette application utilise un signal qui peut être configuré à l'aide de différents matériels. Le programme ci-dessous détermine quel matériel est installé afin d'initialiser la variable **diSignal** qui est utilisée ensuite dans le reste de l'application.

```
if(ioStatus(diDevice1Signal)>=0)
 // device 1 is installed: use it
 dioLink(diSignal, diDevice1Signal)
elseIf (ioStatus(diDevice2Signal)>=0)
 // device 2 is installed: use it
 dioLink(diSignal, diDevice2Signal)
else
 sOutput="Error: no io device installed"
endIf
```

num **dioGet**(dio diArray[]) M0005809.1

Fonction

Cette instruction renvoie la valeur numérique de **diArray** lue sous la forme d'un entier écrit en code binaire, à savoir :

diArray[0] + 2 *diArray[1] + 4 *diArray[2] + ... + 2 ^k *diArray[k],
où **diArray[i] = 1** si **diArray[i]** est **true**, 0 sinon.

Une erreur d'exécution est générée si un élément de **diArray** n'est pas lié à une entrée/sortie système.

DANGER
diArray est obligatoirement un tableau à une dimension.

Exemples

```
diArray[0] = false
diArray[1] = true
diArray[2] = false
diArray[3] = true
```

Voir aussi

dioSet

Fonction

Cette instruction convertit la partie entière de **nValue** en code binaire, l'affecte aux sorties de **diArray** et renvoie la valeur correspondante, à savoir :

diArray[0] + 2 *diArray[1] + 4 *diArray[2] + ... + 2 ^k *diArray[k],
où **diArray[i] = 1** si **diArray[i]** est **true**, 0 sinon.

Une erreur d'exécution est générée si un élément de **diArray** n'est pas lié à une sortie système.

DANGER
diArray est obligatoirement un tableau à une dimension.

Exemples

En utilisant **di4bitsArray**, tableau de taille 4 :

```
dioSet(di4bitsArray, 10) returns 10
dioSet(di4bitsArray, 26) returns 10, because 26 requires 5 bits in a binary encoding: 10 = 26 - 2^4
```

Voir aussi

dioGet

num **ioStatus**(dio diInputOutput) M0005867.1

Fonction

Cette instruction renvoie un nombre positif si la variable d'entrée-sortie spécifiée fonctionne et un nombre négatif si elle est en erreur. La valeur renvoyée détaille le statut de l'entrée-sortie :

0	L'entrée-sortie fonctionne.
1	L'entrée-sortie fonctionne mais elle est verrouillée par l'opérateur. Les entrées ont alors une valeur fixe (contrôlée par l'opérateur) qui peut être différente de la valeur matérielle. Les sorties ont alors une valeur fixe contrôlée par l'opérateur : l'écriture sur la sortie n'a aucun effet. Le mode de verrouillage est un moyen de débogage.
2	L'entrée-sortie est simulée (entrée-sortie logicielle, sans effet sur le matériel).
-1	L'entrée-sortie ne fonctionne pas parce que le lien (adresse physique) n'est pas défini.
-2	L'entrée-sortie ne fonctionne pas parce que le lien (adresse physique) ne correspond à aucune entrée-sortie du système. Le matériel correspondant à l'adresse physique n'est pas installé ou n'a pas pu être initialisé.
-3	L'entrée-sortie ne fonctionne pas parce que le dispositif d'entrée-sortie est en erreur.

Exemples

Cette application utilise un signal qui peut être configuré à l'aide de différents matériels. Le programme ci-dessous détermine quel matériel est installé afin d'initialiser la variable diSignal qui est utilisée ensuite dans le reste de l'application.

```
if(ioStatus(diDevice1Signal)>=0)
 // device 1 is installed: use it
 dioLink(diSignal, diDevice1Signal)
elseIf (ioStatus(diDevice2Signal)>=0)
 // device 2 is installed: use it
 dioLink(diSignal, diDevice2Signal)
else
 sOutput="Error: no io device installed"
endIf
```

Voir aussi

ioStatus(dio...

ioStatus(aio...

num **ioStatus**(dio diInputOutput, string& sDescription, string& sPhysicalPath) M0005833.1

Fonction

Cette instruction se comporte exactement comme l'instruction ioStatus décrite plus haut mais renvoie en outre le texte descriptif et le lien (adresse physique) de l'entrée-sortie spécifiée.

La description est un texte libre, défini à l'aide des outils de configuration des entrées-sorties. Le format du lien physique dépend du dispositif d'entrée-sortie. Il prend habituellement la forme : 'boardName\moduleName\ioAdress' ou une chaine d'identification unique telle que 'C792E6DA-FEA0-44D7-BECF-FFB6CB1B6577'.

Exemples

Ce programme teste un signal et affiche des informations d'erreur si celui-ci ne fonctionne pas.

```
if ioStatus(diSignal, sDecription, sPath)<0
 sOutput="Signal"+sPath+"in error"+"Description:"+sDecription
endIf
```

Voir aussi

ioStatus(aio...

ioStatus(dio...

3.6 - TYPE AIO M0005571.1

3.6.1 - DÉFINITION M0005572.1

Les variables de type aio servent à interfacer une application VAL 3 avec les entrées et sorties analogiques du système. Une variable aio enregistre un lien vers une entrée ou sortie analogique du système, ou "adresse physique".

Toutes les instructions utilisant une variable de type aio non liée à une entrée/sortie déclarée dans le système génèrent une erreur d'exécution. La valeur d'initialisation par défaut des variables de type aio est un lien indéfini. La liaison d'une variable aio peut être initialisée depuis une autre variable aio, depuis le robot MCP (version série SRC uniquement) ou en utilisant VAL 3 Studio de la suite Stäubli Robotics Suite.

3.6.2 - INSTRUCTIONS M0005573.1

void **aioLink**(aio& aiVariable, aio aiSource) M0005820.1

Fonction

Cette instruction lie **aiVariable** à l'entrée/sortie à laquelle **aiSource** est lié.

Exemples

Cette application utilise un signal qui peut être configuré à l'aide de différents matériels. Le programme ci-dessous détermine quel matériel est installé afin d'initialiser la variable aiSignal qui est utilisée ensuite dans le reste de l'application.

```
if(ioStatus(aiDevice1Signal)>=0)
 // device 1 is installed: use it
 aioLink(aiSignal, aiDevice1Signal)
elseIf (ioStatus(aiDevice2Signal)>=0)
 // device 2 is installed: use it
 aioLink(aiSignal, aiDevice2Signal)
else
 sOutput="Error: no io device installed"
endIf
```

num **aioGet**(aio aiInput) M0005769.1

Fonction

Cette instruction renvoie la valeur numérique d'**aiInput**.

Une erreur d'exécution est générée si **aiInput** n'est pas liée à une entrée/sortie système.

Exemples

```
aioGet(aiSensor) returns the current sensor value
```

Voir aussi

aioSet

num **aioSet**(aio aiOutput, num nValue) M0005770.1

Fonction

Cette instruction affecte **nValue** à **aiOutput** et renvoie **nValue**. Si la valeur affectée ne correspond pas à la plage de valeurs de l'aio, le nombre renvoyé est la valeur réelle de la sortie aio.

Une erreur d'exécution est générée si **aiOutput** n'est pas liée à une sortie système.

Exemples

```
aioSet(aiCommand, -12.3) writes -12.3 to the output command and returns -12.3 if aiCommand is a floating point output.
aioSet(aiCommand, 12.3) writes 12 to the output command and returns 12 if aiCommand is an integer output.
```

Voir aussi

aioGet

num **ioStatus**(aio aiInputOutput) M0005818.1

Fonction

Cette instruction renvoie un nombre positif si la variable d'entrée-sortie spécifiée fonctionne et un nombre négatif si elle est en erreur. La valeur renvoyée détaille le statut de l'entrée-sortie :

0	L'entrée-sortie fonctionne.
1	L'entrée-sortie fonctionne mais elle est verrouillée par l'opérateur. Les entrées ont alors une valeur fixe (contrôlée par l'opérateur) qui peut être différente de la valeur matérielle. Les sorties ont alors une valeur fixe contrôlée par l'opérateur : l'écriture sur la sortie n'a aucun effet. Le mode de verrouillage est un moyen de débogage.
2	L'entrée-sortie est simulée (entrée-sortie logicielle, sans effet sur le matériel).
-1	L'entrée-sortie ne fonctionne pas parce que le lien (adresse physique) n'est pas défini.
-2	L'entrée-sortie ne fonctionne pas parce que le lien (adresse physique) ne correspond à aucune entrée-sortie du système. Le matériel correspondant à l'adresse physique n'est pas installé ou n'a pas pu être initialisé.
-3	L'entrée-sortie ne fonctionne pas parce que le dispositif d'entrée-sortie est en erreur.

Exemples

Cette application utilise un signal qui peut être configuré à l'aide de différents matériels. Le programme ci-dessous détermine quel matériel est installé afin d'initialiser la variable **aiSignal** qui est utilisée ensuite dans le reste de l'application.

```
if(ioStatus(aiDevice1Signal)>=0)
 // device 1 is installed: use it
 aioLink(aiSignal, aiDevice1Signal)
elseIf (ioStatus(aiDevice2Signal)>=0)
 // device 2 is installed use it
 aioLink(aiSignal, aiDevice2Signal)
else
 sOutput="Error: no io device installed"
endIf
```

Voir aussi

ioStatus

num **ioStatus**(aio dIInputOutput, string& sDescription, string& sPhysicalPath) M0005821.1

Fonction

Cette instruction se comporte exactement comme l'instruction ioStatus décrite plus haut mais renvoie en outre le texte descriptif et le lien (adresse physique) de l'entrée-sortie spécifiée.

La description est un texte libre, défini à l'aide des outils de configuration des entrées-sorties. Le format du lien physique dépend du dispositif d'entrée-sortie. Il prend habituellement la forme : 'deviceName\moduleName\ioAddress'.

Exemples

Ce programme teste un signal et affiche des informations d'erreur si celui-ci ne fonctionne pas.

```
if ioStatus(aiSignal, sDecription, sPath)<0
 sOutput="Signal"+sPath+"in error"+"Description:"+sDecription
endIf
```

Voir aussi

ioStatus(aio...

ioStatus(dio...

3.7 - TYPE SIO M0005574.1

3.7.1 - DÉFINITION M0005575.1

Le type sio est utilisé pour lier une variable VAL 3 à un port série ou à une connexion Ethernet Socket. Une entrée-sortie sio est caractérisée par :

- Des paramètres propres au type de communication, définis dans le système
- Un caractère de fin de chaîne, pour permettre l'utilisation du type string
- Un délai d'attente de communication

Les entrées-sorties série du système sont toujours actives. Les connexions Ethernet Socket sont établies au moment de l'acccès initial en lecture ou en écriture par un programme VAL 3. Les connexions Ethernet Socket sont interrompues automatiquement lorsque l'application VAL 3 est fermée.

Toutes les instructions utilisant une variable de type sio non liée à une entrée/sortie déclarée dans le système génèrent une erreur d'exécution. La valeur d'initialisation par défaut des variables de type sio est un lien indéfini. Le lien d'une variable sio peut être initialisé à partir d'une autre variable sio, du MCP du robot, ou à l'aide de VAL 3 Studio dans la Stäubli Robotics Suite.

3.7.1.1 - Sockets M0005576.1

La communication via Ethernet est prise en charge par le système avec les protocoles TCP/IP ou UDP. TCP/IP assure une livraison fiable du flux, tandis qu'UDP assure une liaison non fiable mais plus rapide du flux.

La communication Ethernet utilise les **Sockets** déclarées dans le panneau de configuration du contrôleur du robot. Les Sockets sont des ressources IO du contrôleur où les variables sioVAL 3 peuvent être liées (sioLink) pour permettre aux applications VAL 3 de communiquer via Ethernet IP.

3.7.1.2 - Paramètres de Socket

M0005577.1

Adresse IP

L'adresse IP définie a une signification qui varie en fonction du type de Socket.

Client TCP/ : Définit l'adresse IP du serveur à connecter à cette Socket client.

TCP/IP serveur : Définit l'adresse IP des clients à connecter à cette Socket client.

UDP

- Une adresse valable : Cette socket UDP ne peut communiquer (lecture et écriture) qu'avec cette adresse IP.
- 0.0.0.0 : La Socket reçoit tous les messages transmis par les clients UDP connectés à ce port. Noter qu'une tentative d'écriture dans cette Socket UDP génère une erreur d'exécution 122.
- *.*.255.255 : La Socket transmet et reçoit les référentiels uniquement comme une diffusion définie par le masque IP. Le paramètre de diffusion doit être activé.

Délai écoulé

Valeur en secondes du délai utilisé pour gérer les accès au réseau (lecture, écriture).

- 0 : attendre indéfiniment la transaction demandée.
- >0 : la transaction (lecture ou écriture) peut attendre la durée du délai en secondes.
- -1 : la transaction (lecture ou écriture) n'attend pas (pas de délai) pour continuer.

Port

Valeur numérique sur 32 bits définissant le numéro de port IP.

Délimiteur de chaîne

Code ASCII pour le caractère de fin de chaîne à utiliser avec les opérateurs '=' (dans la plage [0, 255]).

Nagle

Ce paramètre booléen optimise le temps de réponse lors des communications avec la Socket.

Si l'optimisation de nagle est désactivée, le temps de réponse est amélioré mais la charge sur le système est accrue.

3.7.1.3 - TCP (Protocole de contrôle de transmission)

M0005578.1

TCP est un protocole complémentaire du protocole Internet (IP), la suite complète est donc TCP/IP. TCP assure une livraison fiable et ordonnée d'un flux d'octets depuis un programme installé sur un ordinateur vers un autre programme installé sur un autre ordinateur.

D'autres applications, qui n'exigent aucun service de correction de flux de données, peuvent utiliser le protocole datagramme utilisateur (UDP), qui réduit la latence au détriment de la fiabilité.

Pour procéder à des échanges TCP/IP, une connexion entre un client TCP/IP et un serveur TCP/IP doit être établie. Les connexions TCP/IP sont gérées automatiquement par le système en fonction des paramètres qui ont été définis lors de la déclaration de la socket.

Les connexions de la Socket sont créées dès qu'un programme VAL 3 doit l'utiliser (sioGet, sioSet, clearBuffer) :

- Les Sockets client tentent de se connecter aux serveurs déclarés. Noter qu'une seule instance de connexion client existe. Elle est partagée par toutes les variables sioVAL 3 qui y sont liées.
- Les Sockets du serveurs attendent une connexion de client. Noter que chaque variable sio liée à un serveur de socket utilise sa propre instance de connexion.

Les connexions de Socket sont automatiquement interrompues lorsqu'elles ne sont plus utilisées par un programme VAL 3 :

- Lorsque l'application est fermée.
- Lorsqu'une erreur est détectée en lecture ou en écriture (sauf dépassement de délai).
- Par la fonction **clearBuffer**.
- Une connexion de serveur est interrompue lorsque sa variable VAL 3 propriétaire est détruite (les variables locales sont détruites à la fin de la routine).
- Une connexion client est interrompue lorsque plus aucune variable VAL 3 n'y est référencée (déchargement d'une librairie).

3.7.1.4 - UDP (Protocole datagramme utilisateur) M0005579.1

UDP utilise un modèle de transmission simple sans échanges d'établissement de liaison implicite pour garantir la fiabilité, le classement et l'intégrité des données. UDP fournit par conséquent un service non fiable et les datagrammes peuvent arriver dans le désordre, être dupliqués ou être manquants sans avis.

UDP suppose que le contrôle et la correction des erreurs sont soit non nécessaires soit effectués par l'application, évitant ainsi de surcharger l'interface de réseau avec ces traitements. Les applications ayant des contraintes de temps utilisent souvent UDP car l'abandon de paquets est préférable à l'attente de paquets en retard, ce qui ne peut pas être une option dans un système en temps réel.

A la différence de TCP, les Sockets UDP ne sont pas qualifiées client ou serveur mais compatibles avec les deux types : la même Socket peut être utilisée en lecture et en écriture.

Si des fonctions de correction d'erreur sont nécessaires au niveau de l'interface réseau, le protocole de contrôle de transmission (TCP), conçu à cette fin, est préférable.

3.7.2 - OPÉRATEURS

M0005580.1

Une erreur d'exécution est générée quand le délai de communication est dépassé lors de la lecture ou de l'écriture de l'entrée/sortie.

string <sio siOutput> = <string sText>	Ecrit successivement dans **siOutput** les caractères **sText** des codes Unicode UTF8, suivis du caractère de fin de chaîne, et renvoie **sText**.
num <sio siOutput> = <num nData item>	Ecrit sur **siOutput** l'entier le plus proche de **nData item**, modulo **256**, et renvoie la valeur effectivement envoyée.
num <num nData> = <sio siInput>	Lit un octet sur **siInput** et affecte la valeur de l'octet à **nData**.
string <string sText> = <sio siInput>	Lit dans **siInput** une chaîne de caractères Unicode UFT8 et affecte cette chaîne à **sText**. Les caractères non supportés par le type string sont ignorés. La chaîne est terminée lorsque le caractère de fin de chaîne est lu ou lorsque **sText** atteint la taille maximale d'un string (**128** octets). Le caractère de fin de chaîne n'est pas recopié dans **sText**.

Afin d'éviter les confusions entre les opérateurs = et ==, l'opérateur = n'est pas autorisé dans les expressions de VAL 3 servant de paramètre d'instructions.

nLen=len(**sString**=**siInput**) doit être remplacé par

sString=**siInput** ; **nLen**=len(**sString**).

3.7.3 - INSTRUCTIONS

M0005581.1

void **sioLink**(sio& siVariable, sio siSource) M0005582.1

Fonction

Cette instruction lie **siVariable** à l'entrée/sortie série du système auquel **siSource** est lié.

Voir aussi

dioLink

aioLink

num **clearBuffer**(sio siVariable) M0005583.1

Fonction

Cette instruction vide la mémoire de lecture **siVariable** et renvoie le nombre de caractères effacés

Pour une connexion à une Socket Ethernet, clearBuffer désactive (ferme) la Socket. clearBuffer renvoie **-1** si la Socket a déjà été désactivée.

Une erreur d'exécution est générée si **siVariable** n'est pas connectée à une liaison série du système ou à une Socket Ethernet.

num **ioStatus**(sio siInputOutput) M0005584.1

Fonction

Cette instruction renvoie un nombre positif si la variable d'entrée-sortie spécifiée fonctionne et un nombre négatif si elle est en erreur. La valeur renvoyée détaille le statut de l'entrée-sortie :

0	L'entrée-sortie fonctionne.
-1	L'entrée-sortie ne fonctionne pas parce que le lien (adresse physique) n'est pas défini.
-2	L'entrée-sortie ne fonctionne pas parce que le lien (adresse physique) ne correspond à aucune entrée-sortie du système.

num **ioStatus**(sio siInputOutput, string& sDescription, string& sPhysicalPath) M0005585.1

Fonction

Cette instruction se comporte exactement comme l'instruction ioStatus décrite plus haut mais renvoie en outre le texte descriptif et le lien (adresse physique) de l'entrée-sortie spécifiée.

La description est un texte libre, défini à l'aide des outils de configuration des entrées-sorties. Le format du lien physique dépend du dispositif d'entrée-sortie. Il prend habituellement la forme :

'deviceName\moduleName\ioAddress'

Exemples

Ce programme teste un signal et affiche des informations d'erreur si celui-ci ne fonctionne pas.

```
if ioStatus(siSignal, sDecription, sPath)<0
 sOutput="Signal "+sPath+ "in error"+"Description:"+sDecription
endIf
```

num **sioGet**(sio siInput, num& nData[]) M0005586.1

Fonction

Cette instruction lit un caractère unique ou un tableau de caractères sur **siInput** et renvoie le nombre de caractères lus.

La séquence de lecture s'arrête lorsque le tableau **nData** est plein ou lorsque le tampon de lecture des entrées est vide.

- Pour une connexion à une Socket Ethernet, sioGet tente d'abord d'établir une connexion si aucune connexion n'est établie. Lorsque le délai d'attente de communication d'entrée est atteint, sioGet retourne **-1**.
- Lorsque la connexion est établie mais le tampon de lecture d'entrée ne contient aucune donnée, sioGet attend que des données soient reçues et renvoie la taille des données lues, ou attend la fin du délai programmé et renvoie 0. Il attend indéfiniment si le délai de la socket a été programmé à 0. En cas de détection d'erreur ou d'interruption de la connexion, la fonction s'arrête immédiatement et renvoie -1.

Si le délai de la Socket est -1, la fonction lit le tampon de lecture et renvoie le nombre de caractères lus (0 si le tampon de lecture est vide) ou -1 en cas d'erreur de lecture.

- L'erreur d'exécution 123 est générée si **siInput** n'est lié à aucun port série ou une Socket Ethernet du système.
- Les erreurs d'exécution 20 ou 21 sont générées si **nData** n'est pas utilisé avec le bon index.

num **sioSet**(sio siOutput, num& nData[]{,nNbElements}) M0005587.1

Fonction

Cette instruction écrit un ou plusieurs caractères dans **siOutput** et renvoie le nombre de caractères écrits.

Le paramètre facultatif **nNbElements** spécifie le nombre de caractères à écrire.

Les valeurs numériques sont converties avant transmission en entier entre **0** et **255**, en prenant l'entier le plus proche modulo **256**.

Pour une connexion à une Socket Ethernet, sioSet tente d'abord d'établir une connexion si aucune connexion n'est établie. Lorsque le délai d'attente de communication de sortie est atteint, sioSet retourne **-1**. Le nombre de caractères écrits peut être inférieur à la taille de **nData** si un erreur de communication est détectée.

L'erreur d'exécution 123 est générée si **siOutput** n'est lié à aucun port série ou une Socket Ethernet du système.

- L'erreur d'exécution 20 est générée si **nNbElements** est supérieur à la dimension du tableau **nData**.
- Les erreurs d'exécution 20 ou 21 sont générées si **nData** n'est pas utilisé avec le bon index.
- Avec une Socket UDP, l'erreur d'exécution 122 est générée en cas de tentative d'écriture sur IP 0.0.0.0.

num **sioCtrl**(sio siChannel, string nParameter, *value) M0005588.1

Fonction

Cette instruction modifie un paramètre de communication de l'entrée/sortie série/Ethernet **siChannel** spécifiée.

Pour les lignes série, le matériel peut ne pas accepter certains paramètres ou certaines valeurs de paramètres : se reporter au manuel du contrôleur.

L'instruction renvoie :

0	Le paramètre a été modifié.
-1	Le paramètre n'est pas défini.
-2	La valeur du paramètre n'a pas le type attendu.
-3	La valeur du paramètre n'est pas prise en charge.
-4	Le canal n'est pas prêt à appliquer le changement de paramètre (il doit d'abord être arrêté).
-5	Le paramètre n'est pas défini pour ce type de canal.

Les paramètres pris en charge sont indiqués dans le tableau ci-dessous :

Nom du paramètre	Type de paramètre	Description
"port"	num	(Pour client ou serveur TCP) Port TCP
"target"	string	(Pour client TCP) Adresse IP du serveur TCP à atteindre, par exemple "192.168.0.254"
"endOfString"	num	(Pour ligne série, client et serveur TCP) Code ASCII pour le caractère de fin de chaîne à utiliser avec les opérateurs '=' (dans la plage [0, 255])
"timeout"	num	(Pour ligne série, client et serveur TCP) Temps de réponse maximal pour le canal de communication. 0 signifie : pas de limite de délai.
"baudRate"	num	(Pour la ligne série) Vitesse de communication
"parity"	string	(Pour la ligne série) Contrôle de parité : "none", "even" ou "odd"
"bits"	num	(Pour la ligne série) Nombre de bits par octet (5, 6, 7 ou 8)
"stopBits"	num	(Pour la ligne série) Nombre de bits d'arrêt par octet (1 ou 2)
"mode"	string	(Pour la ligne série) Mode de communication : "rs232"
"flowControl"	string	(Pour la ligne série) Contrôle du débit : "none" ou "hardware"
"nagle"	bool	(Pour le client ou serveur TCP) Activation (par défaut) ou désactivation de l'optimisation de nagle. Si l'optimisation de nagle est désactivée, le temps de réponse est amélioré mais la charge sur le système est accrue.

Exemples

Ce programme configure les paramètres principaux d'une ligne série.

```
sioCtrl(siPortSerial1, "baudRate", 115200)
sioCtrl(siPortSerial1, "bits", 8)
sioCtrl(siPortSerial1, "parity", "none")
sioCtrl(siPortSerial1, "stopBits", 1)
sioCtrl(siPortSerial1, "timeout", 0)
sioCtrl(siPortSerial1, "endOfString", 13)
```

4 - INTERFACE UTILISATEUR

M0005589.1

num **userPage**(string sPageName)

M0005590.1

Fonction

Cette instruction affiche la page utilisateur spécifiée à l'écran du MCP. Le paramètre **sPageName** est le nom de la page. En fait, ce nom correspond à un nom de fichier sans extension html décrivant la page. Par défaut, la fonction userPage recherche le fichier dans le dossier nommé "interface" du dossier de travail de l'application. Le chemin du fichier peut également être spécifié en utilisant les conventions de chemin de fichier VAL 3.

La fonction userPage renvoie 0 lorsque le fichier correspondant à la page est retrouvé et ouvert correctement, en cas contraire, elle renvoie -1.

Exemples

Ce programme affiche la page utilisateur nommée **myUserPage** correspondant au fichier **myUserPage.html** depuis l'interface du dossier de l'application.

```
nRet=userPage("myUserPage")
```

void **userPageBind**(string sPage, string sGraphicalObjectId, ...)

M0005591.1

void **userPageBind**(string **sPage**, string **sGraphicalObjectId**, string **sGraphicalObjectAttrib**, num& **nVar**, num **nSize**, string **sDirection**, num **nRefresh**)

void **userPageBind**(string **sPage**, string **sGraphicalObjectId**, string **sGraphicalObjectAttrib**, string& **sVar**, num **nSize**, string **sDirection**, num **nRefresh**)

void **userPageBind**(string **sPage**, string **sGraphicalObjectId**, string **sGraphicalObjectAttrib**, bool& **bVar**, num **nSize**, string **sDirection**, num **nRefresh**)

void **userPageBind**(string **sPage**, string **sGraphicalObjectId**, string **sGraphicalObjectAttrib**, dio& **dDio**, num **nSize**, string **sDirection**, num **nRefresh**)

void **userPageBind**(string **sPage**, string **sGraphicalObjectId**, string **sGraphicalObjectAttrib**, aio& **aAio**, num **nSize**, string **sDirection**, num **nRefresh**)

Fonction

Cette fonction lie l'attribut **sGraphicalObjectAttrib** de l'objet graphique **sGraphicalObjectId** de la page **sPage** à une variable VAL 3.

sPage	La page contenant l'élément graphique à lier (Même chaîne que celle utilisée par la fonction userPage()).	
sGraphicalObjectId	ID unique de l'élément graphique à lier.	
sGraphicalObjectAttrib	Attribut de l'élément graphique à lier.	
nVar	Un élément d'une variable.	
nSize	Nombre d'éléments à lier (max. : 49 éléments).	
sDirection	Définit le sens de la liaison du point de vue de la page graphique.	
	Valeurs	**Description**
	r	Lecture. L'élément graphique est mis à jour avec la valeur de la variable VAL 3.
	w	Écriture. L'élément graphique met à jour la valeur de la variable VAL 3.
	rw	Lecture/Écriture. L'élément graphique est mis à jour avec la valeur de la variable VAL 3 et met à jour la valeur de la variable VAL 3.
	r1	Lecture une seule fois. L'élément graphique est mis à jour avec la valeur de la variable VAL 3 une seule fois lorsque la page est ouverte.
	r1w	Lecture une seule fois/Écriture.
nRefresh	Paramètre facultatif définissant le temps de cycle d'actualisation. 1 pour tous les cycles ; 2 pour un cycle sur 2, ... Ce paramètre n'est utile que si la direction est "r" ou "rw".	

Notes concernant la plage de variables liées

Les variables liées font l'objet d'un certain nombre de restrictions. Une variable peut être :

- Une variable globale,
- Une partie de variable structurée telle qu'un joint ou un point,
- Une partie d'une structure définie par l'utilisateur,
- Une variable issue d'une autre librairie,
- Un paramètre de fonction si ce paramètre a été passé par référence,
- La taille de la liaison peut être > 1 uniquement si le type de conteneur est un tableau.

Exemples

```
// Fill select options with firsts elements of val3_array
userPageBind("page1","select_id1","options",val3_array[0],5,"r1")

// get the slected value in the variable val_string
userPageBind("page1","select_id1","value",val_string,1,"w")

// Force the selection to the option "hello"
val_string = "hello"
userPageBind("page1","select_id1","value",val_string,1,"r")
```

bool **userPageUnbind**(string sPage, string sGraphicalObjectId, string sGraphicalObjectAttrib) M0005592.1

Fonction

Cette fonction élimine la liaison de l'attribut **sGraphicalObjectAttrib** de l'objet graphique **sGraphicalObjectId** de la page **sPage**.

La fonction renvoie **false** si la liaison n'existe pas. Ce n'est pas obligatoire pour contrôler la valeur renvoyée.

void **userPageRefresh**(string sPageName) M0005593.1

Fonction

Cette instruction oblige à actualiser les liaisons de la page définies comme paramètres. Dans le cas du prototype sans arguments, toutes les pages d'utilisateur sont actualisées.

Il est intéressant d'utiliser cette instruction si vous voulez actualiser certaines valeurs liées en mode "r1" sans devoir redessiner toute la page.

Exemples

Dans cet exemple, la variable **sString** change toutes les 30 secondes. Il serait peu économique de lier cette variable en mode "r1" puisqu'elle serait interrogée toutes les 200 ms et cela utiliserait beaucoup de puissance de calcul de la CPU. En conséquence, nous la lions comme une chaîne de caractères lue une seule fois et utilisons **userPageRefresh**() pour l'actualiser lorsque cela est nécessaire.

```
userPageBind("page1", "id1", "innerHTML", sString, "r1")
userPage("page1")
sString = "a"
while true
  sString = sString + "b"
  userPageRefresh ("page1")
  delay(30)
endWhile
```

void **userPageBindClick**(string sPage, string sGraphicalObjectId, ...) M0005594.1

void **userPageBindClick**(string **sPage**, string **sGraphicalObjectId**, bool& **bVar**, bool **bValue**)

void **userPageBindClick**(string **sPage**, string **sGraphicalObjectId**, dio& **dVar**, bool **bValue**)

void **userPageBindClick**(string **sPage**, string **sGraphicalObjectId**, num& **nVar**, num **nValue**)

void **userPageBindClick**(string **sPage**, string **sGraphicalObjectId**, aio& **aVar**, num **nValue**)

void **userPageBindClick**(string **sPage**, string **sGraphicalObjectId**, string& **sVar**, string **sValue**)

void **userPageBindClick**(string **sPage**, string **sGraphicalObjectId**, bool& **bVar**)

void **userPageBindClick**(string **sPage**, string **sGraphicalObjectId**, dio& **dVar**)

Fonction

Cette fonction lie l'événement de clic sur un objet graphique **sGraphicalObjectId** de la page **sPage** à une variable de VAL 3.

Cet événement est transmis quand on clique sur un élément graphique.

La variable **xVar** de VAL 3 est réglée à **xValue** chaque fois que l'événement de clic est déclenché.

Pour bool et dio, un autre prototype existe sans le paramètre valeur. Dans ce cas, la variable bascule chaque fois que l'événement se produit.

void **userPageBindMouseup**(string sPage, ...) M0005595.1

void **userPageBindMouseup**(string **sPage**, string **sGraphicalObjectId**, bool& **bVar**, bool **bValue**)

void **userPageBindMouseup**(string **sPage**, string **sGraphicalObjectId**, dio& **dVar**, bool **bValue**)

void **userPageBindMouseup**(string **sPage**, string **sGraphicalObjectId**, num& **nVar**, num **nValue**)

void **userPageBindMouseup**(string **sPage**, string **sGraphicalObjectId**, aio& **aVar**, num **nValue**)

void **userPageBindMouseup**(string **sPage**, string **sGraphicalObjectId**, string& **sVar**, string **sValue**)

void **userPageBindMouseup**(string **sPage**, string **sGraphicalObjectId**, bool& **bVar**)

void **userPageBindMouseup**(string **sPage**, string **sGraphicalObjectId**, dio& **dVar**)

Fonction

Cette fonction lie l'événement de levée de la souris de l'objet graphique **sGraphicalObjectId** de la page **sPage** à une variable de VAL 3.

Cet événement est transmis quand l'utilisateur relâche un élément graphique sur lequel il avait précédemment cliqué.

La variable **xVar** de VAL 3 est réglée à **xValue** chaque fois que l'événement de levée de la souris est déclenché.

Pour bool et dio, un autre prototype existe sans le paramètre valeur. Dans ce cas, la variable bascule chaque fois que l'événement se produit.

void **userPageBindMousedown**(string sPage, ...) M0005596.1

void **userPageBindMousedown**(string **sPage**, string **sGraphicalObjectId**, bool& **bVar**, bool **bValue**)

void **userPageBindMousedown**(string **sPage**, string **sGraphicalObjectId**, dio& **dVar**, bool **bValue**)

void **userPageBindMousedown**(string **sPage**, string **sGraphicalObjectId**, num& **nVar**, num **nValue**)

void **userPageBindMousedown**(string **sPage**, string **sGraphicalObjectId**, aio& **aVar**, num **nValue**)

void **userPageBindMousedown**(string **sPage**, string **sGraphicalObjectId**, string& **sVar**, string **sValue**)

void **userPageBindMousedown**(string **sPage**, string **sGraphicalObjectId**, bool& **bVar**)

void **userPageBindMousedown**(string **sPage**, string **sGraphicalObjectId**, dio& **dVar**)

Fonction

Cette fonction lie l'événement de descente de la souris de l'objet graphique **sGraphicalObjectId** de la page **sPage** à une variable de VAL 3.

Cet événement est transmis quand l'utilisateur clique sur un élément graphique.

La variable **xVar** de VAL 3 est réglée à **xValue** chaque fois que l'événement de descente de la souris est déclenché.

Pour bool et dio, un autre prototype existe sans le paramètre valeur. Dans ce cas, la variable bascule chaque fois que l'événement se produit.

bool **userPageUnbindClick**(string **sPage**, string **sGraphicalObjectId**, bool& **bVar**)

bool **userPageUnbindClick**(string **sPage**, string **sGraphicalObjectId**, dio& **bVar**)

bool **userPageUnbindClick**(string **sPage**, string **sGraphicalObjectId**, num& **nVar**)

bool **userPageUnbindClick**(string **sPage**, string **sGraphicalObjectId**, aio& **aVar**)

bool **userPageUnbindClick**(string **sPage**, string **sGraphicalObjectId**, string& **sVar**)

Fonction

Cette fonction supprime la liaison de l'événement de clic sur l'objet graphique **sGraphicalObjectId** de la page **sPage** à la variable de VAL 3.

Un événement peut être mappé à plusieurs liaisons.

Cette fonction renvoie vrai si une liaison correspondante a été identifiée et supprimée.

Exemples

Utilisation d'un booléen pour créer une fonction de rappel.

clickCallback.pgx

```
begin
 x_nCounter = 0
 while true
  // setMutex() wait for x_bClicked to be false, then set it to true and exit
  setMutex(x_bClicked)
  x_nCounter = x_nCounter + 1
 endWhile
end
```

start.pgx

...

```
// Init
bButtonClicked = true
nButtonClickCounter = 0

// Create callback function
taskCreate "tClickCb", 10, clickCallback(bButtonClicked, nButtonClickCounter)

// Force bButtonClicked to false each time some_button is clicked
userPageBindClick("index", "some_button", bButtonClicked, false)
```

...

...

Changer le texte d'un bouton en fonction de son état

...

```
userPageBindMousedown("some_page","some_button",sButtonState,"PUSHED")
userPageBindMouseup("some_page","some_button",sButtonState,"RELEASED")
userPageBind("some_page","some_button","innerHTML",sButtonState,1,"r")
```

...

bool **userPageUnbindMouseup**(string sPage, ...) M0005597.1

bool **userPageUnbindMouseup**(string **sPage**, string **sGraphicalObjectId**, bool& **bVar**)

bool **userPageUnbindMouseup**(string **sPage**, string **sGraphicalObjectId**, dio& **bVar**)

bool **userPageUnbindMouseup**(string **sPage**, string **sGraphicalObjectId**, num& **nVar**)

bool **userPageUnbindMouseup**(string **sPage**, string **sGraphicalObjectId**, aio& **aVar**)

bool **userPageUnbindMouseup**(string **sPage**, string **sGraphicalObjectId**, string& **sVar**)

Fonction

Cette fonction supprime la liaison de l'événement de levée de la souris de l'objet graphique **sGraphicalObjectId** de la page **sPage** à la variable de VAL 3.

Un événement peut être mappé à plusieurs liaisons.

Cette fonction renvoie vrai si une liaison correspondante a été identifiée et supprimée.

Voir aussi

userPageUnbindClick

bool **userPageUnbindMousedown**(string sPage, ...) M0005598.1

bool **userPageUnbindMousedown**(string **sPage**, string **sGraphicalObjectId**, bool& **bVar**)

bool **userPageUnbindMousedown**(string **sPage**, string **sGraphicalObjectId**, dio& **bVar**)

bool **userPageUnbindMousedown**(string **sPage**, string **sGraphicalObjectId**, num& **nVar**)

bool **userPageUnbindMousedown**(string **sPage**, string **sGraphicalObjectId**, aio& **aVar**)

bool **userPageUnbindMousedown**(string **sPage**, string **sGraphicalObjectId**, string& **sVar**)

Fonction

Cette fonction supprime la liaison de l'événement de descente de la souris de l'objet graphique **sGraphicalObjectId** de la page **sPage** à la variable de VAL 3.

Un événement peut être mappé à plusieurs liaisons.

Cette fonction renvoie vrai si une liaison correspondante a été identifiée et supprimée.

Voir aussi

userPageUnbindClick

void **userPageSet**(string sPage, string sGraphicalObjectId, ...) M0005599.1

void **userPageSet**(string **sPage**, string **sGraphicalObjectId**, string **sGraphicalObjectAttrib**, bool **bVar**)

void **userPageSet**(string **sPage**, string **sGraphicalObjectId**, string **sGraphicalObjectAttrib**, num **nVar**)

void **userPageSet**(string **sPage**, string **sGraphicalObjectId**, string **sGraphicalObjectAttrib**, string **sVar**)

Fonction

Cette fonction lie statiquement l'attribut **sGraphicalObjectAttrib** de l'objet graphique **sGraphicalObjectId** de la page **sPage** à une valeur variable VAL 3 ou une valeur constante.

sPage	La page contenant l'élément graphique à lier (Même chaîne que celle utilisée par la fonction userPage()).
sGraphicalObjectId	ID unique de l'élément graphique à lier.
sGraphicalObjectAttrib	Attribut de l'élément graphique à lier.
bVar/nVar/sVar	Une variable VAL 3 ou une valeur constante.

Notes concernant la plage de variables liées

Les variables liées font l'objet d'un certain nombre de restrictions. Une variable peut être :

- Une variable globale,
- Une partie de variable structurée telle qu'un joint ou un point,
- Une partie d'une structure définie par l'utilisateur,
- Une variable issue d'une autre librairie,
- Un paramètre de fonction si ce paramètre a été passé par référence.

Exemples

```
// Set label text directly
userPageSet("index", "myLabel", "innerHTML", "Hello, world!")

// Hide a button (you can use local variables here)
l_sVisibility = "hidden"
userPageSet("index", "myButton", "style.visibility", l_sVisibility)

// The following line won't make the button visible because l_sVisibility is not bound
l_sVisibility = "visible"

// Disable a button with a boolean value
userPageSet("index", "myButton", "disabled", true)
```

void **userPageAlert**(string sMessage) M0005600.1

Fonction

Affiche **sMessage** dans une case dans la page utilisateur et attend que l'utilisateur appuie sur le bouton OK.

bool **userPageConfirm**(string sMessage) M0005601.1

Fonction

Affiche **sMessage** dans une case dans la page utilisateur et attend que l'utilisateur confirme cette case. La fonction renvoie vrai si l'utilisateur appuie sur le bouton OK et false s'il appuie sur CANCEL.

bool **userPagePrompt**(string sMessage, string& sEntry) M0005602.1

Fonction

Elle demande à l'utilisateur une entrée avec **sMessage**. L'entrée est renvoyée dans sEntry.

Elle renvoie vrai si le bouton OK est appuyé et faux si le bouton CANCEL est appuyé.

bool **userPagePrompt**(string sMessage, num& nNum) M0005603.1

Fonction

Elle demande à l'utilisateur une entrée avec **sMessage**. L'entrée est renvoyée dans nNum.

Elle renvoie vrai si le bouton OK est appuyé et faux si le bouton CANCEL est appuyé.

void **popUpMsg**(string sText), void **popUpMsg**(string sText, num nSeverity) M0005863.1

Fonction

Cette instruction affiche **sText** dans une fenêtre **"popup"**. Cette fenêtre reste affichée quelques secondes. Le message est également enregistré avec la date et l'heure actuelles dans un fichier spécialement réservé aux journaux d'utilisateur (user.log).

Le paramètre optionnel **nSeverity** permet de définir un degré de gravité du message dans le contexte de l'application VAL 3. Les valeurs suivantes sont prises en charge :

sPage	Signification	Description
1	Information	Messages de fonctionnement ordinaires qui n'exigent aucune action.
2	Attention	Peut indiquer qu'une erreur va se produire si aucune mesure n'est prise.
3	Erreur	Conditions d'erreur.
4	Erreur grave	Conditions graves.

Voir aussi

userPage

bool **logMsg**(string sText), bool **logMsg**(string sText, num nSeverity) M0005775.1

Fonction

Cette instruction écrit **sText** dans l'historique. Le message est également enregistré avec la date et l'heure actuelles dans un fichier spécialement réservé aux journaux d'utilisateur (user.log). Certains messages du fichier d'historique peuvent se perdre si de nombreux messages sont enregistrés pendant la même seconde. L'instruction renvoie alors false de sorte que les messages sont enregistrés plus tard si cela est important.

Le paramètre optionnel **nSeverity** permet de définir un degré de gravité du message dans le contexte de l'application VAL 3. Les valeurs suivantes sont prises en charge :

Valeur nSeverity	Signification	Description
1	Information	Messages de fonctionnement ordinaires qui n'exigent aucune action.
2	Attention	Peut indiquer qu'une erreur va se produire si aucune mesure n'est prise.
3	Erreur	Conditions d'erreur.
4	Erreur grave	Conditions graves.

Exemples

Ce programme assure l'enregistrement du message :

```
while logMsg(sMessage)==false
 delay(0)
endWhile
```

Voir aussi

popUpMsg

string **getProfile**() M0005798.1

Fonction

Cette instruction renvoie le nom du profil utilisateur actuel.

Voir aussi

setProfile

num **setProfile**(string sUserLogin, string sUserPassword) M0005834.1

Fonction

Cette instruction change le profil utilisateur actuel (avec effet immédiat).

La fonction renvoie :

0	Le profil d'utilisateur spécifié prend effet
-1	Le profil d'utilisateur spécifié n'est pas défini.
-2	Le mot de passe d'utilisateur spécifié est incorrect.
-3	'**staubli**' n'est pas un profil d'utilisateur autorisé avec cette instruction.
-4	Le profil d'utilisateur en cours est '**staubli**' et ne peut pas être changé avec cette instruction.

Voir aussi

getProfile

string **getLanguage**() M0005803.1

Fonction

Cette instruction renvoie la langue actuelle du contrôleur du robot.

Exemples

```
switch(getLanguage())
 case "francais"
 sMessage="Attention!"
 break
 case "english"
   sMessage="Warning!"
 break
 case "deutsch"
   sMessage="Achtung!"
 break
 case "italiano"
 sMessage="Avviso!"
 break
 case "espanol"
 sMessage="¡Advertencia!"
 break
 default
 sMessage="Warning!"
 break
endSwitch
```

Voir aussi

setLanguage

bool **setLanguage**(string sLanguage) M0005802.1

Fonction

Cette instruction modifie la langue actuelle du contrôleur du robot : le nom de la langue spécifiée **sLanguage** doit correspondre au nom d'un fichier de traduction sur le contrôleur. Voir le manuel du contrôleur pour supprimer ou installer une langue dans le contrôleur du robot.

Exemples

Ce programme fait passer la langue du robot au chinois :

```
if(setLanguage("chinese")==false)
 sOutput="The Chinese language is not available on the robot controller"
endIf
```

Voir aussi

getLanguage

string **getDate**(string sFormat) M0005604.1

Fonction

Cette instruction renvoie la date et/ou l'heure actuelles du contrôleur du robot. Le paramètre **sFormat** spécifie le format de la date à renvoyer. Dans cette chaîne, chaque occurrence de certains mots-clés est remplacée par la valeur de date ou d'heure correspondante. Les mots-clés de format acceptés sont indiqués dans le tableau ci-dessous :

Mot-clé	Description
%y	Année en 2 chiffres (00-99), sans le siècle
%Y	Année en 4 chiffres, par exemple 2007
%m	Mois (00-12)
%d	Jour (00-31)
%H	Heure au format 24 (00-23)
%I	Heure au format 12 (01-12)
%p	Indicateur A.M./P.M. pour l'horloge sur 12 heures
%M	Minutes (00-59)
%S	Secondes (00-59)

Exemples

Ce programme affiche la date et l'heure au format "January 01, 2007 13:45:23"

```
switch (getDate("%m"))
   case "01"
     sMonth="January"
   break
   case "02"
     sMonth="February"
   break
   case "03"
     sMonth="March"
   break
   case "04"
     sMonth="April"
   break
   case "05"
     sMonth="May"
   break
   case "06"
     sMonth="June"
   break
   case "07"
     sMonth="July"
   break
   case "08"
     sMonth="August"
   break
   case "09"
     sMonth="September"
   break
   case "10"
     sMonth="October"
   break
   case "11"
     sMonth="November"
   break
   case "12"
     sMonth="December"
   break
   default
     sMonth="???"
   break
endSwitch
  // Display date and date in the form: "January 01, 2007  13:45:23"
 sOutput=getDate(sMonth+" %d, %Y %H:%M:%S")
```

5 - TÂCHES

M0005605.1

5.1 - DÉFINITION

M0005827.1

Une tâche est un programme en cours d'exécution. Plusieurs tâches peuvent s'exécuter dans une même application, et c'est habituellement le cas.

Dans une application, on trouvera typiquement une tâche pour les déplacements du bras, une tâche automate, une tâche pour l'interface utilisateur, une tâche pour le suivi des signaux de sécurité, des tâches de communication...

Une tâche est caractérisée par les éléments suivants :

- un nom : un identifiant de tâche unique à l'intérieur de la librairie ou de l'application
- une priorité ou une période : paramètre pour le séquencement des tâches
- un programme : le point d'entrée (et de sortie) de la tâche
- un état : actif ou arrêté
- la prochaine instruction à exécuter (et son contexte)

5.2 - REPRISE APRÈS UNE ERREUR D'EXÉCUTION

M0005606.1

Quand une instruction provoque une erreur d'exécution, la tâche est interrompue. L'instruction taskStatus() sert à diagnostiquer l'erreur d'exécution. Il est alors possible de redémarrer la tâche avec l'instruction taskResume(). Si l'erreur d'exécution peut être corrigée, la tâche peut reprendre à partir de la ligne d'instruction où elle s'est arrêtée. Dans le cas contraire, il est nécessaire de repartir soit avant, soit après cette ligne d'instruction.

Démarrage et arrêt d'application

Lorsqu'une application est démarrée, son programme **start**() est exécuté dans une tâche du nom de l'application suivi de **'~'**, et de priorité **10**.

Lorsqu'une application se termine, son programme **stop**() est exécuté dans une tâche du nom de l'application précédé de **'~'**, et de priorité **10**.

Si l'arrêt d'une application VAL 3 est provoqué depuis l'interface utilisateur du contrôleur MCP, la tâche de démarrage, si elle existe encore, est immédiatement détruite. Le programme **stop**() est exécuté ensuite, puis les tâches de l'application éventuellement restantes sont supprimées dans l'ordre inverse de celui de leur création, après quoi les librairies sont déchargées de la mémoire.

5.3 - VISIBILITÉ

M0005607.1

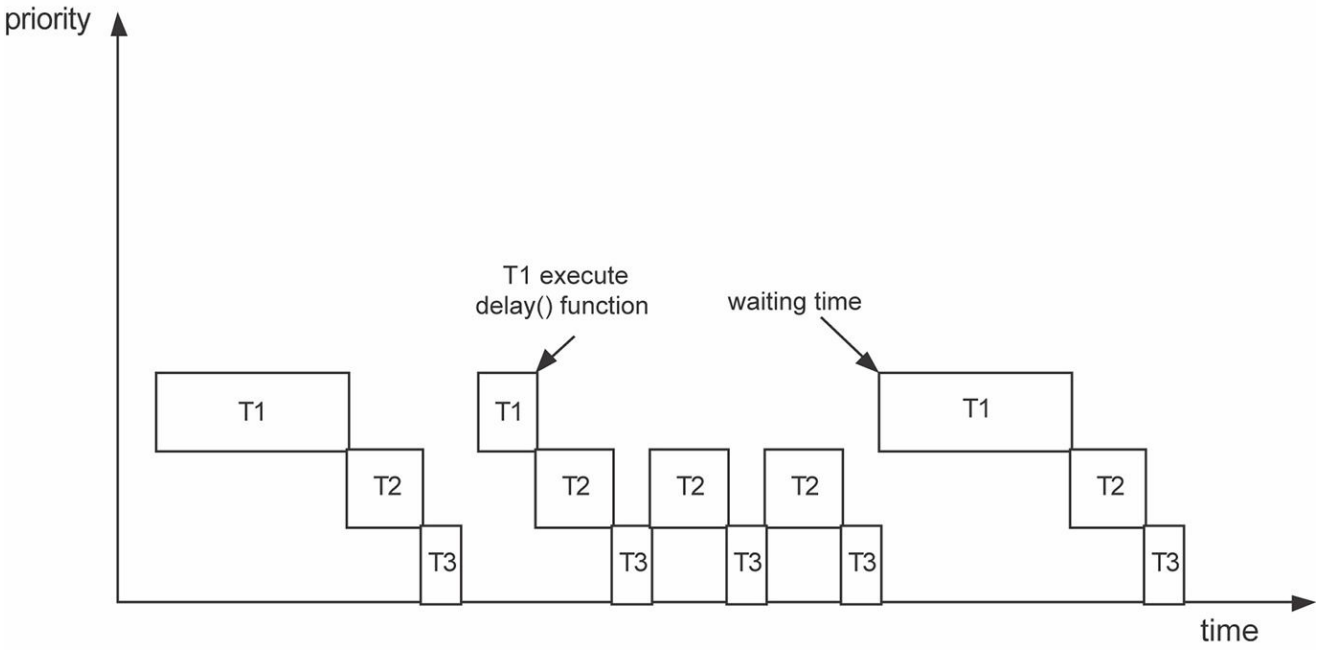
Une tâche n'est visible que de l'intérieur de l'application ou de la librairie qui l'a créée. Les instructions taskSuspend(), taskResume(), taskKill() et taskStatus() agissent sur une tâche créée par une autre librairie comme si la tâche n'existait pas. Deux librairies différentes peuvent donc créer des tâches ayant le même nom.

5.4 - SÉQUENCEMENT

M0005608.1

Lorsque qu'une application a plusieurs tâches en cours d'exécution, celles-ci semblent s'exécuter simultanément et indépendamment. Ceci est vrai si l'on observe l'application globalement sur des intervalles de temps suffisamment larges (de l'ordre de la seconde), mais n'est pas exact lorsque l'on s'intéresse au comportement précis sur un intervalle de temps réduit.

En effet, le système n'ayant qu'un seul processeur, il ne peut exécuter qu'une seule tâche à la fois. La simultanéité de l'exécution est simulée par un séquencement rapide des tâches, qui exécutent à tour de rôle quelques instructions avant que le système ne passe à la tâche suivante.



Anglais	Traduction
Priority	Priorité
Time	Temps
T1 execute delay() function	T1 exécute la fonction delay()
Waiting time	Délai écoulé

Figure 5.1 : Séquencement

Le séquencement des tâches VAL 3 est fait suivant les règles suivantes :

1) Les tâches sont séquencées suivant leur ordre de création

2) A chaque séquence, le système essaye d'exécuter un nombre de lignes d'instructions VAL 3 égal à la priorité de la tâche.

3) Quand une ligne d'instruction ne peut pas être terminée (erreur d'exécution, attente d'un signal, tâche arrêtée, etc.), le système passe à la tâche VAL 3 suivante.

4) Lorsque toutes les tâches VAL 3 sont terminées, le système conserve du temps disponible pour les tâches système de priorité inférieure (par ex. communication réseau, rafraîchissement de l'écran utilisateur, accès aux fichiers) avant de démarrer un nouveau cycle. L'attente maximum entre deux cycles consécutifs est égale à la durée du dernier cycle de séquencement ; mais le plus souvent, cette attente est nulle car le système n'en a pas besoin.

Les instructions VAL 3 pouvant provoquer le séquencement immédiat de la tâche suivante sont :

- watch() (attente temporisée d'une condition)
- delay() (temporisation)
- wait() (attente d'une condition)
- waitEndMove() (attente de l'arrêt du bras)
- open() et close() (attente de l'arrêt du bras, puis temporisation)
- taskResume() (attend que la tâche soit prête à être relancée)
- taskKill() (attend que la tâche soit effectivement tuée)
- disablePower() (attend que la puissance soit effectivement coupée)
- Les instructions accédant au contenu du disque (libLoad, libSave, libDelete, libList, setProfile)
- Les instructions de lecture/écriture de sio (opérateur =, sioGet(), sioSet())
- setMutex() (attend que le mutex booléen soit faux)

5.5 - TÂCHES SYNCHRONES

M0005837.1

La séquence décrite ci-dessus est la séquence des tâches normales, appelées tâches asynchrones, dont le système programme l'exécution la plus rapide possible. Il est parfois nécessaire de programmer les tâches à intervalles réguliers, que ce soit pour l'acquisition de données ou pour le contrôle de périphérique : on parle alors de tâches synchrones.

Celles-ci sont exécutées dans le cycle de séquence en interrompant la tâche asynchrone en cours entre deux lignes de VAL 3. Lorsque les tâches synchrones sont terminées, la tâche asynchrone reprend.

Le séquencement des tâches synchrones VAL 3 obéit aux règles suivantes :

1) Chaque tâche synchrone est séquencée exactement une fois pour chaque période définie lors de la création de la tâche (par exemple toutes les 4 ms).

2) Lors de chaque séquencement, le système exécute jusqu'à 3000 lignes d'instructions VAL 3. Il passe à la tâche suivante lorsqu'une ligne d'instructions ne peut être terminée immédiatement (erreur d'exécution, attente d'un signal, tâche arrêtée, ...). En pratique, une tâche synchrone se termine souvent explicitement par l'instruction delay(0) qui oblige le système à passer au séquencement de la tâche suivante.

3) Les tâches synchrones de même période sont séquencées suivant leur ordre de création.

5.6 - ERREUR DE CADENCE

M0005609.1

Si la durée d'exécution d'une tâche synchrone VAL 3 est supérieure à la période définie, le cycle en cours se termine normalement mais le cycle suivant est annulé. Cette erreur de cadence est signalée à l'application VAL 3 en mettant à **true** la variable booléenne spécialement prévue à cet effet lors de la création de la tâche. Ainsi, cette variable booléenne indique au début de chaque cycle si le séquencement précédent s'est exécuté entièrement ou non.

5.7 - RAFRAÎCHISSEMENT DES ENTRÉES/SORTIES

M0005610.1

Concernant l'actualisation des entrées/sorties, il existe 2 groupes de cartes :

- **Groupe de cartes du système :**

  CpuIO, DsiIO, FastIO, PowerSupplyIO, Rsi9IO, StarcIO.

- **Groupe de cartes utilisateur :**

  J206 (Maître EtherCAT), J207/J208 (Ethernet temps réel), Fieldbus (Cartes Hilscher CIFX50E-xx PCIe).

Le temps de cycle d'actualisation du groupe de cartes du système est défini par le temps de cycle du système. Le temps de cycle d'actualisation du groupe de cartes utilisateur est, quant à lui, défini en sélectionnant un multiple du temps de cycle du système.

L'actualisation du groupe de cartes utilisateur peut être configurée selon 2 modes différents :

- Mode par défaut.
- Mode sans gigue.

Description des figures de temporisation ci-dessous :

Nom de tâche	Description
Synchro task	Cette tâche gère la trajectoire du robot et l'actualisation des cartes d'E/S du système.
IO refresh task	Cette tâche gère l'actualisation des cartes d'E/S.
VAL3 synchro	Cette tâche gère les tâches synchrones VAL 3 (voir chapitre 5.5).
VAL3 asynchro	Cette tâche gère les tâches VAL 3 (voir chapitre 5.1).

S : Signifie groupe de carte d'E/S du Système.

U : Signifie groupe de carte d'E/S Utilisateur.

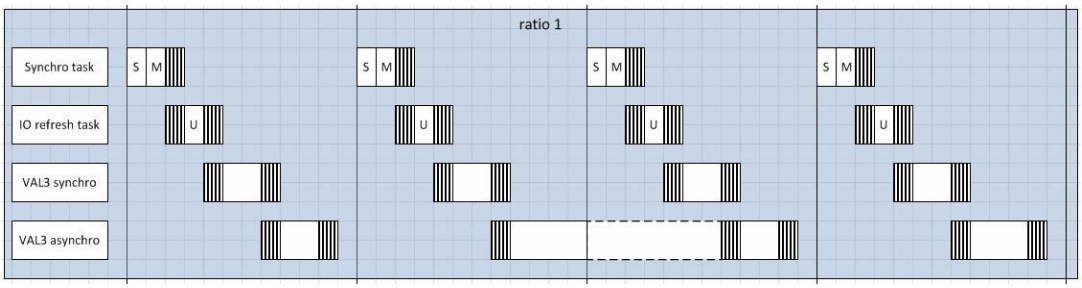
M : Signifie processus de Mouvement, générateur de trajectoire.

: Montre la gigue d'exécution de la tâche.

5.7.1 - ACTUALISATION DES CARTES UTILISATEUR MODE PAR DÉFAUT

M0005611.1

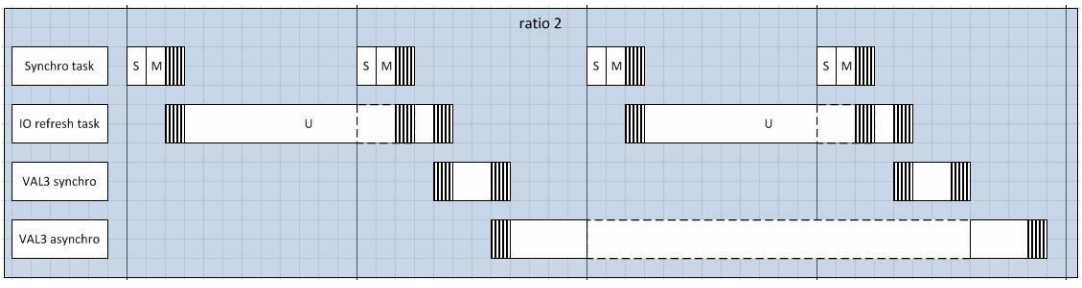
La figure ci-dessous représente l'exécution des différentes tâches lorsque le ratio du cycle d'actualisation du groupe de cartes utilisateur a la valeur 1.



I0002062

Si le temps d'actualisation de la carte d'E/S utilisateur est trop long (en fonction du nombre d'E/S) un dépassement de délai se produit pendant la tâche d'actualisation des E/S. Dans ce cas, le ratio d'actualisation de la carte d'E/S utilisateur doit être augmenté.

La figure ci-dessous représente l'exécution des différentes tâches lorsque le ratio du cycle d'actualisation des cartes utilisateur a la valeur 2.



I0002063

Remarque :

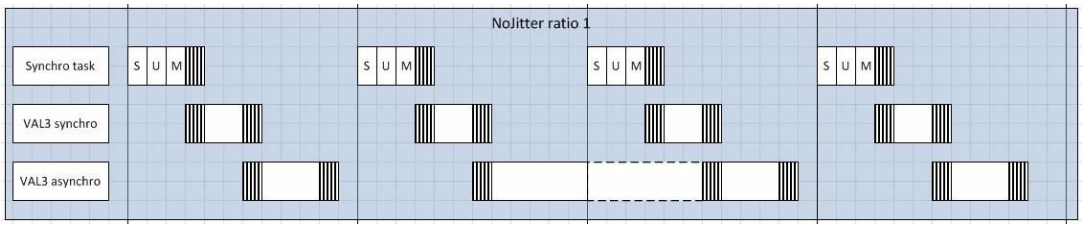
- Les valeurs des E/S utilisateur peuvent changer au cours d'un cycle "VAL3 asynchro".
- Un dépassement de délai de la tâche "IO refresh task" n'interrompt pas le robot mais toutes les valeurs des E/S utilisateur sont invalides. Le ratio d'actualisation des cartes utilisateur peut être modifié depuis le MCP et n'exige aucun redémarrage.

5.7.2 - ACTUALISATION DES CARTES UTILISATEUR MODE SANS GIGUE

M0005612.1

Dans ce mode, la tâche d'actualisation des E/S utilisateur n'est pas abandonnée et les cartes utilisateur sont actualisées dans le contexte de la tâche de synchronisation.

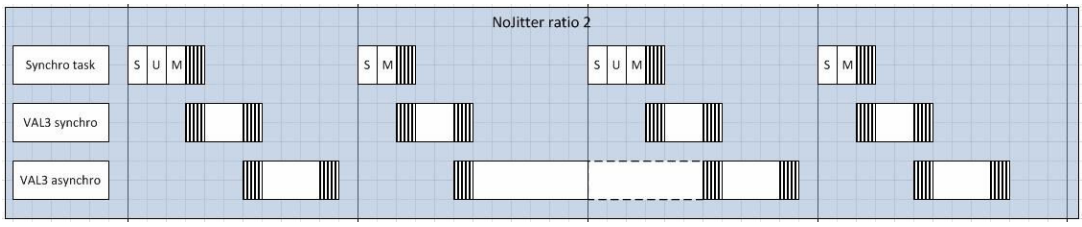
La figure ci-dessous représente l'exécution des différentes tâches lorsque le ratio du cycle d'actualisation du groupe de cartes utilisateur a la valeur 1.



I0002064

Si le temps d'actualisation de la carte utilisateur (U) est trop long (en fonction du nombre d'E/S) un dépassement de délai de la "synchro task" se produit. Ce dépassement de délai est considéré comme une erreur, le robot est arrêté et un redémarrage est nécessaire.

La figure ci-dessous représente l'exécution des différentes tâches lorsque le ratio du cycle d'actualisation du groupe de cartes utilisateur a la valeur 2.



I0002065

Il convient de remarquer que les valeurs des E/S utilisateur peuvent changer au cours d'un cycle "VAL3 asynchro".

5.7.3 - COMMENT SÉLECTIONNER LE MODE D'ACTUALISATION DES CARTES UTILISATEUR ET DÉFINIR LE RATIO DU CYCLE

M0005613.1

Le mode d'actualisation des cartes utilisateur et le ratio cyclique peuvent être modifiés dans le fichier usr/configs/cell.cfx.

```
<UserIORefresh>
 <string name = "mode" value = "MODE" />
 <Float name = "cyclicRatio" value = "RATIO" />
</UserIORefresh>
```

Mode	Ratio
default	-1 : utiliser la valeur par défaut (valeur 1). 1 à N : Ratio du temps de cycle du système (par défaut 4 ms).
noJitter	

Par exemple, avec le temps de cycle du système par défaut (4 ms)

Ratio	Ratio
1	4 ms
2	8 ms
3	12 ms
...	...

5.8 - SYNCHRONISATION

M0005614.1

Il est parfois nécessaire de synchroniser plusieurs tâches avant qu'elles ne poursuivent leur exécution.

Si l'on connaît a priori la durée d'exécution de chacune des tâches, cette synchronisation peut se faire par simple attente d'un signal émis par la tâche la plus lente. Mais lorsqu'on ne sait pas quelle tâche sera la plus lente, il faut utiliser un mécanisme de synchronisation plus complexe dont un exemple de programmation VAL 3 est donné ci-dessous.

Exemples

```
// Synchronization program for N tasks
```

Le programme synchro(num& n, bool& bSynch, num nN) ci-après doit être appelé dans chaque tâche à synchroniser.

La variable n doit être initialisée à 0, bSynch à **false** et nN au nombre de tâches à synchroniser.

```
begin
 n = n + 1
 // Task synchronization waiting instruction
 // makes sure all tasks are waiting here to resume operation
 wait((n==nN) or (bSynch==true))
 bSynch = true
 n = n - 1
 // Task release waiting instruction
 // makes sure all tasks have resumed operation to clear synch context
 wait((n==0) or (bSynch == false))
 bSynch = false
end
```

5.9 - PARTAGE DE RESSOURCE

M0005866.1

Quand plusieurs tâches utilisent le même système ou la même ressource de cellule (données globales, écran, clavier, robot, etc.). il est important d'éviter tout conflit entre elles.

On peut pour cela utiliser un mécanisme d'exclusion mutuelle **(’mutex’)** qui protège une ressource en autorisant son accès à une seule tâche à la fois. Un exemple de programmation de mutex en VAL 3 est donné ci-dessous.

Exemples

Cet affichage de programme (num c) remplit l'écran des mêmes caractères, en vérifiant qu'aucune autre tâche n'écrit sur l'écran en même temps avec le même programme. **bScreen** doit être initialisé à **false**.

```
begin
 // make sure only one task accesses the screen at one time
  setMutex(bScreen)
  c=c%10
  // write something to the screen with chars
  sOutput="Hello world"
  // wait for screen refresh
  delay(0.2)
  // let other tasks access the screen now
  bScreen=false
end
```

5.10 - VAL 3 WATCHDOG

M0005615.1

Le VAL 3 WatchDog est un moyen de sécuriser l'utilisation du robot. Il permet d'assurer que le robot ne peut être utilisé que si une tâche VAL 3 est en cours.

Le système peut être configuré avec ou sans la fonction VAL 3 WatchDog. Quand le système est configuré avec la fonction VAL 3 WatchDog, l'alimentation du robot ne peut être activée que si une application VAL 3 maintient le WatchDog activé. Pour maintenir le WatchDog activé, la fonction VAL 3 wdgRefresh() dédiée doit être exécutée avant que le délai du **val3WatchdogPeriod** ne soit écoulé.

Le WatchDog est configuré avec le paramètre **WatchdogPeriod** dans le fichier de configuration cell.cfx :

val3WatchdogPeriod<=0	Le système démarre sans WatchDog. La valeur par défaut de **val3WatchdogPeriod** est -1 (sans WatchDog).
val3WatchdogPeriod>0 (en second)	Le système démarre avec le WatchDog activé. L'alimentation électrique du robot peut être activée uniquement après que le WatchDog a démarré. Quand le WatchDog est activé, un message est enregistré dans l'historique au démarrage du système.

La fonction wdgRefresh() de VAL 3 l'active :

bool **wdgRefresh**()

M0005616.1

Fonction

Cette fonction démarre et actualise le WatchDog. Elle renvoie l'état actuel du WatchDog.

false	La fonction a démarré le WatchDog (premier appel de la fonction ou en cas de dépassement du délai du WatchDog).
true	Le WatchDog est activé et l'actualisation continue...

Lorsqu'un dépassement de délai du WatchDog a lieu, le robot effectue un arrêt cat 1 (**EstopSoft**). Le rappel de la fonction wdgRefresh supprime immédiatement le Estop.

L'utilisation de la fonction wdgRefresh quand le système est configuré sans **WatchDog** génère une **Runtime Error 045**. Dans ce cas, il n'est plus possible de mettre le robot sous tension (redémarrage nécessaire).

void **taskSuspend**(string sName) M0005862.1

Fonction

Cette instruction suspend l'exécution de la tâche **sName**.

Si la tâche est déjà dans l'état **STOPPED**, l'instruction est sans effet.

Une erreur d'exécution est générée si **sName** ne correspond à aucune tâche VAL 3 ou correspond à une tâche VAL 3 créée par une autre librairie.

Voir aussi

taskResume

taskKill

void **taskResume**(string sName, num nSkip) M0005861.1

Fonction

Cette instruction reprend l'exécution de la tâche **sName** sur la ligne située à **nSkip** lignes d'instructions avant ou après la ligne courante.

Si **nSkip** est négatif, l'exécution reprend avant la ligne courante. Si la tâche n'est pas dans l'état **STOPPED**, l'instruction est sans effet.

Une erreur d'exécution est générée si **sName** ne correspond pas à une tâche VAL 3, correspond à une tâche VAL 3 créée par une autre librairie, ou s'il n'y a pas de ligne d'instruction à l'endroit du **nSkip** spécifié.

DANGER
Si la tâche suspendue commande des mouvements, il est conseillé d'utiliser **nSkip** = 1 pour assurer la continuité de la trajectoire.

Voir aussi

taskSuspend

taskKill

void **taskKill**(string sName) M0005860.1

Fonction

Cette instruction suspend puis supprime la tâche **sName**. Après l'exécution de cette instruction, la tâche **sName** n'est plus présente dans le système.

S'il n'existe pas de tâche **sName** ou si la tâche **sName** a été créée par une autre librairie, l'instruction n'a aucun effet.

Voir aussi

taskSuspend

taskCreate

void **setMutex**(bool& bMutex) M0005618.1

Fonction

Cette instruction attend que la variable **bMutex** soit false puis la règle à true.

Cette instruction est nécessaire pour utiliser une variable booléenne comme mécanisme d'exclusion mutuelle en vue de la protection de ressources partagées (voir chapitre 5.9).

string **help**(num nErrorCode) M0005619.1

Fonction

Cette instruction renvoie la description du code d'erreur d'exécution spécifié avec le paramètre **nErrorCode**. La description est donnée dans la langue actuelle du contrôleur.

Exemples

Ce programme vérifie s'il y a une erreur dans la tâche "robot" et affiche le code d'erreur pour l'opérateur s'il y en a une.

```
nErrorCode=taskStatus("robot")
if (nErrorCode > 1)
 sOutput=help(nErrorCode)
endIf
```

num **taskStatus**(string sName) M0005776.1

Fonction

Cette instruction renvoie le statut actuel de la tâche **sName** ou le code d'erreur d'exécution de la tâche si elle est en erreur :

Code	Description
-1	Aucune tâche **sName** n'a été créée par la librairie ou l'application actuelle
0	La tâche **sName** est suspendue sans erreur d'exécution (instruction taskSuspend() ou mode de débogage)
1	La tâche sName créée par la librairie ou l'application actuelle est en cours d'exécution

Voir aussi

Le chapitre 15.1 présente toutes les erreurs de temps d'exécution possibles.

taskResume

taskKill

void **taskCreate** string sName, num nPriority, program(...) M0005859.1

Fonction

Cette instruction crée et lance la tâche **sName**.

sName doit contenir au moins 1 caractère choisi parmi **"a..zA..Z0..9_"**. Aucune autre tâche créée par la même librairie ne doit porter le même nom.

L'exécution de **sName** commence par l'appel de **program** avec les paramètres spécifiés. Il n'est pas possible d'utiliser une variable locale pour un paramètre transmis par référence, afin d'assurer que la variable ne sera pas effacée avant la fin de la tâche.

La tâche se terminera par défaut avec la dernière ligne d'instructions de **program**, ou plus tôt si elle est détruite explicitement.

nPriority doit être compris entre **1** et **100**. A chaque séquencement de la tâche, le système exécutera un nombre de ligne d'instructions égal à **nPriority**, ou moins si une instruction bloquante est rencontrée (voir le chapitre Séquencement).

Une erreur d'exécution est générée si le système n'a pas assez de mémoire pour créer la tâche, si **sName** n'est pas valide ou déjà utilisé dans la même librairie, ou si **nPriority** n'est pas valide.

Exemples

```
// start a new task to read a message
taskCreate "t1", 10, read(sMessage)
// waits for the end of task t1
wait(taskStatus("t1") == -1)
// Use the message
sOutput=sMessage
```

Voir aussi

taskSuspend

taskKill

taskStatus

void **taskCreateSync** string sName, num nPeriod, bool& bOverrun, program(...) M0005864.1

Fonction

Cette instruction crée et lance une tâche synchrone.

L'exécution de la tâche commence par l'appel du programme spécifié avec les paramètres spécifiés. Il n'est pas possible d'utiliser une variable locale pour un paramètre transmis par référence, afin d'assurer que la variable ne sera pas effacée avant la fin de la tâche.

Une erreur d'exécution est générée si le système ne possède pas la mémoire nécessaire pour pouvoir créer la tâche ou si un ou plusieurs paramètres ne sont pas valides.

Pour une présentation détaillée des tâches synchrones (voir chapitre 5.5).

Paramètres

string sName	Nom de la tâche à créer. Doit contenir au moins 1 caractère choisi parmi "_a..zA..Z0..9". Deux tâches appartenant à la même application ou librairie ne peuvent porter le même nom.

num nPeriod	Période d'exécution en secondes de la tâche à créer (s). La valeur spécifiée est arrondie à un multiple de la période de l'horloge du système. N'importe quelle période positive est prise en charge mais le système ne prend en charge que 2 périodes différentes de tâches synchrones à la fois. Par défaut, la période d'horloge du système a la valeur 4 ms (0.004s).
bool& bOverrun	Variable booléenne indiquant les erreurs de cadence. Seules les variables globales sont supportées, afin de garantir que la variable ne sera pas supprimée avant la tâche.
program	Nom du programme VAL 3 à appeler lorsque la tâche est lancée, les paramètres correspondants apparaissent entre parenthèses.

Exemples

```
// Create a supervisor task scheduled every 20 ms
taskCreateSync "supervisor", 0.02, bSupervisor, supervisor()
```

void **wait**(bool bCondition) M0005858.1

Fonction

Cette instruction met la tâche courante en attente jusqu'à ce que **bCondition** soit **true**.

La tâche reste **RUNNING** pendant la durée de l'attente. Si **bCondition** est **true** lors de la première évaluation, l'exécution se poursuit immédiatement sur la même tâche (il n'y a pas séquencement de la tâche suivante).

Voir aussi

delay

watch

void **delay**(num nSeconds) M0005857.1

Fonction

Cette instruction met la tâche courante en attente pendant **nSeconds**.

La tâche reste **RUNNING** pendant la durée de l'attente. Si **nSeconds** est négative ou nulle, le système procède immédiatement au séquencement de la tâche VAL 3 suivante.

Exemples

Ce programme fonctionne en boucle et veille à ne pas utiliser inutilement les ressources du CPU :

```
while bExit==false
 sOutput=toString("", clock()* 10)
 // let another task perform its operation immediately
 delay(0)
endWhile
```

Voir aussi

clock

watch

Fonction

Cette instruction renvoie la valeur actuelle de l'horloge système interne, exprimée en secondes.

La précision de l'horloge interne du système est la milliseconde. Elle est initialisée à **0** lors du démarrage du contrôleur, et n'a donc aucune relation avec l'heure calendaire.

Exemples

Pour calculer le délai d'exécution entre deux instructions, enregistrer la valeur d'horloge avant la première instruction :

```
nStart=clock()
```

Après la dernière instruction, calculer le délai d'exécution avec :

```
nDelay = clock()-nStart
```

Voir aussi

delay

watch

bool **watch**(bool bCondition, num nSeconds) M0005778.1

Fonction

Cette instruction met la tâche courante en attente jusqu'à ce que **bCondition** soit **true** ou jusqu'à ce que **nSeconds** secondes soient écoulées.

Renvoie **true** si l'attente se termine lors que **bCondition** est **true**, sinon **false** lorsque l'attente se termine parce que le délai est écoulé.

La tâche reste **RUNNING** pendant la durée de l'attente. Si **bCondition** est **true** à la première évaluation, la tâche est exécutée immédiatement ; dans le cas contraire, le système séquence les VAL 3 autres tâches (même si **nSeconds** est inférieur ou égal à **0**).

Exemples

Ce programme attend un signal et affiche un message d'erreur après 20 s.

```
if (watch (diSignal==true, 20)) == false
 sOutput="Error: waiting for Signal"
 wait(diSignal==true)
endIf
```

Voir aussi

delay

wait

clock

6 - LIBRAIRIES

M0005620.1

6.1 - DÉFINITION

M0005621.1

Une librairie VAL 3 est une application VAL 3 contenant des variables ou programmes qui peuvent être réutilisés par une autre application ou par d'autres librairies VAL 3.

Comme la librairie VAL 3 est une application VAL 3, elle contient les composants suivants :

- un ensemble de programmes : les instructions VAL 3 à exécuter
- un ensemble de variables globales : les données de la librairie
- un ensemble de librairies : les instructions externes et variables utilisées par la librairie

Lorsqu'une librairie est en cours d'exécution, elle peut contenir également :

- un ensemble de tâches : les programmes propres à la librairie en cours d'exécution

Toute application peut être utilisée comme librairie, et toute librairie peut être utilisée comme application, si les programmes **start**() et **stop**() y sont définis.

6.2 - INTERFACE

M0005622.1

Les programmes et variables globales d'une librairie peuvent être publics ou privés. Seuls les programmes et variables publics sont accessibles en dehors de la librairie. Les programmes et variables globales privés ne peuvent être utilisés que par les programmes de la librairie.

Tous les programmes et variables globaux publics d'une librairie constituent l'interface de celle-ci : plusieurs librairies différentes peuvent avoir la même interface, tant que leurs programmes et variables publics utilisent les mêmes noms.

Les tâches créées par un programme d'une librairie sont toujours privées, c'est-à-dire qu'elles ne sont accessibles que par cette librairie.

6.3 - IDENTIFIANT D'INTERFACE

M0005623.1

Pour pouvoir utiliser une librairie, une application doit d'abord déclarer un identifiant qui lui soit affecté, puis demander, dans un programme, de charger la librairie dans la mémoire en utilisant cet identifiant.

L'identifiant est affecté à l'interface de la librairie et non pas à la librairie elle-même. Toute librairie possédant la même interface peut ensuite être chargée en utilisant cet identifiant. Ce mécanisme peut être utilisé, par exemple, pour définir une librairie pour toutes les parties d'une application, puis ne charger que la partie utilisée actuellement par chaque cycle.

6.4 - CONTENU

M0005624.1

Une librairie n'a pas de contenu obligatoire : elle peut ne contenir que des programmes, que des variables, ou les deux.

L'accès au contenu d'une librairie se fait en écrivant le nom de l'identifiant suivi de ':' avant le nom du programme ou de la donnée de la librairie, par exemple :

```
// Load the "article_7" library under the "article" identifier
article:libLoad("article_7")
// Display as title the content of the 'sName' variable of the article_7 library
title(article:sName)
// Call the init() program for the current article
call article:init()
```

L'accès au contenu d'une librairie qui n'a pas encore été chargée en mémoire entraîne une erreur d'exécution.

6.5 - CRYPTAGE

M0005799.1

VAL 3 peut utiliser des librairies cryptées, sur la base des outils de compression ZIP et de cryptage courants.

Une librairie cryptée est un fichier ZIP standard du contenu du répertoire de la librairie (attention : le cryptage 128 avancé et 256 AES n'est pas possible). Le nom du fichier ZIP doit être suivi de l'extension ".zip" et doit comporter moins de 15 caractères (y compris l'extension). Pour obtenir un cryptage robuste, le mot de passe du ZIP doit comporter plus de 10 caractères et il ne doit pas apparaître dans un dictionnaire.

Mot de passe secret, mot de passe public

L'interpréteur VAL 3 doit avoir accès au mot de passe ZIP secret pour charger une librairie cryptée. Un outil pour PC est prévu pour cela dans la Stäubli Robotics Suite pour coder le mot de passe ZIP secret dans un mot de passe VAL 3 public. Le mot de passe VAL 3 public permet d'utiliser la bibliothèque cryptée dans un programme VAL 3. Le contenu de la librairie reste toutefois secret puisque le mot de passe ZIP ne peut pas être calculé à partir du mot de passe VAL 3.

Cryptage de projet

L'application de démarrage ne peut pas être cryptée directement sur le contrôleur. Une application complète peut être cryptée :

- en déclarant son programme start() comme public.
- en créant une autre application qui charge simplement l'application cryptée sous la forme d'une librairie et appelle son programme start().

6.6 - CHARGEMENT ET DÉCHARGEMENT

M0005625.1

Lorsqu'une application VAL 3 est ouverte, toutes les librairies déclarées sont analysées pour construire les interfaces correspondantes. Cette étape ne charge pas les librairies en mémoire.

DANGER
Il ne peut pas y avoir de références circulaires entre les librairies. Si la librairie A utilise la librairie B, la librairie B ne peut pas utiliser la librairie A.

Lors du chargement d'une librairie, ses variables globales sont initialisées et ses programmes vérifiés pour détecter d'éventuelles erreurs de syntaxe. Quand plusieurs identifiants de librairie différents chargent la même librairie sur le disque, ils partagent la même librairie en mémoire. La librairie est ainsi chargée une seule fois et réutilisée par tous les identifiants. Dans l'exemple ci-dessous, lib1 et lib2 utilisent la même variable en mémoire.

```
lib1:libLoad("appData")
lib1:sText = "lib1"
lib2:libLoad("appData")
// The change on lib2:sText applies here also to lib1:sText
lib2:sText = "lib2"
```

Le déchargement d'une librairie n'est pas nécessaire, il se fait automatiquement lorsque l'application se termine ou lorsqu'une nouvelle librairie est chargée à la place d'une autre.

Lorsqu'une application VAL 3 est arrêtée depuis l'interface utilisateur du contrôleur MCP, le programme **stop**() est d'abord exécuté, puis toutes les tâches de l'application et de ses librairies, s'il en reste, sont détruites.

Chemin d'accès

Les instructions libLoad(), libSave() et libDelete() utilisent un chemin d'accès à une librairie, spécifié sous forme de chaîne de caractères. Un chemin d'accès contient une racine (facultative), un chemin (facultatif), et un nom de librairie, suivant le format :

root://Path/Name

La racine spécifie le support de fichier : **"Floppy"** pour une disquette, **"USB0"** pour un dispositif sur un port **USB** (clé, disquette), **"Disk"** pour le disque Flash du contrôleur, ou le nom d'une connexion **Ftp** défini dans le contrôleur pour un accès au réseau.

Par défaut, la racine est **"Disk"** et le chemin vide.

Exemples

```
// load library "article_1" on Disk Equivalent to"Disk://article_1"
article:libLoad("article_1")
// Save library on USB device
article:libSave("USB0://articles/article_1")
// Load the default article defined within the current application
article:libLoad("./defaultArticle")
```

Codes d'erreur

Les fonctions VAL 3 de manipulation de librairie ne génèrent jamais d'erreur d'exécution, mais renvoient un code d'erreur permettant de vérifier le résultat de l'instruction et de résoudre les problèmes éventuels.

Code	Description
0	Pas d'erreur
1	La librairie a bien été chargée mais avec des avertissements. Consulter l'historique des événements pour plus d'informations sur la cause de l'avertissement.
10	L'identifiant de librairie n'a pas été initialisé par libLoad().
11	Librairie chargée, mais l'interface publique ne correspond pas. Une erreur d'exécution 80 se produit si le programme VAL 3 tente d'accéder aux éléments manquants. Voir l'instruction libExist ou les instructions libExist.
12	Chargement de librairie impossible : la librairie contient des données ou des programmes invalides, ou, si elle est cryptée, le mot de passe spécifié n'est pas correct.
13	Déchargement de librairie impossible : La librairie est en cours d'exécution par une autre tâche.
14	Déchargement de librairie impossible : La librairie comprend une tâche VAL 3 en cours. Toutes les tâches créées par les programmes VAL 3 à partir de la librairie doivent être terminées avant le déchargement de la librairie.
15	La librairie ne peut pas être chargée parce qu'une donnée est liée à un élément graphique. Cette donnée doit être déliée avant libLoad() avec l'instruction userPageUnBind() et peut être liée à nouveau après libLoad() avec l'instruction userPageBind().
20	Erreur d'accès fichier : la racine du chemin est invalide.
21	Erreur d'accès fichier : le chemin est invalide.
22	Erreur d'accès fichier : le nom est invalide.
23	Librairie cryptée attendue. La librairie n'est pas ou pas correctement cryptée.
24	La librairie ne peut pas être chargée parce qu'elle utilise un type d'utilisateur qui ne peut pas être chargé.

Code	Description
>=30	Erreur de lecture/écriture sur un fichier.
31	Impossible d'enregistrer la librairie : le chemin spécifié contient déjà une librairie. Pour remplacer une librairie sur le disque, commencer par l'effacer avec libDelete().
32	Message du pilote de périphérique "Dispositif introuvable"
33	Message du pilote de périphérique "Erreur de dispositif"
34	Message du pilote de périphérique "Fin de temporisation du dispositif"
35	Message du pilote de périphérique "Dispositif protégé en écriture"
36	Message du pilote de périphérique "Pas de disque"
37	Message du pilote de périphérique "Disque non formaté"
38	Message du pilote de périphérique "Disque plein"
39	Message du pilote de périphérique "Fichier introuvable"
40	Message du pilote de périphérique "Fichier en lecture seule"
41	Message du pilote de périphérique "Connexion refusée"
42	Message du pilote de périphérique "Pas de réponse du serveur FTP"
43	Message du pilote de périphérique "Erreur du kernel FTP"
44	Message du pilote de périphérique "Erreur de paramètres FTP"
45	Message du pilote de périphérique "Erreur d'accès FTP"
46	Message du pilote de périphérique "Disque FTP plein"
47	Message du pilote de périphérique "Nom d'utilisateur FTP invalide"
48	Message du pilote de périphérique "Pas de connexion FTP définie"

6.7 - INSTRUCTIONS

M0005626.1

num identifier:**libLoad**(string ...) M0005628.1

num identifier:libLoad(string **sPath**)

num identifier:libLoad(string **sPath**, string **sPassword**)

Fonction

Cette instruction initialise l'identifiant de librairie en chargeant le programme et les variables de librairie dans la mémoire après le **sPath** spécifié. Le paramètre **sPassword** spécifié (facultatif) est utilisé comme clé de déchiffrage pour les librairies cryptées. Le **sPassword** spécifié doit être le mot de passe VAL 3 public calculé à partir du mot de passe ZIP secret de la librairie cryptée (voir chapitre 6.5).

L'instruction renvoie **0** après un chargement réussi ou un code d'erreur de chargement de librairie si des tâches créées par la librairie sont encore en cours d'exécution, si le chemin d'accès à la librairie est incorrect, si la librairie contient des erreurs de syntaxe ou si elle ne correspond pas à l'interface déclarée pour l'identifiant.

Voir aussi

libSave

num identifier:**libSave**(), num **libSave**() M0005779.1

Fonction

Cette instruction sauvegarde les variables et programmes attribués à l'identifiant de la librairie. Si libSave() est appelé sans identifiant, l'application contenant l'instruction libSave() est sauvegardée. Si un paramètre est spécifié, la sauvegarde se fait au **sPath** indiqué. Sinon, la sauvegarde se fait au chemin spécifié lors du chargement.

L'instruction renvoie **0** si le contenu a été sauvegardé ou un code d'erreur si l'identifiant n'a pas été initialisé, si le chemin est incorrect, si une erreur d'écriture s'est produite ou si le chemin indiqué contient déjà une librairie.

DANGER
Certains appareils tels que le disque Flash du contrôleur n'acceptent qu'un nombre limité d'accès en écriture. Si libSave() est utilisé fréquemment dans un programme (une fois ou plus par minute), il doit l'être sur un support acceptant cette fréquence.

Voir aussi

libDelete

num **libDelete**(string sPath) M0005780.1

Fonction

Cette instruction supprime la librairie située dans le **sPath** indiqué.

L'instruction renvoie **0** si la librairie indiquée n'existe pas ou a été supprimée et un code d'erreur si l'identifiant n'a pas été initialisé, si le chemin est incorrect ou si une erreur d'écriture se produit.

Voir aussi

libSave

libPath

string identifier:**libPath**(), string **libPath**() M0005781.1

Fonction

Cette instruction renvoie le chemin d'accès de la librairie associé à l'identifiant, ou de l'application ou librairie appelante si aucun identifiant n'est spécifié.

Voir aussi

libList

bool **libList**(string sPath**,** string& sContents[]) M0005835.1

Fonction

Cette instruction liste le contenu du chemin **sPath** spécifié dans le tableau **sContents**. Renvoie **true** si le tableau **sContents** contient le résultat complet ou **false** si le tableau est trop petit pour contenir toute la liste.

Tous les éléments du tableau **sContents** sont d'abord initialisés à "" (chaîne vide). Après l'exécution de libList(), la fin de la liste est recherchée dans la première chaîne vide du tableau **sContents**.

Si **sContents** est une variable globale, le tableau est automatiquement agrandi autant que nécessaire pour permettre l'enregistrement du résultat complet.

Voir aussi

libPath

bool identifier:**libExist**(string sSymbolName) M0005629.1

Fonction

L'instruction libExist vérifie si un symbole (variable globale ou programme) est défini dans une librairie. Elle renvoie un résultat "true" si le symbole existe et s'il est accessible (public), sinon "false".

Le nom de symbole d'un programme doit se terminer par "()" : "mySymbol" désigne un nom de variable, tandis que "mySymbol()" est un nom de programme.

L'instruction libExist est utile pour tester si une entrée/sortie est définie sur un contrôleur ; elle est également utile pour gérer l'évolution de l'interface d'une librairie et adapter son utilisation selon s'il s'agit d'une version récente ou ancienne de l'interface.

Exemples

L'exemple suivant teste l'interface d'une librairie.

```
// Load part library
nLoadCode = part:libLoad(sPartPath)
// part:sVersion was not defined in the first version of the library
// Test if this library defines it
if (nLoadCode==0) or (nLoadCode==11)
 if (part:libExist("sVersion")==false)
   // initial version
   sLibVersion = "v1.0"
 else
   sLibVersion = part:sVersion
 endIf
endIf
```

Ce programme appelle le programme "init" dans la librairie "protocol", si elle existe :

```
if(protocol:libExist("init()")==true)
 call protocol:init()
endIf
```

Voir aussi

isDefined

7 - TYPE D'UTILISATEUR

M0005630.1

7.1 - DÉFINITION

M0005631.1

Un type d'utilisateur est un type structuré, défini dans une application VAL 3 dans laquelle il peut être utilisé comme type standard. Un type d'utilisateur combine des types simples, structurés ou même d'autres types d'utilisateur dans un nouveau type de données. Les types d'utilisateur élèvent le niveau d'abstraction des programmes et les rendent plus facile à comprendre, à développer et à entretenir. Ils nécessitent toutefois davantage de travail de conception pour identifier les types convenant le mieux aux contraintes de l'application.

Un type d'utilisateur est un ensemble de champs dont chacun comprend :

- un nom : une chaîne de caractères
- un type de variable (simple, structuré ou type d'utilisateur)
- un conteneur de variables (élément, tableau ou collection)
- un ensemble de valeurs par défaut

Les champs des types standard de VAL 3 utilisent toujours un conteneur élément (avec une seule valeur). Les champs des types d'utilisateur peuvent utiliser des conteneurs tableau ou collection et peuvent donc contenir plusieurs valeurs. Le nombre d'éléments du champ définit le nombre par défaut d'éléments dans le conteneur de champ et la valeur par défaut de chacun de ces éléments. Dans une variable définie avec un type utilisateur, on peut changer à tout moment les valeurs des champs, mais aussi le nombre d'éléments dans les conteneurs de champs.

7.2 - CRÉATION

M0005632.1

Les champs d'un type d'utilisateur ont les mêmes caractéristiques que les variables globales d'une application VAL 3. C'est la raison pour laquelle la création d'un nouveau type d'utilisateur se résume à sélectionner une application VAL 3 et à l'associer à un nom.

- L'ensemble des variables globales publiques dans l'application sélectionnée définit l'ensemble des champs du type d'utilisateur avec les valeurs par défaut de ceux-ci.
- Le nom définit le nom à utiliser pour le nouveau type d'utilisateur dans l'application dans laquelle il est défini.

Les variables privées et les programmes de l'application utilisés comme définition du type ne sont pas pris en considération dans le type d'utilisateur.

Une fois qu'un nouveau type d'utilisateur est défini dans une application, il est possible de créer des variables de ce type. L'application ainsi obtenue peut aussi être utilisée comme définition de type.

7.3 - UTILISATION

Les champs d'une variable de type d'utilisateur sont accessibles à l'aide d'un ’.’ suivi du nom du champ : userVariable.field1.field2 désigne la valeur du champ 'field2' du champ 'field1' des données userVariable. La création ou la suppression d'éléments dans un conteneur de champ sont possibles, comme pour une variable, à l'aide des instructions insert(), delete(), append() ou resize(). Quand un nouvel élément est créé, chaque champ reçoit une valeur par défaut composée des éléments et de leurs valeurs définies dans l'application utilisée comme définition du type.

DANGER
Les champs de type point, outil ou repère ne sont pas liés par défaut.

L'opérateur ’=’ est toujours défini entre deux variables du même type d'utilisateur. Après l'exécution de l'opérateur ’=’, la variable de gauche est une copie de celle de droite : le conteneur des champs a le même nombre d'éléments et les éléments ont la même valeur.

8 - CONTRÔLE DU ROBOT

M0005634.1

Ce chapitre liste les instructions donnant accès à l'état des différentes parties du robot.

8.1 - INSTRUCTIONS GÉNÉRALES

M0005635.1

void **disablePower**()

M0005856.1

Fonction

Cette instruction couple l'alimentation du bras et attend que la coupure soit effective.

Si le bras est en mouvement, un arrêt rapide sur trajectoire est effectué avant la coupure de puissance.

Voir aussi

enablePower

isPowered

void **enablePower**()

M0005855.1

Fonction

En mode déporté, cette instruction met le bras sous tension.

Cette instruction n'a aucun effet en mode local, manuel ou test, ou lorsque la puissance est en cours de coupure. Elle crée toutefois un message dans le fichier d'historique des évènements afin d'éviter des tentatives répétées sans temporisation d'établir l'alimentation.

Exemples

```
// Switches on the power and waits for the arm power to be switched on
enablePower()
 if(watch(isPowered(), 5) == false)
 sOutput="Arm power supply cannot be switched on"
endIf
```

Voir aussi

disablePower

isPowered

bool **isPowered**(), bool **isPowered**(num& nStatus)

M0005782.1

Fonction

Cette instruction renvoie l'état d'alimentation du bras :

true : le bras est sous puissance.

false : le bras est hors tension ou est en cours de mise sous tension. **nStatus** sera actualisé avec la phase de tension en cours.

Phase	Valeurs	Commentaires
Désactivé	1	Alimentation coupée
Activation	2	L'établissement de la tension est en cours
activé	3	Alimentation établie
Désactivation	4	La coupure de l'alimentation est en cours

bool **isCalibrated**()

M0005636.1

Fonction

Cette instruction renvoie l'état de calibrage du robot :

true : tous les axes du robot sont calibrés

false : au moins un axe du robot n'est pas calibré

Fonction

Cette instruction renvoie le mode de marche courant du robot :

Mode	Etat	Mode de marche	Etat
0	**0**	**Invalide ou en transition**	-
1	**0**	**Manuel**	Mouvement programmé
	1		Mouvement de connexion
	2		Joint jogging
	3		(Frame jogging) cartésien
	4		Tool jogging
	5		Vers point (Point jogging)
	6		Hold
2	**0**	**Test**	Mouvement programmé (< 250 mm/s)
	1		Mouvement de connexion (< 250 mm/s)
	2		Mouvement programmé rapide ((> 250 mm/s))
	3		Hold
3	**0**	**Local**	Move (mouvement programmé)
	1		Move (mouvement de connexion)
	2		Hold
4	**0**	**Déporté**	Move (mouvement programmé)
	1		Move (mouvement de connexion)
	2		Hold

num **esStatus**() M0005840.1

Fonction

Cette instruction renvoie l'état du circuit d'arrêt d'urgence :

Code	Etat
0	Pas d'arrêt de sécurité
1	Attente de redémarrage de sécurité
2	SS1 - Condition d'arrêt de sécurité
3	SS2 - Condition d'arrêt de fonctionnement sûr

Voir aussi

workingMode

num **getMonitorSpeed**() M0005841.1

Fonction

Cette instruction renvoie la vitesse moniteur actuelle du robot (dans la plage [0, 100]).

Exemples

Ce programme, qui est appelé dans une tâche spécifique, vérifie si le premier cycle du robot est exécuté à petite vitesse :

```
while true
 if(nCycle < 2)
   if (getMonitorSpeed()> 10)
     stopMove()
     sOutput="For the first cycle the monitor speed must remain at 10%"
     wait(getMonitorSpeed()<= 10)
   endIf
   restartMove()
 endIf
 delay(0)
endWhile
```

Voir aussi

setMonitorSpeed

num **setMonitorSpeed**(num nSpeed) M0005842.1

Fonction

Cette instruction modifie la vitesse moniteur actuelle du robot. setMonitorSpeed() est toujours capable de réduire la vitesse moniteur. Pour augmenter celle-ci, setMonitorSpeed() n'est efficace que si le robot fonctionne en mode déporté et si l'opérateur n'a pas accès à la vitesse moniteur (quand le profil utilisateur courant ne permet pas l'utilisation des boutons de vitesses ou quand le MCP a été déconnecté).

Elle renvoie 0 si la vitesse moniteur a été modifiée, un code d'erreur négatif dans le cas contraire :

Code	Description
-1	Le robot n'est pas en mode déporté
-2	La vitesse moniteur est contrôlée par l'opérateur : modifier le profil d'utilisateur actuel pour supprimer l'accès de l'opérateur à la vitesse moniteur
-3	La vitesse spécifiée n'est pas acceptée : elle doit se situer dans la plage [0, 100]

Voir aussi

getMonitorSpeed

string **getVersion**(string sComponent)

Fonction

Cette instruction renvoie la version de différents composants matériels et logiciels du contrôleur du robot. Le tableau ci-dessous donne la liste des composants pris en charge et le format de la valeur renvoyée pour chacun d'entre eux :

Composant	Description
"System"	Version du système du contrôleur, telle que "s8.9.1cs9_BS1830"
"Arm Type"	Type de bras fixé au contrôleur, par exemple "tx2_90-S1" ou "ts2_60-S1-D25-L200"
"Arm Tuning"	Version du réglage du bras, par exemple "R3"
"Arm Mounting"	Montage du bras, par exemple "sol", "mur" ou "plafond"
"Arm S/N"	Numéro de série du bras, par exemple "F_19_00016047_A_07"
"Controller S/N"	Numéro de série du contrôleur, par exemple "F_19_00016047_C_07"
"Configuration Version"	Version du fichier de configuration du contrôleur, telle que "c2.028"
"System OS"	Indique la version du système d'exploitation ou "Windows" sur l'émulateur
Nom de la licence	Etat de la licence du logiciel du contrôleur : ■ "" (chaîne vide) : non installée ou période de démo terminée ■ "demo" : licence activée pour un temps limité ■ "enabled" : licence activée Les noms des licences de contrôleur installées (par exemple "alter", "remoteMcp", "oemLicense", "mcpMode"...) figurent dans la page Robot - Information du boîtier de commande du robot

Exemples

```
if getVersion("oemlicense")!="Enabled"
 sOutput="The oemLicense license is missing on the controller"
endIf
```

Voir aussi

getLicence

joint **getJntRef**(string sReferenceName)

Fonction

Cette fonction renvoie la valeur de la référence de l'articulation appelée **sReferenceName**. Si **sReferenceName** ne correspond à aucune référence existante, une erreur d'exécution est générée.

Exemples

```
jsafeRef1 = getJntRef("safeReference1")
jsafeRef2 = getJntRef("safeReference2")
```

joint **min**(joint, joint)

Calcule articulation par articulation le minimum des deux entrées.

Voir aussi

max

joint **max**(joint, joint) M0005843.1

Calcule articulation par articulation le maximum des deux entrées.

Voir aussi

min

8.2 - INSTRUCTIONS POUR LES ÉCONOMIES D'ÉNERGIE M0005639.1

Il est possible de réduire la consommation d'énergie du robot si celui-ci est inactif en coupant l'alimentation électrique des moteurs et de l'interface du codeur. Cette opération doit être gérée par l'application à l'aide des instructions suivantes.

num **hibernateRobot**() M0005845.1

Fonction

Cette fonction coupe l'alimentation électrique des moteurs et de l'interface du codeur.

Valeurs retournées :

Code	
0	Pas d'erreur.
-1	Fonction non prise en charge.
-2	Dépassement du délai d'attente pour la confirmation de la carte de sécurité.
-3	Le robot doit être désactivé pour activer l'économie d'énergie.
-4	Dépassement du délai d'attente pour l'arrêt des moteurs et de l'interface du codeur. Dans ce cas, la fonction wakeUpRobot doit être exécutée pour le rétablissement.

Voir aussi

wakeUpRobot

disablePower

num **wakeUpRobot**() M0005836.1

Fonction

Cette fonction rétablit le robot à l'état opérationnel.

Valeurs retournées :

Code	
0	Pas d'erreur. La fonction renvoie également 0 si hibernateRobot() n'a pas été exécutée avant.
-1	Fonction non prise en charge.
-2	Dépassement du délai d'attente de l'état prêt de l'interface du codeur.
-3	Dépassement du délai d'attente de l'état prêt de la carte de sécurité.

Voir aussi

hibernateRobot

8.3 - INSTRUCTIONS POUR LA GESTION DES PANNES D'ÉLECTRICITÉ

M0005640.1

Le connecteur J213 du tiroir informatique (CPT) permet d'utiliser une alimentation électrique 24 V externe. En cas d'utilisation d'une alimentation électrique de secours, l'application gère le comportement en cas de coupure de courant (en principe en enregistrant les données de l'application).

Par défaut, lorsqu'une coupure de courant est détectée (entrée powerSupplyIO//mainPowerOk) le contrôleur ferme tous les fichiers et verrouille l'accès au système de fichiers. Ce comportement peut être modifié en mettant le drapeau **cpuExtPowerSupply** à l'état vrai dans le fichier /usr/configs/controller.cfx.

Si le drapeau **cpuExtPowerSupply** est mis à l'état vrai, le contrôleur ne réagit pas en cas de coupure de courant. L'application VAL 3 est donc chargée de surveiller l'entrée powerSupplyIO/mainPowerOk. Lorsque l'application VAL 3 détecte une coupure de courant (powerSupplyIO/mainPowerOk a la valeur 0) toutes les données sont enregistrées. Après l'enregistrement des données, l'application peut : soit appeler la fonction prepareCpuShutdown() et attendre que le 24 V disparaisse soit attendre que le powerSupplyIO/mainPowerOk revienne et exécuter la fonction wakeUpRobot() pour attendre que le robot soit de nouveau opérationnel.

bool **hasCpuExtPowerSupply**() M0005838.1

Fonction

Cette fonction permet de lire le comportement en cours. Elle renvoie :

false : Quand le contrôleur contrôle et réagit à une coupure de courant (comportement par défaut).

true : Quand le contrôleur ne contrôle pas et ne réagit pas à une coupure de courant.

Voir aussi

prepareCpuShutdown

void **prepareCpuShutdown**() M0005850.1

Fonction

Cette fonction doit être exécutée depuis l'application VAL 3 avant de couper l'alimentation électrique externe du tiroir informatique. Elle exécute le comportement par défaut : Fermeture de tous les fichiers et verrouillage de l'accès au système de fichiers.

Voir aussi

hasCpuExtPowerSupply

8.4 - INSTRUCTIONS POUR LE TEST DES FREINS

M0005641.1

num **brakeTest**(num& nBrakeStatus), num **brakeTest**()

M0005642.1

Fonction

Cette fonction effectue une procédure d'essai des freins. Pour une description plus détaillée de la procédure, consulter le manuel de sécurité.

La valeur renvoyée correspond au résultat de la procédure :

Code	Description
0	Tous les freins ont été testés et sont OK.
1	Un des freins au moins n'a pas réussi le test. Il doit être remplacé.
Outil Valeurs négatives	Un des freins au moins renvoie une erreur (la procédure d'essai a échoué). Cf. les codes d'erreur ci-après.

Le paramètre optionnel **nBrakeStatus** du tableau peut être utilisé pour obtenir l'état de l'essai de chaque frein. **nBrakeStatus**[0] correspond au frein de l'axe 1, **nBrakeStatus**[1] correspond au frein de l'axe 2, ...

Code	Description
1	Test non exécuté parce non demandé (réservé à une utilisation future).
2	Le frein n'a pas réussi le test. Il doit être remplacé.
25	La capacité du frein est estimée à 25%. Il doit être remplacé pour assurer la sécurité de l'application.
50	Test réussi. La capacité du frein est d'au moins 50% et <75%.
75	Test réussi. La capacité du frein est d'au moins 75% et <100%.
100	Test réussi. La capacité du frein est d'au moins 100%.
Valeurs négatives	La procédure d'essai a échoué. Cf. les codes d'erreur ci-après.

Codes d'erreur :

Code	Description
-1	Mauvais mode de fonctionnement. Le mode de fonctionnement doit être "à distance".
-2	Le robot bouge. Le robot doit être arrêté pour pouvoir effectuer la procédure de test des freins.
-3	Le robot n'est pas en position brakeTest.
-5	Échec de la désactivation de la puissance.
-6	Échec de l'application de la puissance ; voir l'historique pour plus de précisions.
-7	Impossible d'empiler un mouvement nécessaire pour la procédure brakeTest. Le robot est dans une zone sûre ou a atteint une limite. Erreur d'exécution du mouvement : xxx
-8	Le mouvement ne peut pas déplacer le robot. La puissance est peut-être appliquée.
-13	Procédure brakeTest interrompue par l'utilisateur (killTask).
-14	Les données brakeTest manquent dans le fichier de configuration (system.zfx).
-15	Couple d'essai non atteint sur le robot.
-16	Coupure inopinée de la puissance du bras. Il est arrivé quelque chose qui désactive la puissance (fonction VAL3, EStop, configuration de sécurité...).
-17	brakeTest impossible : aucune rétrosignalisation de position de sécurité des encodeurs. Vérifier le statut de la carte DSI, désélectionner la sortie de carte DSI QnoSafePos.
-18	brakeTest impossible : le robot n'est pas étalonné.
-19	Mouvement non autorisé (stopMove() est peut-être appelé). Le mouvement doit être autorisé pour l'exécution de la procédure brakeTest.
-20	brakeTest : échec de l'initialisation de la carte Starc. La puissance est peut-être appliquée.
-21	brakeTest impossible : erreur lors des mouvements préliminaires avant l'essai de frein. Le nombre d'allers-retours est peut-être trop grand.

DANGER
La fonction enablePower de VAL 3 est désactivée pendant cette procédure.
Toute demande de disablePower annule la procédure.

8.5 - CONSIGNES DE SÉCURITÉ

M0005643.1

bool **setSafetyRestart**()

M0005644.1

Cette instruction effectue un redémarrage de la sécurité si nécessaire.

Un redémarrage de la sécurité est nécessaire quand esStatus() renvoie 1.

C'est une fonction bloquante.

Elle renvoie "true" quand un redémarrage de la sécurité a été effectué et esStatus() a renvoyé 0 après le redémarrage, sinon elle renvoie "false".

Exemples

```
if esStatus() == 1
  bBool = setSafetyRestart()
  if (bBool)
    sOutput="safety restart OK"
  else
    sOutput="safety restart KO"
  endif
endIf
```

Voir aussi

esStatus

num **getSafeRefStatus**() M0005645.1

Cette instruction renvoie l'état de la calibration de la sécurité sous la forme d'une valeur numérique.

Les valeurs qui peuvent être renvoyées sont :

Etat	Description
0	Référencé.
+/-1	Attention.
+/-2	Initialisé.
+/-3	Perdu.
+/-4	Réinitialisé.
+/-5	Invalide.
-100	Échec de la lecture de la configuration de la sécurité.
-1000	État indéterminé.

- Les états de -1 à -5 nécessitent une action de l'utilisateur. Consulter le manuel de sécurité pour plus d'informations.
- Les états -100 et -1000 signifient que le contrôleur n'a pas pu déterminer la calibration de la sécurité.

Exemples

```
if getSafeRefStatus() == 0
  sOutput="the safety is referenced"
endIf
```

bool **isInSafeRange**(joint jPos) M0005646.1

Renvoie "false" si au moins un des points de contrôle pour la position de l'articulation **jPos** se situe dans une zone actuellement interdite (Les zones interdites 3 et 4 peuvent changer avec les entrées USI-B et USI-C). Les limites d'orientation seront prises en compte ultérieurement.

bool **getSafeLimit**(joint jPos, mdesc& mDesc) M0005647.1

Sature le descripteur de mouvement avec les vitesses maximales autorisées dans la configuration de zone actuelle dans la position **jPos**. Renvoie la même chose que isInSafeRange.

num **safetyFault**(string& x_sMsg) M0005894.1

Renvoie le numéro de code de défaut d'arrêt d'urgence SS1 et actualise le message correspondant dans **x_sMsg** (en utilisant la langue actuelle).

9 - POSITIONS DU BRAS

M0005648.1

9.1 - INTRODUCTION

M0005649.1

Ce chapitre décrit les types de données VAL 3 permettant de programmer les positions du bras occupées dans une application VAL 3.

Deux types de positions sont définies en VAL 3 : des positions articulaires (type articulation joint) qui indiquent la position angulaire de chaque axe de rotation et la position linéaire de chaque axe linéaire, et des points cartésiens (type point) qui donnent la position cartésienne du centre de l'outil à l'extrémité du bras par rapport à un repère de référence.

Le type tool décrit un outil avec sa géométrie, utilisé pour le positionnement et le contrôle de vitesse du bras ; il décrit également le mode d'activation de l'outil (sortie numérique, temporisation).

Le type frame décrit un repère géométrique. L'utilisation de repères rend en particulier les manipulations géométriques sur les points plus simples et intuitives.

Le type trsf décrit une transformation géométrique. Il est utilisé par les types tool, point et frame.

Enfin, le type config décrit la notion plus avancée de configuration de bras.

Les relations entre ces différents types peuvent être résumées ainsi :

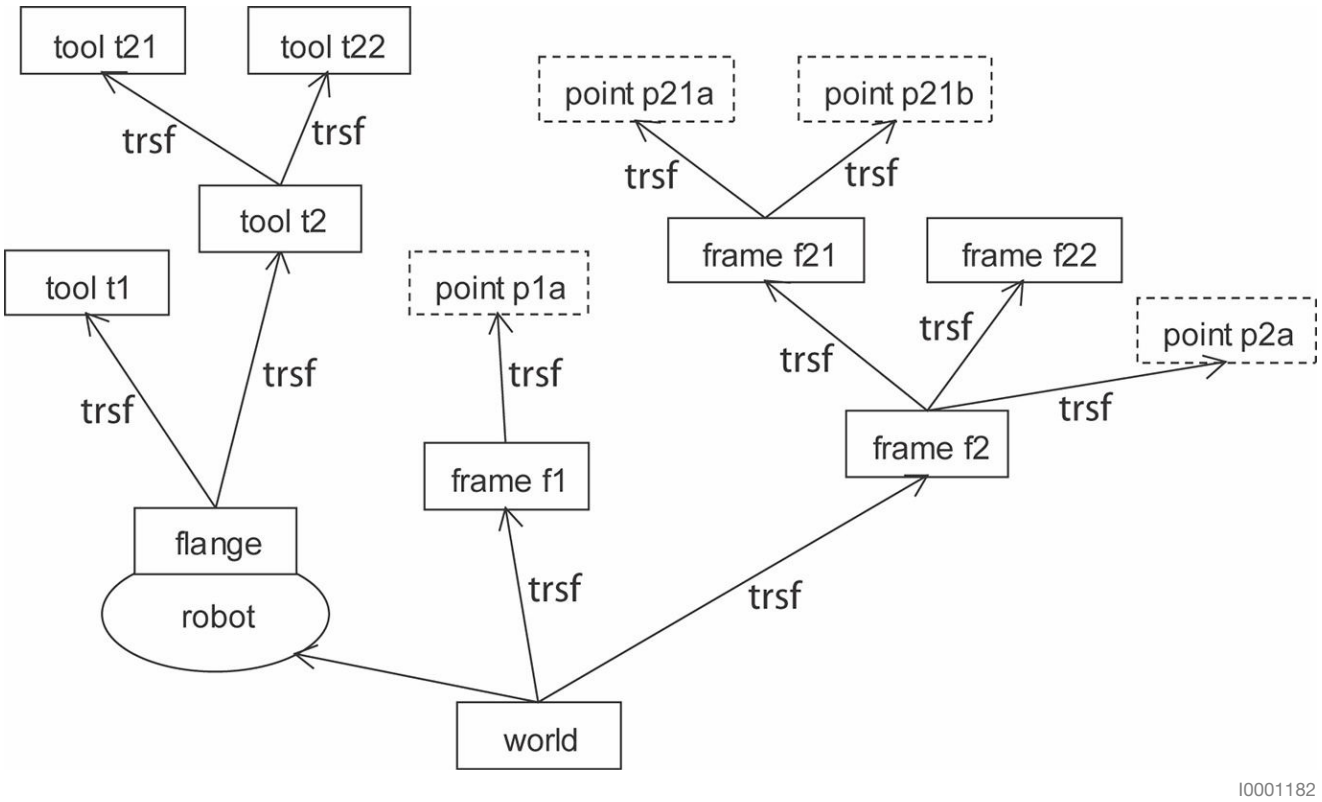


Figure 9.1 : Organigramme : frame / point / tool / trsf

9.2 - TYPE JOINT

M0005650.1

9.2.1 - DÉFINITION

M0005651.1

Une positions articulaire (type joint) définit la position angulaire de chaque angle de rotation et la position linéaire de chaque axe linéaire.

Le type joint est un type structuré dont les champs sont, dans l'ordre :

num j1	position Joint de l'axe 1
num j2	position Joint de l'axe 2
num j3	position Joint de l'axe 3
num j...	position Joint de l'axe ... (un champ par axe)

Ces champs sont exprimés en degrés pour les axes rotatifs, en millimètres ou en pouces pour les axes linéaires. L'origine de chaque axe est définie par le type de bras utilisé.

Par défaut, chaque champ d'une variable de type joint est initialisé à la valeur 0.

9.2.2 - OPÉRATEURS

M0005652.1

Par ordre de priorité croissant :

joint <joint& jPosition1> = <joint jPosition2>	Affecte **jPosition2** à la variable **jPosition1** champ par champ et renvoie **jPosition2**.
bool <joint jPosition1> != <joint jPosition2>	Renvoie **true** si un champ de **jPosition1** n'est pas égal au champ correspondant de **jPosition2** à la précision du robot près, **false** sinon.
bool <joint jPosition1> == <joint jPosition2>	Renvoie **true** si chaque champ de **jPosition1** est égal au champ correspondant de **jPosition2** à la précision du robot près, **false** sinon.
bool <joint jPosition1> > <joint jPosition2>	Renvoie **true** si chaque champ de **jPosition1** est strictement supérieur au champ correspondant de **jPosition2**, **false** sinon.
bool <joint jPosition1> < <joint jPosition2>	Renvoie **true** si chaque champ de **jPosition1** est strictement inférieur au champ correspondant de **jPosition2**, **false** sinon. **DANGER** Attention : **jPosition1**>**jPosition2** n'est pas l'équivalent de !(**jPosition1**<**jPosition2**)
joint <joint jPosition1> - <joint jPosition2>	Renvoie la différence champ par champ de **jPosition1** et **jPosition2**.
joint <joint jPosition1> + <joint jPosition2>	Renvoie la somme champ par champ de **jPosition1** et **jPosition2**.

Afin d'éviter les confusions entre les opérateurs = et ==, l'opérateur = n'est pas autorisé dans les expressions de VAL 3 servant de paramètre d'instructions.

joint **abs**(joint jPosition) M0005654.1

Fonction

Cette instruction renvoie la valeur absolue d'une articulation **jPosition**, champ par champ.

Détails

La valeur absolue d'un joint, avec les opérateurs joint ">" ou "<", permet de calculer facilement la distance séparant la position d'un joint de la position de référence.

Exemples

```
jReference = {90, 45, 45, 0, 30, 0}
jMaxDistance = {5, 5, 5, 5, 5, 5}
j = herej()
// Checks that all the axis are less than 5 degrees from the reference
if(!(abs(j - jReference) < jMaxDistance))
 sOutput="Move closer to the marks"
endIf
```

Voir aussi

Operators

joint **herej**() M0005790.1

Fonction

Cette instruction renvoie la position actuelle de l'articulation du bras.

Lorsque le bras est sous tension, la valeur renvoyée correspond à la position transmise aux amplificateurs par le contrôleur, et non pas à la position relevée sur les encodeurs de l'axe.

Lorsque le bras est hors tension, la valeur renvoyée correspond à la position relevée sur les encodeurs de l'axe ; en raison du bruit dans les mesures des encodeurs, cette mesure peut varier légèrement même lorsque le bras est à l'arrêt.

La position du bras est rafraîchie toutes les 4 ms.

Exemples

```
//Wait until the arm is near the reference position, with a 60 s time out
bStart = watch(abs(herej() - jReference) < jMaxDistance, 60)
if bStart==false
 sOutput="Move closer to the start position")
endIf
```

Voir aussi

here

getLatch

isInRange

joint **min**(joint jJ1, joint jJ2) M0005849.1

Fonction

Cette instruction renvoie la valeur minimum d'une articulation **jJ1** et **jJ2**, champ par champ.

Exemples

```
l_jJoint1 = {10,20,30,40,50,60}
l_jJoint2 = {60,50,40,30,20,10}
min(l_jJoint1, l_jJoint2) returns {10,20,30,30,20,10}
```

Voir aussi

max

joint **max**(joint jJ1, joint jJ2) M0005848.1

Fonction

Cette instruction renvoie la valeur minimum d'une articulation **jJ1** et **jJ2**, champ par champ.

Exemples

```
l_jJoint1 = {10,20,30,40,50,60}
l_jJoint2 = {60,50,40,30,20,10}
max(l_jJoint1, l_jJoint2) returns {60,50,40,40,50,60}
```

Voir aussi

min

bool **isInRange**(joint jPosition) M0005783.1

Fonction

Cette instruction vérifie qu'une position articulaire se trouve dans les limites articulaires logicielles du bras.

Lorsque le bras est hors des limites logicielles de l'articulation (après une opération de maintenance), il ne peut pas être déplacé au moyen d'une application VAL 3 ; seuls les déplacements manuels sont possibles (avec des directions de mouvement qui permettent uniquement de ramener le bras vers ses limites).

Exemples

```
// Check if the current position is within the joint limits
if isInRange(herej())==false
 sOutput="Please place the arm within its workspace"
endIf
```

Voir aussi

herej

void **setLatch**(dio diInput) M0005794.1

Fonction

Cette instruction active la capture de position du robot sur front ascendant du signal d'entrée.

Détails

La capture de la position du robot est une fonctionalité électronique qui n'est supportée que par les entrées rapides du contrôleur (fIn0, fIn1).

La détection du front montant du signal d'entrée n'est garantie que si le signal reste bas pendant au moins 31.25 µs avant le front montant, et haut pendant au moins 31.25 µs après le front montant.

DANGER
La capture n'est activée qu'au bout d'un certain temps (entre 0 et 0.2 ms) après l'exécution de l'instruction setLatch. Vous pouvez ajouter une instruction delay(0) après setLatch pour vous assurer que la capture est effective avant l'exécution de l'instruction VAL 3 suivante.

Une erreur d'exécution 70 (valeur de paramètre invalide) est générée si l'entrée digitale spécifiée ne supporte pas la capture de la position du robot.

Voir aussi

getLatch

enableContinuousLatch

disableContinuousLatch

getContinuousLatch

bool **getLatch**(joint& jPosition)

M0005796.1

Fonction

Cette instruction lit la dernière position capturée du robot.

La fonction renvoie true si une capture de position valide peut être lue. Si une capture est en attente, ou si la capture n'a jamais été activée, la fonction renvoie false et la position n'est pas mise à jour.

getLatch renvoie la même capture de position tant qu'une nouvelle capture n'est pas activée par l'instruction setLatch.

La position du bras est rafraîchie dans le contrôleur toutes les 62.5 µs ; la capture de position correspond à la position du bras entre 0 et 62.5 µs après le front montant de l'entrée rapide.

Exemples

```
setLatch(diLatch)
// Wait for setLatch to be effective before using getLatch
delay(0)
// Wait for a latched position during 5 seconds.
bLatch = watch(getLatch(jPosition)==true, 5)
if bLatch==true
 sOutput="Successful position latch"
else
 sOutput="Nolatch signal was detected"
endIf
```

Voir aussi

setLatch

herej

enableContinuousLatch

disableContinuousLatch

getContinuousLatch

void **enableContinuousLatch**(dio diInput) M0005889.1

Fonction

Après l'appel de cette instruction, les positions de l'articulation du robot correspondant à chaque flanc ascendant du signal d'entrée **diInput** sont stockées dans une mémoire tampon interne.

Détails

La capture de la position du robot est une fonctionalité électronique qui n'est supportée que par les entrées rapides du contrôleur (fIn0, fIn1). Une erreur d'exécution 70 (valeur de paramètre invalide) est générée si l'entrée digitale spécifiée ne supporte pas la capture de la position du robot. Une seule entrée rapide peut être utilisée pour la capture. L'appel de l'instruction enableContinuousLatch() fait passer immédiatement à l'entrée indiquée.

La détection du flanc ascendant du signal d'entrée est garantie seulement :

- si le signal reste bas pendant au moins 31.25 µs avant le flanc ascendant et haut pendant au mois 31.25 µs après le flanc ascendant.
- et si la fréquence du signal est inférieure à 2 kHz pour CPT 9.2 (D243400xx) ou à 4 kHz pour CPT 9.3 (D244009xx).

L'appel de enableContinuousLatch() vide le tampon interne. Pour lire toutes les données du tampon interne, l'instruction getContinuousLatch(...) doit être appelée jusqu'à ce qu'elle renvoie 0.

Voir aussi

getLatch

setLatch

disableContinuousLatch

getContinuousLatch

void **disableContinuousLatch**() M0005890.1

Fonction

Cette instruction désactive le stockage de la position du robot dans un tampon interne activé au préalable avec enableContinuousLatch(dio **diInput**).

Voir aussi

getLatch

setLatch

enableContinuousLatch

getContinuousLatch

num **getContinuousLatch**(joint& jPosition[], num& nLost) M0005891.1

Fonction

Cette instruction remplit la table **jPosition**[] avec les positions du robot capturées et issues du tampon interne et renvoie le nombre de valeurs inscrites dans **jPosition**[].

Les positions perdues depuis le dernier appel de getContinuousLatch() sont ajoutées au paramètre **nLost**.

Voir aussi

getLatch

setLatch

enableContinuousLatch

disableContinuousLatch

getContinuousLatch

num **getContinuousLatch**(joint& jPosition[], num& nLost, num& nTimeStamp[]) M0005892.1

Fonction

Comme getContinuousLatch avec une table d'horodatage correspondant à chaque valeur de **jPosition**.

jPosition[] et **nTimeStamp**[] doivent être de la même taille. Une erreur d'exécution 70 (valeur du paramètre incorrecte) est produite si leur taille n'est pas égale.

Voir aussi

getLatch

setLatch

enableContinuousLatch

disableContinuousLatch

getContinuousLatch

9.3 - TYPE TRSF M0005655.1

9.3.1 - DÉFINITION M0005656.1

Une transformation (type trsf) décrit un changement de position et/ou d'orientation. C'est l'assemblage mathématique d'une translation et d'une rotation.

La transformation ne représente pas elle-même une position dans l'espace, mais elle peut être interprétée comme la position et l'orientation d'un point ou d'un repère cartésien par rapport à un autre repère.

Le type ttrsf est un type structuré dont les champs sont, dans l'ordre :

num x	Translation sur l'axe **x**
num y	Translation sur l'axe **y**
num z	Translation sur l'axe **z**
num rx	Rotation autour de l'axe **x**
num ry	Rotation autour de l'axe **y**
num rz	Rotation autour de l'axe **z**

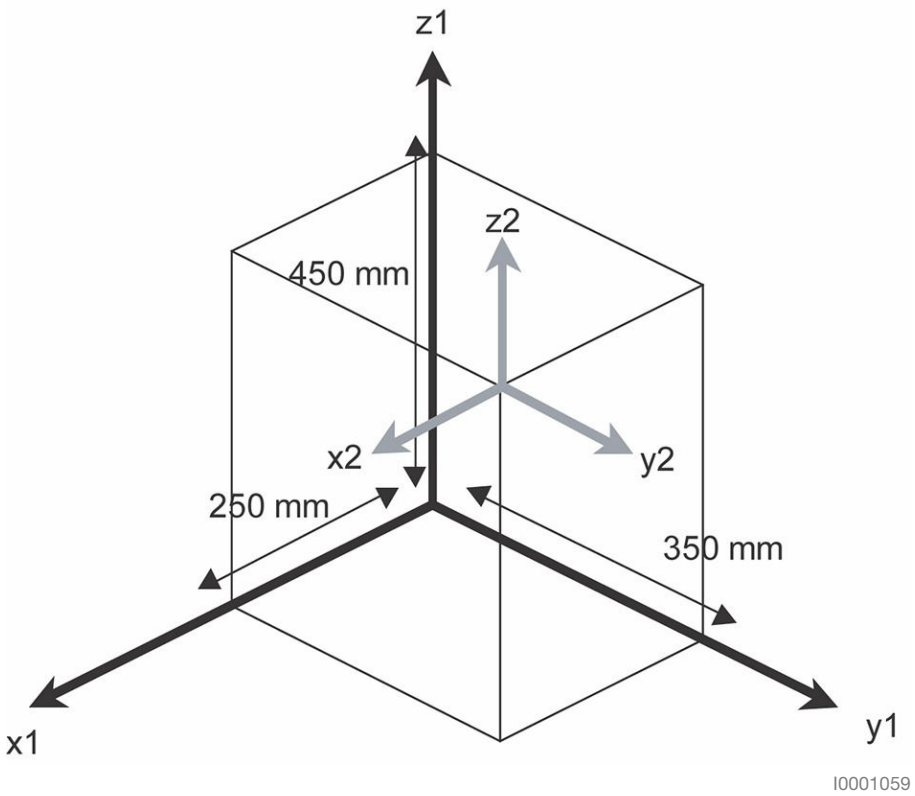
Les champs **x, y** et **z** sont exprimés dans l'unité de longueur de l'application (millimètre ou inch, voir chapitre Unité de longueur). Les champs **rx, ry** et **rz** sont exprimés en degrés.

Les coordonnées **x, y** et **z** sont les coordonnées cartésiennes de la translation (ou la position d'un point ou d'un repère dans le repère de référence). Lorsque **rx, ry** et **rz** sont nuls, la transformation est une translation sans changement d'orientation.

Par défaut, une variable de type trsf est initialisée à la valeur **{0,0,0,0,0,0}**.

9.3.2 - ORIENTATION

M0005657.1



La position du repère **R2**(gris) par rapport à **R1**(noir) est :

x = 250 mm, y = 350 mm, z = 450 mm, rx = 0°, ry = 0°, rz = 0°

Figure 9.2 : Orientation

Les coordonnées **rx, ry** et **rz** correspondent aux angles de rotation qui doivent être appliqués successivement autour des axes **x, y** et **z** pour obtenir l'orientation du repère.

Par exemple, l'orientation **rx = 20°, ry = 10°, rz = 30°** est obtenue de la manière suivante. Le repère **(x,y,z)** est d'abord tourné de **20°** autour de l'axe **x**. On obtient un nouveau repère **(x',y',z')**. Les axes **x** et **x'** sont confondus.

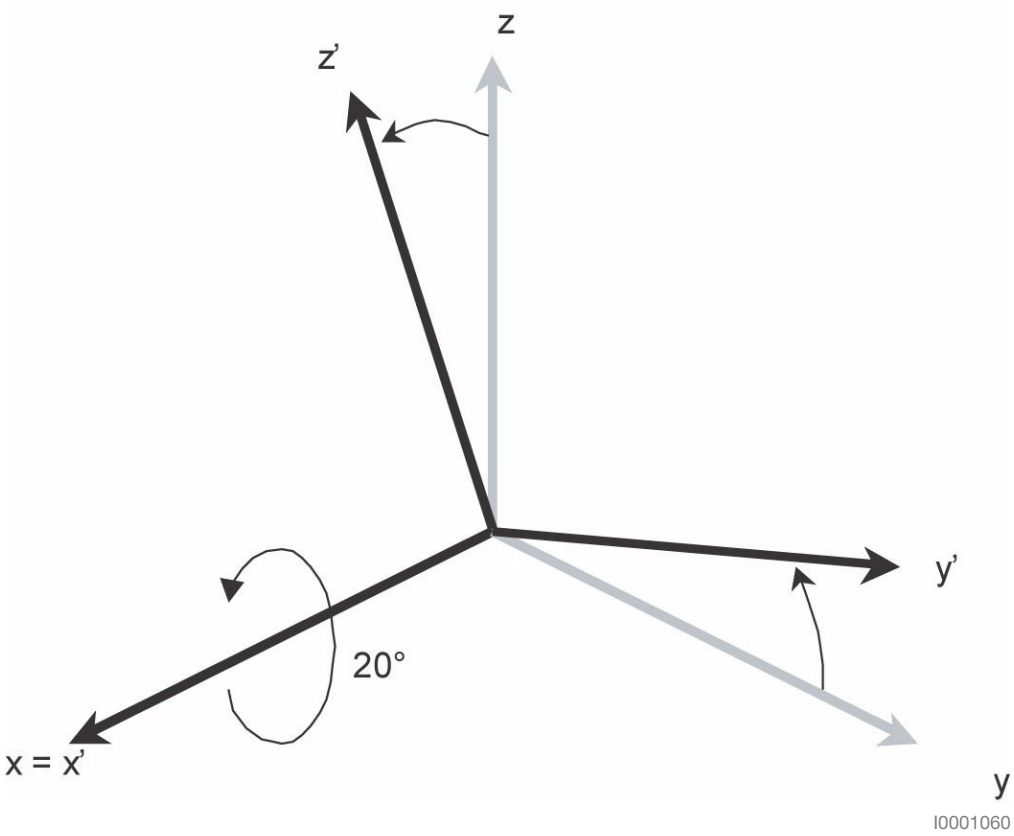


Figure 9.3 : Rotation repère par rapport à l'axe : X

Le repère est ensuite tourné de **20**° autour de l'axe **y'** du repère obtenu dans l'étape précédente. On obtient un nouveau repère **(x'',y'',z'')**. Les axes **y'** et **y''** sont confondus.

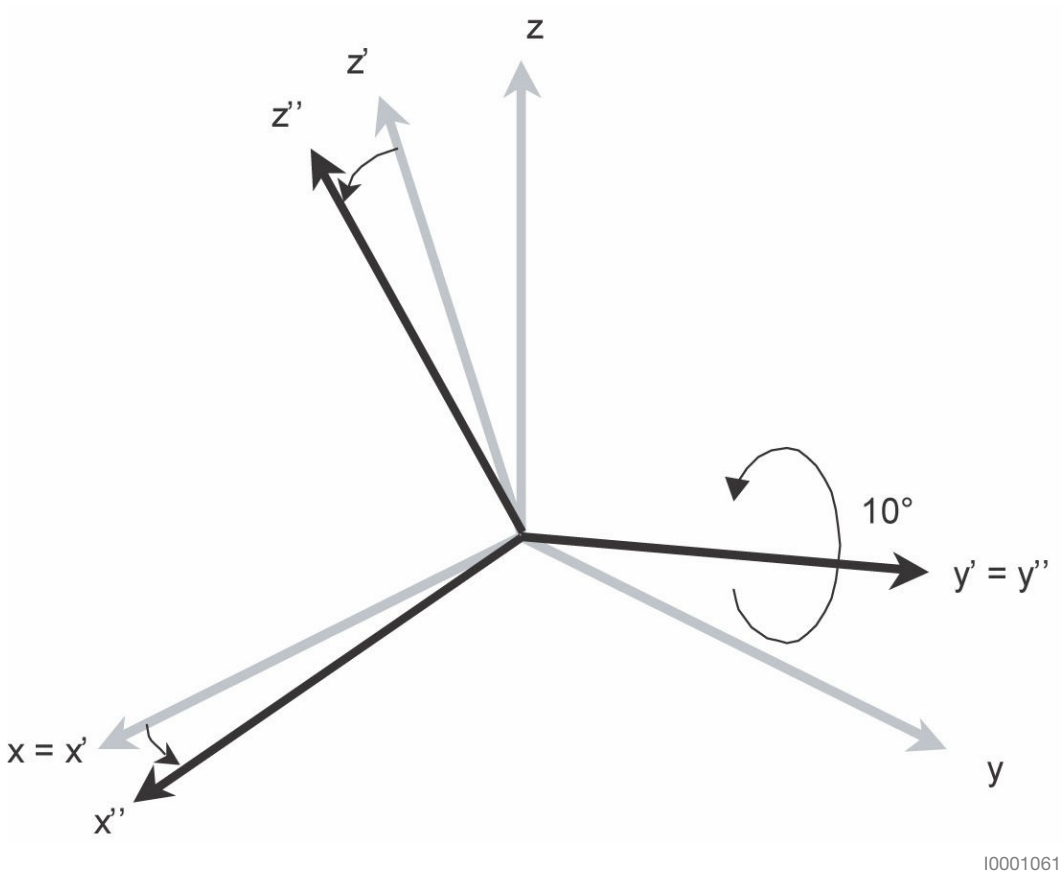
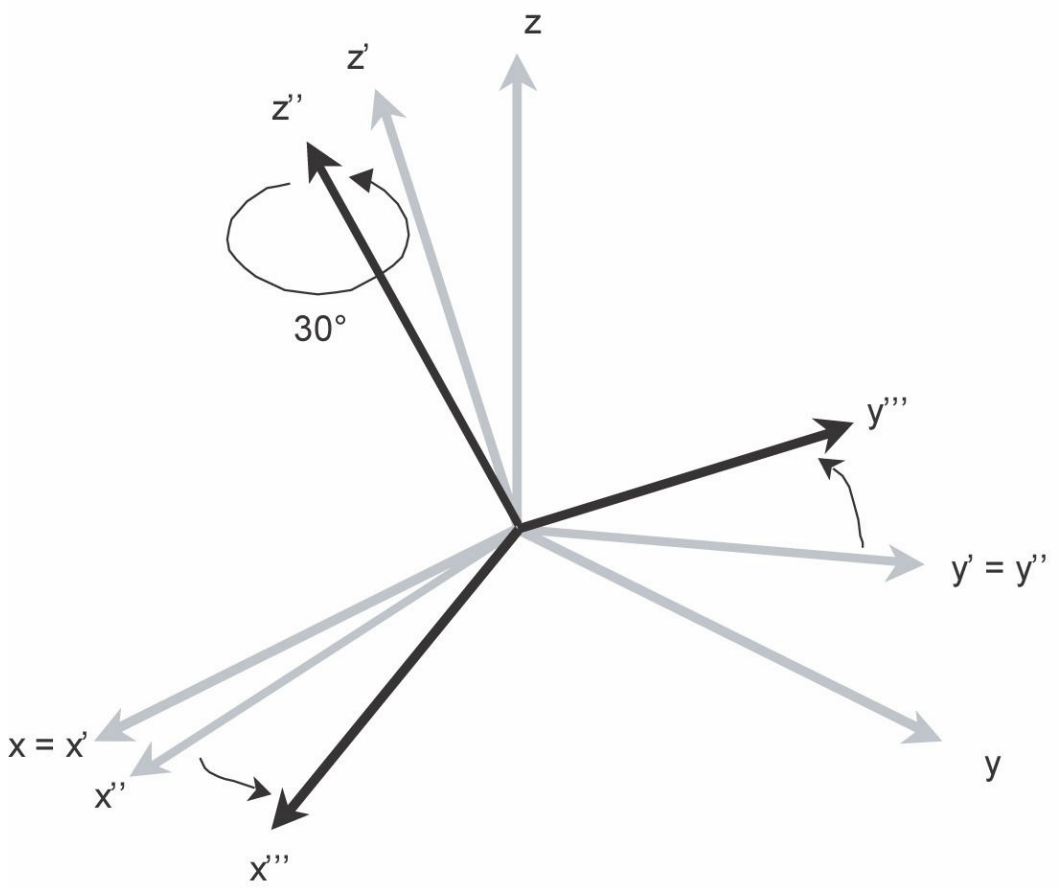


Figure 9.4 : Rotation repère par rapport à l'axe : Y'

Enfin, le repère est tourné de **20°** autour de l'axe **z''** du repère qu'on a obtenu à l'étape précédente. Le nouveau repère **(x''' y''' z''')** obtenu est celui dont l'orientation est définie par **rx, ry, rz**. Les axes **z''** et **z'''** sont confondus.



La position du repère **R2**(gris) par rapport à **R1**(noir) est :

x = 250 mm, y = 350 mm, z = 450 mm, rx = 20°, ry = 10°, rz = 30°

Figure 9.5 : Rotation repère par rapport à l'axe : Z''

Les valeurs de **rx, ry** et **rz** sont définies modulo **360** degrés. Lorsque le système calcule **rx, ry** et **rz**, leurs valeurs sont toujours comprises entre **-180** et **+180** degrés. Une même orientation peut être représentée par différents triplets de valeur. Voir la définition des angles d'Euler (par exemple dans wikipedia). Nous utilisons la convention de Tait-Bryan avec le système XYZ.

9.3.3 - OPÉRATEURS

M0005658.1

Par ordre de priorité croissant :

trsf <trsf& trPosition1> = <trsf trPosition2>	Affecte **trPosition2** à la variable **trPosition1** champ par champ et renvoie **trPosition2**.
bool <trsf trPosition1> != <trsf trPosition2>	Renvoie **true** si un champ de **trPosition1** n'est pas égal au champ correspondant de **trPosition2**, **false** sinon.
bool <trsf trPosition1> == <trsf trPosition2>	Renvoie **true** si chaque champ de **trPosition1** est égal au champ correspondant de **trPosition2**, **false** sinon.
trsf <trsf trPosition1> * <trsf trPosition2>	Renvoie la composition géométrique des transformations **trPosition1** et **trPosition2**. **DANGER** Dans le cas général : **trPosition1** * **trPosition2** != **trPosition2** * **trPosition1**!
trsf ! <trsf trPosition>	Renvoie la transformation inverse de **trPosition**.

Afin d'éviter les confusions entre les opérateurs = et ==, l'opérateur = n'est pas autorisé dans les expressions de VAL 3 servant de paramètre d'instructions.

9.3.4 - INSTRUCTIONS

M0005659.1

num **distance**(trsf trPosition1, trsf trPosition2)

M0005795.1

Fonction

Renvoie la distance entre **trPosition1** et **trPosition2**.

DANGER
Pour que la distance soit valide, il faut que position1 et position2 soient définies par rapport au même repère de référence.

Exemples

Cette ligne calcule la distance entre deux outils :

```
distance(position(tTool1, flange), position(tTool2, flange))
```

Voir aussi

appro

compose

position

distance

trsf **interpolateL**(trsf trStart, trsf trEnd, num nPosition) M0005807.1

Fonction

Cette instruction renvoie une position intermédiaire alignée avec une position de départ **trStart** et une position cible **trEnd**. Le paramètre **nPosition** spécifie l'interpolation linéaire à appliquer selon l'équation pour la coordonnée x : trsf.x0 = trStart.x + (trEnd.x-trStart.x)*nPosition. La même équation est applicable pour les coordonnées Y et Z.

L'orientation rx, ry, rz est calculée selon une équation similaire mais plus complexe. L'algorithme utilisé par interpolateL est le même que celui utilisé par le générateur de mouvement pour calculer les positions intermédiaires lors d'une instruction movel.

interpolateL(trStart, trEnd, 0) renvoie **trStart** ; interpolateL(trStart, trEnd, 1) renvoie **trEnd** ; interpolateL(trStart, trEnd, 0.5) renvoie la position médiane entre **trStart** et **trEnd**. Une valeur négative du paramètre **nPosition** donne une position située "avant" **trStart**. Une valeur supérieure à 1 donne une position située "après" **trEnd**.

Une erreur d'exécution est générée si le paramètre **nPosition** ne se trouve pas dans la plage [-1, 2].

Voir aussi

position(point...

position(frame...

position(tool...

interpolateC

align

trsf **interpolateC**(trsf trStart, trsf trIntermediate, trsf trEnd, num nPosition) M0005812.1

Fonction

Cette instruction renvoie une position intermédiaire sur l'arc d'un cercle défini par les positions **trStart**, **trIntermediate** et **trEnd**. Le paramètre **nPosition** indique l'interpolation circulaire à appliquer. L'algorithme utilisé par **interpolateC** est le même que celui utilisé par le générateur de mouvement pour calculer les positions intermédiaires dans une instruction **movec**.

interpolateC(trStart, trIntermediate, trEnd, 0) renvoie **trStart** ;

interpolateC(trStart, trIntermediate, trEnd, 1) renvoie **trEnd** ;

interpolateC(trStart, trIntermediate, trEnd, 0.5) renvoie la position médiane sur l'arc entre **trStart** et **trEnd**.

Une valeur négative du paramètre **nPosition** donne une position située "avant" **trStart**. Une valeur supérieure à 1 donne une position située "après" **trEnd**.

Une erreur d'exécution est générée si l'arc n'est pas correctement défini (positions trop proches) ou si l'interpolation de rotation reste indéterminée (voir le chapitre "Contrôle des mouvements - Interpolation de l'orientation").

Voir aussi

position(point...

position(frame...

position(tool...

interpolateL

align

Fonction

Cette instruction renvoie l'entrée **trPosition** avec une orientation modifiée de telle sorte que l'axe Z de l'orientation renvoyée est aligné sur l'axe X, Y ou Z le plus proche de l'orientation de référence de **trReference**. Les coordonnées X, Y, Z de **trPosition** et **trReference** ne sont pas utilisées : les coordonnées x, y, z de la valeur renvoyée sont les mêmes que les coordonnées x, y, z de **trPosition**.

Voir aussi

position(point...

position(frame...

position(tool...

interpolateL

interpolateC

9.4 - TYPE FRAME M0005660.1

9.4.1 - DÉFINITION M0005661.1

Le type frame permet de définir la position de repères de référence dans la cellule.

Le type frame est un type structuré avec un seul champ accessible :

trsf trsf	position du repère dans son repère de référence

Le repère de référence d'une variable de type **frame** est défini quand il est initialisé (à l'aide de l'interface utilisateur, de l'opérateur = ou de l'instruction link()). Le repère de référence **world** est toujours défini dans une application VAL 3 : tout repère de référence est, directement ou via d'autres repères, lié au repère **world**.

Une erreur d'exécution est générée pendant un calcul géométrique si les coordonnées du repère **world** ont été modifiées.

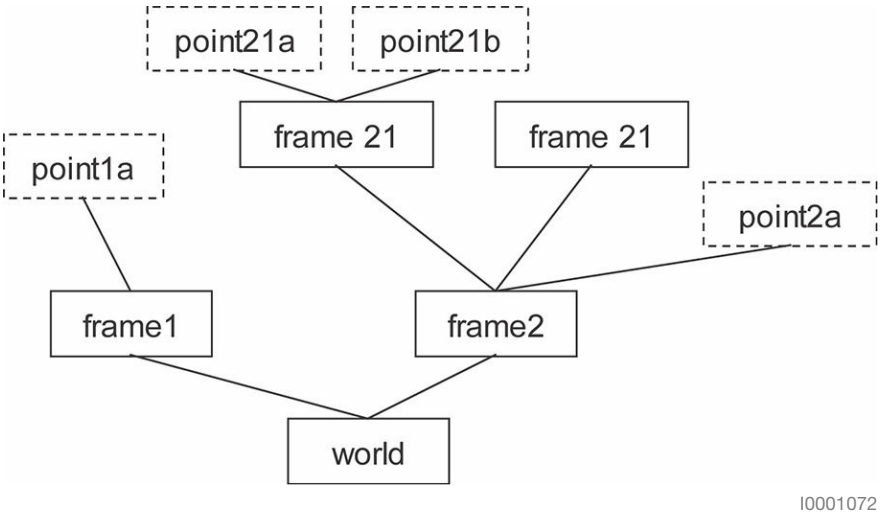


Figure 9.6 : Liens entre repères de référence

Par défaut, les variables de repère local et les repères dans les variables de type utilisateur n'ont pas de repère de référence. Avant de pouvoir être utilisés, ils doivent être initialisés à partir d'un autre repère à l'aide de l'opérateur '=' ou de l'une des instructions link() et setFrame().

9.4.2 - UTILISATION

M0005662.1

L'utilisation de repères de référence dans une application robotique est vivement recommandée :

- **Pour donner une vue plus intuitive des points de l'application**

  L'affichage des points de la cellule est structuré selon l'organisation hiérarchique des repères.

- **Pour mettre à jour rapidement la position d'un ensemble de points**

  Dès qu'un point de l'application est lié à un objet, il est souhaitable de définir un repère pour cet objet, et de lier les points VAL 3 à ce repère. Si l'objet est déplacé, il suffit de réapprendre le repère pour que tous les points qui y sont liés soient corrigés du même coup.

- **Pour reproduire une trajectoire à plusieurs endroits de la cellule**

  On peut pour cela définir les points de la trajectoire par rapport à un repère de travail, et apprendre un repère pour chaque endroit où la trajectoire doit être reproduite. En affectant la valeur d'un repère appris au repère de travail, la trajectoire entière se "déplace" sur le repère appris.

- **Pour calculer facilement des déplacements géométriques**

  L'instruction compose() permet d'effectuer sur tout point des déplacements géométriques exprimés dans un repère de référence quelconque. L'instruction position() permet de calculer la position d'un point dans un repère de référence quelconque.

9.4.3 - OPÉRATEURS

M0005663.1

Par ordre de priorité croissant :

frame <frame& fReference1> = <frame fReference2>	Affecte la position et le repère de référence de **fReference2** à la variable **fReference1**.
bool <frame fReference1> != <frame fReference2>	Renvoie **true** si **fReference1** et **fReference2** n'ont pas le même repère de référence ou n'ont pas la même position dans leur repère de référence.
bool <frame fReference1> == <frame fReference2>	Renvoie **true** si **fReference1** et **fReference2** ont la même position dans le même repère de référence.

Afin d'éviter les confusions entre les opérateurs = et ==, l'opérateur = n'est pas autorisé dans les expressions de VAL 3 servant de paramètre d'instructions.

9.4.4 - INSTRUCTIONS

M0005664.1

num **setFrame**(point pOrigin, point pAxisOx, point pPlaneOxy, frame& fResult) M0005665.1

Fonction

Cette instruction calcule les coordonnées de **fResult** à partir de son origine **pOrigin**, d'un point **pAxisOx** sur l'axe **(Ox)** et d'un point **pPlaneOxy** sur l'axe **(Oxy)**.

Le point **pAxisOx** doit être du côté des **x** positifs. Le point **pPlaneOxy** doit être du côté des **y** positifs.

La fonction renvoie :

0	Pas d'erreur.
-1	Le point **pAxisOx** est trop proche de **pOrigin**.
-2	Le point **pPlaneOxy** est trop proche de l'axe **(Ox)**.

Une erreur d'exécution est générée si l'un des points n'a pas de repère de référence.

trsf **position**(frame fFrame, frame fReference) M0005805.1

Fonction

Cette instruction renvoie les coordonnées du repère **fFrame** dans le repère de référence **fReference**.

Une erreur d'exécution est générée si **fFrame** ou **fReference** n'a pas de repère de référence.

Voir aussi

position(point...

position(tool...

void **link**(frame fFrame, frame fReference) M0005666.1

Fonction

Cette instruction change le repère de référence de **fFrame** et le règle à **fReference**. La position du repère dans le repère de référence reste inchangée.

Voir aussi

Operators

9.5 - TYPE TOOL

M0005667.1

9.5.1 - DÉFINITION

M0005668.1

Le type tool permet de définir la géométrie et l'action d'un outil.

Le type tool est un type structuré avec comme champs, dans l'ordre :

trsf trsf	position du point du centre outil (TCP) dans son outil de base
dio gripper	sortie tout ou rien servant à actionner l'outil
num otime	Délai d'ouverture de l'outil (secondes)
num ctime	Délai de fermeture de l'outil (secondes)

L'outil de référence d'une variable de type tool est défini quand il est initialisé (à l'aide de l'interface utilisateur, de l'opérateur = ou de l'instruction link()). L'outil flange est toujours défini dans une application VAL 3 : tout outil est, directement ou via d'autres outils, lié à l'outil flange.

Une erreur d'exécution est générée pendant un calcul géométrique si les coordonnées de l'outil flange ont été modifiées.

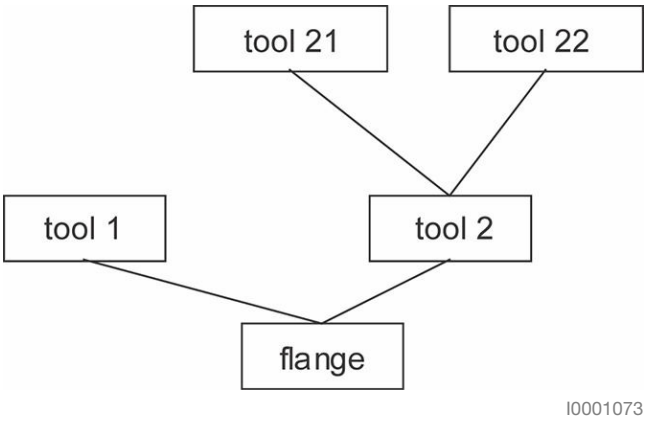


Figure 9.7 : Liens entre outils

Par défaut, la sortie d'un outil est la première sortie de vanne, les temps d'ouverture et de fermeture sont à **0** et l'outil de base est flange. Les variables d'outil local et les outils dans les variables de type utilisateur n'ont pas d'outil de référence. Avant de pouvoir être utilisés, ils doivent être initialisés à partir d'un autre outil à l'aide de l'opérateur '=' ou de l'instruction link().

9.5.2 - UTILISATION

M0005669.1

L'utilisation d'outils dans une application robotique est vivement recommandée :

- **Pour contrôler la vitesse de déplacement**

  Lors de déplacement manuels ou programmés, le système contrôle la vitesse cartésienne en bout d'outil.

- **Pour aller aux mêmes points avec différents outils**

  Il suffit de sélectionner l'outil VAL 3 correspondant à l'outil physique en bout de bras.

- **Pour contrôler l'usure de l'outil ou un changement d'outil**

  L'usure de l'outil peut simplement être compensée par l'actualisation des coordonnées géométriques de celui-ci.

9.5.3 - OPÉRATEURS

M0005670.1

Par ordre de priorité croissant :

tool <tool& tTool1> = <tool tTool2>	Affecte la position et l'outil de base de **tTool2** à la variable **tTool1**.
bool <tool tTool1> != <tool tTool2>	Renvoie **true** si **tTool1** et **tTool2** n'ont pas le même outil de base, la même position dans leur outil de base, la même sortie digitale, ou les mêmes temps d'ouverture et fermeture.
bool <tool tTool1> == <tool tTool2>	Renvoie **true** si **tTool1** et **tTool2** ont la même position dans le même outil de base, utilisent la même sortie digitale avec les mêmes temps d'ouverture et fermeture.

Afin d'éviter les confusions entre les opérateurs = et ==, l'opérateur = n'est pas autorisé dans les expressions de VAL 3 servant de paramètre d'instructions.

9.5.4 - INSTRUCTIONS

M0005671.1

void **open**(tool tTool)

M0005854.1

Fonction

Cette instruction active l'outil (ouverture) en réglant sa sortie numérique à **true**.

Avant d'actionner l'outil, open() attend que le robot soit au point en faisant l'équivalent d'un waitEndMove(). Après l'activation, le système attend **otime** secondes avant d'exécuter l'instruction suivante.

Cette instruction n'assure pas que le robot soit stabilisé à sa position finale avant l'activation de l'outil. Lorsqu'il est nécessaire d'attendre la stabilisation complète du mouvement, il faut utiliser l'instruction isSettled().

Une erreur d'exécution est générée si la dio de **tTool** n'est pas définie ou n'est pas une sortie, ou si une commande de mouvement précédemment enregistrée ne peut être exécutée.

Exemples

```
// the open() instruction is equivalent to:
waitEndMove()
tTool.gripper=true
delay(tTool.otime)
```

Voir aussi

close

waitEndMove

void **close**(tool tTool) M0005853.1

Fonction

Cette instruction active l'outil (fermeture) en réglant sa sortie numérique à **false**.

Avant d'actionner l'outil, close() attend que le robot soit arrêté au point en faisant l'équivalent d'un waitEndMove(). Après l'activation, le système attend ctime secondes avant d'exécuter l'instruction suivante.

Cette instruction n'assure pas que le robot soit stabilisé à sa position finale avant l'activation de l'outil. Lorsqu'il est nécessaire d'attendre la stabilisation complète du mouvement, il faut utiliser l'instruction isSettled().

Une erreur d'exécution est générée si la dio de **tTool** n'est pas définie ou n'est pas une sortie, ou si une commande de mouvement précédemment enregistrée ne peut être exécutée.

Exemples

```
// the close instruction is equivalent to:
waitEndMove()
tTool.gripper = false
delay(tTool.ctime)
```

Voir aussi

open

waitEndMove

trsf **position**(tool tTool, tool tReference) M0005824.1

Fonction

Cette instruction renvoie les coordonnées de l'outil **tTool** dans l'outil **tReference**.

Une erreur d'exécution est générée si **tTool** ou **tReference** n'a pas d'outil de référence.

Voir aussi

position(point...

position(frame...

void **link**(tool tTool, tool tReference) M0005672.1

Fonction

Cette instruction change l'outil de référence de **tTool** et le règle à **tReference**. La position de l'outil dans l'outil de référence reste inchangée.

Voir aussi

Operators

9.6 - TYPE POINT

M0005673.1

9.6.1 - DÉFINITION

M0005674.1

Le type point permet de définir la position et l'orientation de l'outil du robot dans la cellule.

Le type point est un type structuré avec comme champs, dans l'ordre :

trsf trTrsf	position du point dans son repère de référence
config config	configuration du bras pour atteindre la position

Le repère de référence d'un point est une variable de type frame, définie lors de son initialisation (depuis l'interface utilisateur, par l'opérateur = et les instructions link(), here(), appro() et compose()).

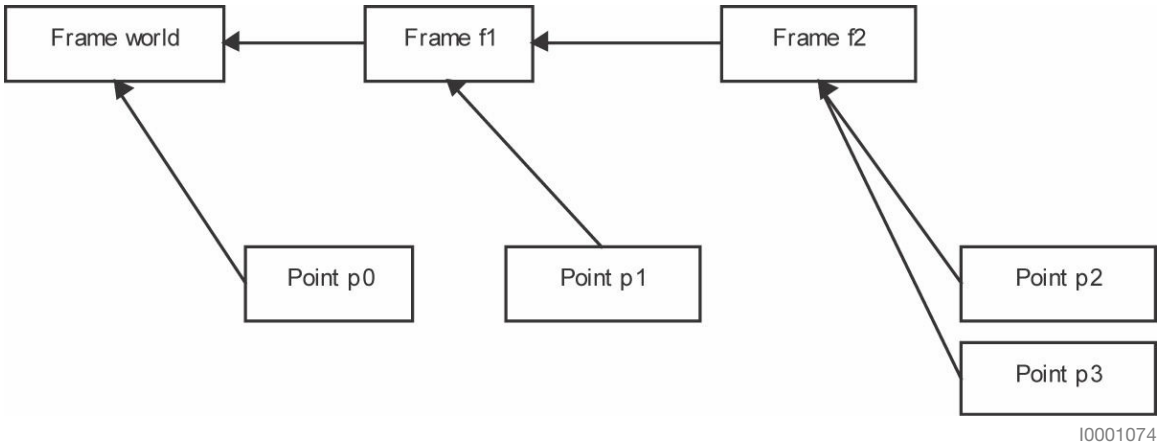


Figure 9.8 : Définition point

Une erreur d'exécution est générée si une variable de type point sans repère de référence est utilisée.

DANGER

Par défaut, les variables de point local et les points dans les variables de type utilisateur n'ont pas de repère de référence. Avant de pouvoir être utilisés, ils doivent être initialisés à partir d'un autre point à l'aide de l'opérateur '=' ou de l'une des instructions link(), here(), appro() et compose().

9.6.2 - OPÉRATEURS

M0005675.1

Par ordre de priorité croissant :

point <point& pPoint1> = <point pPoint2>	Affecte la position, la configuration et le repère de référence de **pPoint2** à la variable **pPoint1**.
bool <point pPoint1> ! = <point pPoint2>	Renvoie **true** si **pPoint1** et **pPoint2** n'ont pas le même repère de référence ou n'ont pas la même position dans leur repère de référence.
bool <point pPoint1> == <point pPoint2>	Renvoie **true** si **pPoint1** et **pPoint2** ont la même position dans le même repère de référence.

Afin d'éviter les confusions entre les opérateurs = et ==, l'opérateur = n'est pas autorisé dans les expressions de VAL 3 servant de paramètre d'instructions.

num **distance**(point pPosition1, point pPosition2) M0005830.1

Fonction

Cette instruction renvoie la distance entre **pPosition1** et **pPosition2**.

Une erreur d'exécution est générée si **pPosition1** ou **pPosition2** n'a pas de repère de référence défini.

Exemples

Ce programme attend que le bras se rapproche à moins de 10 mm de la position **pTarget**

```
wait(distance(here(tTool,world),pTarget)< 10)
```

Voir aussi

appro

compose

position

distance

point **compose**(point pPosition, frame fReference, trsf trTransformation) M0005831.1

Fonction

Cette instruction renvoie le **pPosition** auquel la transformation géométrique **trTransformation** est appliquée par rapport au repère **fReference**.

DANGER
La composante rotation de **tTransformation** modifie en général non seulement l'orientation de **pPosition**, mais aussi ses coordonnées cartésiennes (sauf si **pPosition** se situe sur l'origine de **fReference**).
Si l'on souhaite que **tTransformation** ne modifie que l'orientation de **pPosition**, il faut mettre à jour le résultat avec les coordonnées cartésiennes de **pPosition** (voir exemple).

Le repère de référence et la configuration du point renvoyé sont ceux de **pPosition**.

Une erreur d'exécution est générée si aucun repère de référence n'est défini pour **pPosition**.

Exemples

```
// modification of the orientation without modification of Position
pResult = compose(pPosition,fReference,trTransformation)
pResult.trsf.x = pPosition.trsf.x
pResult.trsf.y = pPosition.trsf.y
pResult.trsf.z = pPosition.trsf.z
// modification of Position without modification of the orientation
trTransformation.rx = trTransformation.ry =trTransformation.rz = 0
pResult = compose(pResult,fReference,trTransformation)
```

Voir aussi

Operators

appro

point **appro**(point pPosition, trsf trTransformation) M0005832.1

Fonction

Cette instruction renvoie un point modifié par transformation géométrique. La transformation est définie par rapport aux axes du centre outil d'entrée.

Le repère de référence et la configuration du point renvoyé sont ceux du point d'entrée.

Une erreur d'exécution est générée si aucun repère de référence n'est défini pour pPosition.

Exemples

```
// Approach: move to 100 mm above the point (Z axis)
movej(appro(pDestination,{0,0,-100,0,0,0}), flange, mNomDesc)
// Go to point
movel(pDestination, flange, mNomDesc)
```

Voir aussi

Operators

compose

point **here**(tool tTool, frame fReference) M0005784.1

Fonction

Cette instruction renvoie la position actuelle de l'outil **tTool** dans le repère **fReference** (position commandée et non position mesurée). Le repère de référence du point renvoyé est **fReference**. La configuration du point renvoyé est la configuration courante du bras.

Voir aussi

herej

config

jointToPoint

point **jointToPoint**(tool tTool, frame fReference, joint jPosition) M0005785.1

Fonction

Cette instruction renvoie la position de **tTool** dans le repère **fReference** quand le bras est dans la position articulaire **jPosition**.

Le repère de référence du point renvoyé est **fReference**. La configuration du point renvoyé est la configuration du bras dans la position articulaire **jPosition**.

Voir aussi

here

pointToJoint

bool **pointToJoint**(tool tTool, joint jInitial, point pPosition, joint& jResult) M0005786.1

Fonction

Cette instruction calcule la position articulaire **jResult** correspondant au point **pPosition** spécifié. Elle renvoie **true** si **jResult** est actualisé ou **false** si aucune solution n'a été trouvée.

La position articulaire à calculer correspond à la localisation du **pPosition**. Les champs à la valeur free n'imposent pas la configuration. Les champs à la valeur **same** imposent la même configuration que **jInitial**.

Pour un axe capable d'effectuer plus d'un tour complet, il existe plusieurs solutions articulaires ayant exactement la même configuration : la solution la plus proche de **jInitial** est alors prise.

Il peut ne pas y avoir de solution si **pPosition** est hors d'atteinte (bras trop court) ou hors des butées logicielles. Si **pPosition** spécifie une configuration, il peut être hors des butées pour cette configuration, mais dans les butées pour une autre configuration.

Une erreur d'exécution est générée si aucun repère de référence n'est défini pour **pPosition**.

Voir aussi

herej

jointToPoint

trsf **position**(point pPosition, frame fReference) M0005787.1

Fonction

Cette instruction renvoie les coordonnées de **pPosition** dans le repère **fReference**.

Une erreur d'exécution est générée si **pPosition** n'a pas de repère de référence.

Exemples

La distance entre 2 points est la distance entre leurs positions dans world :

```
distance(position(pPoint1, world), position(pPoint2, world)) is distance(pPoint1, pPoint2)
```

Voir aussi

distance

position(tool...

position(frame...

void **link**(point pPoint, frame fReference) M0005677.1

Fonction

Cette instruction change le repère de référence de **pPoint** et le règle à **fReference**. La position du point dans le repère de référence reste inchangée.

Voir aussi

Operators

9.7 - TYPE CONFIG M0005678.1

La notion de configuration d'un point cartésien est une notion "avancée" qui peut être omise en première lecture.

9.7.1 - INTRODUCTION

M0005679.1

En général, le robot a plusieurs possibilités pour atteindre une position cartésienne donnée.

Ces différentes possibilités sont appelées « configurations ». Deux configurations différentes sont représentées sur la figure suivante :

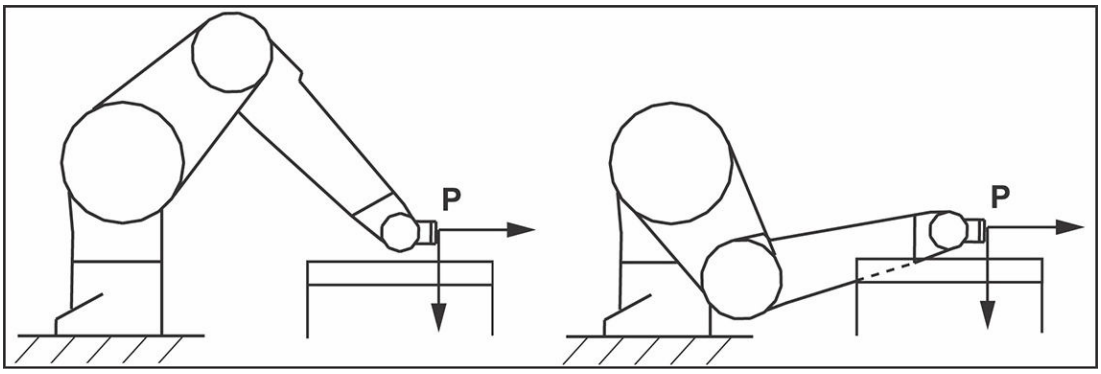


Figure 9.9 : Deux configurations possibles pour atteindre le même point : P

Dans certains cas, il est important de spécifier, parmi toutes les configurations possibles, celles qui sont valides et celles que l'on veut interdire. Pour résoudre ce problème, le type point permet de spécifier les configurations admises pour le robot grâce à son champ de type config défini ci-dessous.

9.7.2 - DÉFINITION

M0005680.1

Le type config permet de définir les configurations autorisées pour une positions cartésienne donnée.

Il dépend du type de bras utilisé.

Pour un bras Stäubli TX2 le type config est un type structuré dont les champs sont, dans l'ordre :

shoulder	configuration de l'épaule
elbow	configuration du coude
wrist	configuration du poignet

Pour un bras Stäubli TS2/TP, le type config se limite au champ **Shoulder** :

shoulder	configuration de l'épaule

Les champs **shoulder**, **elbow** et **wrist** peuvent prendre les valeurs suivantes :

shoulder	**righty**	Configuration de l'épaule **righty** imposée
	lefty	Configuration de l'épaule **lefty** imposée
	ssame	Changement de configuration de l'épaule interdit
	sfree	Configuration de l'épaule libre

elbow	**epositive**	Configuration du coude **epositive** imposée
	enegative	Configuration du coude **enegative** imposée
	esame	Changement de configuration du coude interdit
	efree	Configuration du coude libre

wrist	**wpositive**	Configuration du poignet **wpositive** imposée
	wnegative	Configuration du poignet **wnegative** imposée
	wsame	Changement de configuration du poignet interdit
	wfree	Configuration du poignet libre

9.7.3 - OPÉRATEURS

M0005681.1

Par ordre de priorité croissant :

config <config& configuration1> = <config configuration2>	Affecte les champs **shoulder**, **elbow** et **wrist** de **configuration2** à la variable **configuration1**.
bool <config configuration1> != <config configuration2>	Renvoie true si c**onfiguration1** et **configuration2** n'ont pas la même valeur de champ **shoulder**, **elbow** ou **wrist**.
bool <config configuration1> == <config configuration2>	Renvoie true si **configuration1** et **configuration2** ont les mêmes valeurs de champs **shoulder**, **elbow** et **wrist**.

Afin d'éviter les confusions entre les opérateurs = et ==, l'opérateur = n'est pas autorisé dans les expressions de VAL 3 servant de paramètre d'instructions.

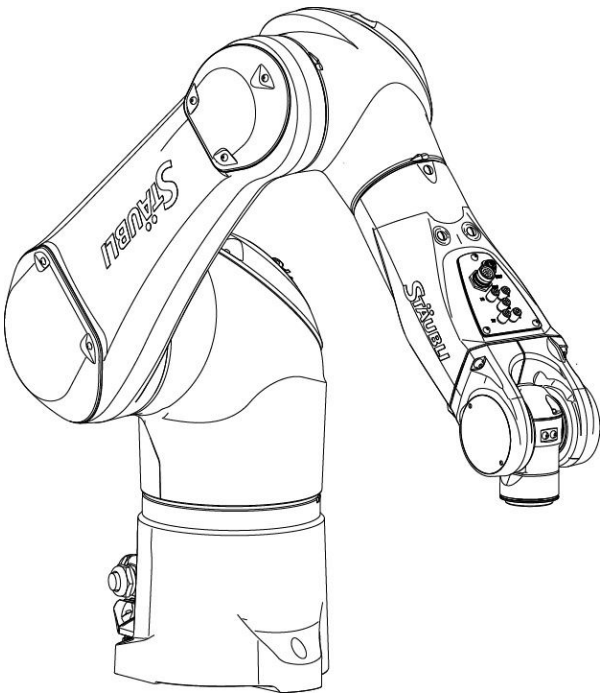
9.7.4 - CONFIGURATION (BRAS TX2)

M0005682.1

9.7.4.1 - Configuration de l'épaule

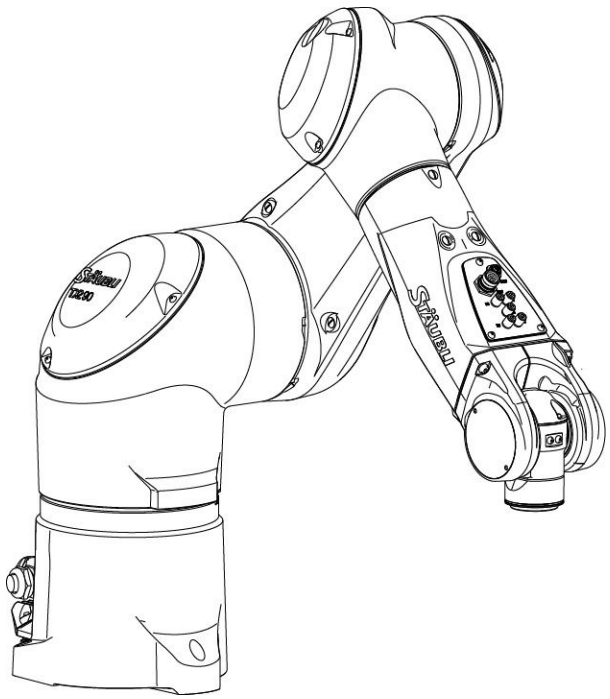
M0005683.1

Pour atteindre un point cartésien, le bras du robot peut être à droite ou à gauche du point : ces deux configurations sont appelées **righty** et **lefty**.



I0003606

Figure 9.10 : Configuration : righty



I0003607

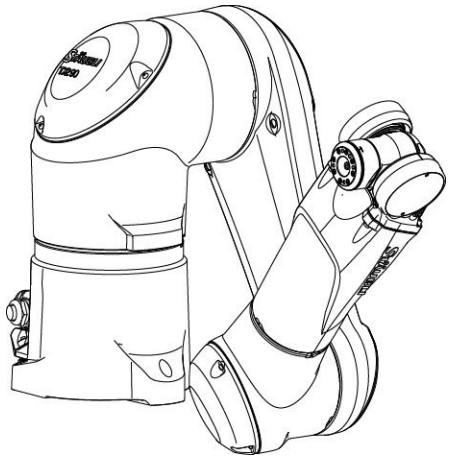
Figure 9.11 : Configuration : lefty

La configuration righty est définie par (d1 * sin(j2) + d2 * sin(j2+j3) + δ) < 0, la configuration lefty est définie par (d1 * sin(j2) + d2 * sin(j2+j3) + δ) >= 0, où d1 est la longueur du bras du robot, d2 est la longueur de l'avant-bras, et δ est la distance entre l'axe 1 et l'axe 2, dans la direction x.

9.7.4.2 - Configuration du coude

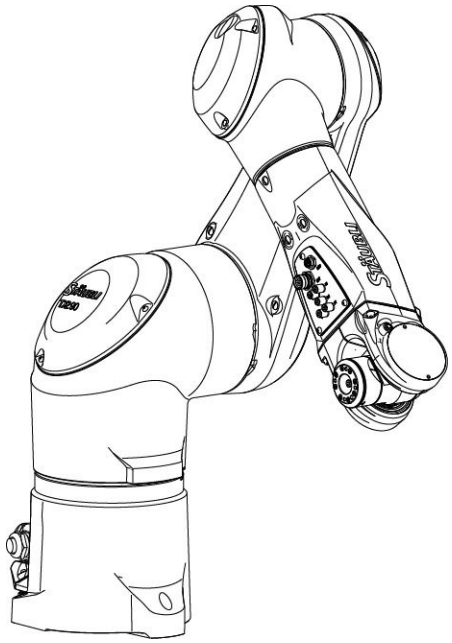
M0005684.1

En plus de la configuration de l'épaule, il y a deux possibilités pour le coude du robot : les configurations du coude sont appelées **epositive** et **enegative**.



I0003608

Figure 9.12 : Configuration : enegative



I0003609

Figure 9.13 : Configuration : epositive

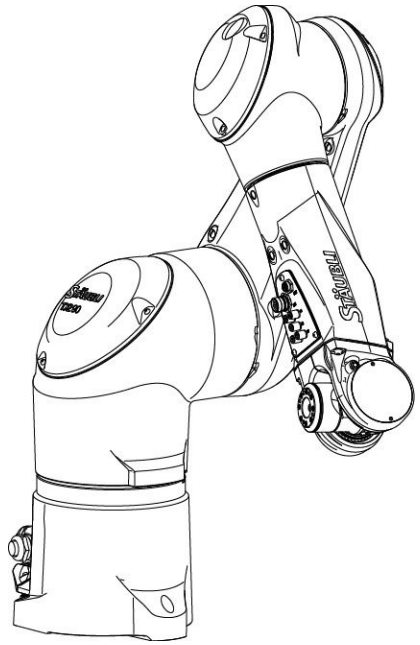
La configuration **epositive** est définie par **j3 >= 0**.

La configuration **enegative** est définie par **j3 < 0**.

9.7.4.3 - Configuration du poignet

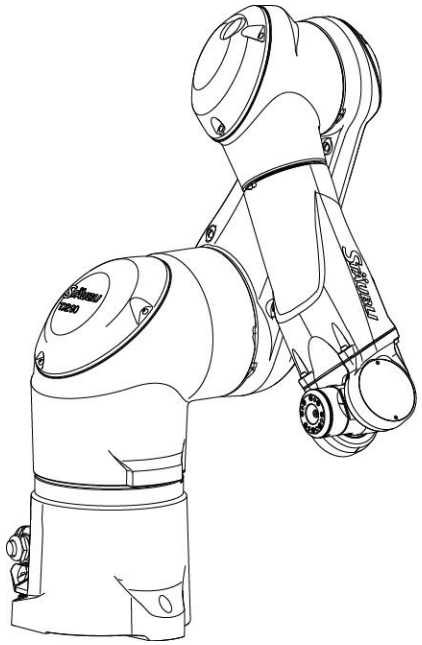
M0005685.1

En plus de la configuration de l'épaule et du coude, il y a deux possibilités pour positionner le poignet du robot. Les deux configurations du poignet sont appelées **wpositive** et **wnegative**.



I0003610

Figure 9.14 : Configuration : wnegative



I0003611

Figure 9.15 : Configuration : wpositive

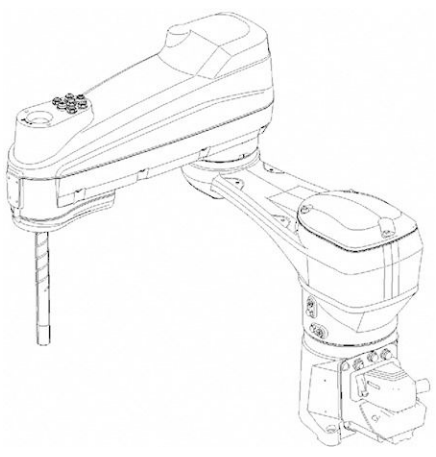
La configuration **wpositive** est définie par **j5 >= 0**.

La configuration **wnegative** est définie par **j5 < 0**.

9.7.5 - CONFIGURATION SCARA (BRAS TS2/TP)

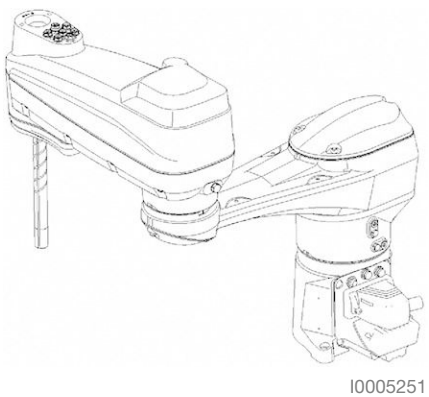
M0005686.1

Pour atteindre un point cartésien, le bras du robot peut être à droite ou à gauche du point : ces deux configurations sont appelées **righty** et **lefty**.



I0005250

Figure 9.16 : Configuration : righty



I0005251

Figure 9.17 : Configuration : lefty

La configuration **righty** est définie par **sin(j2) > 0**, la configuration **lefty** est définie par **sin(j2) < 0**.

config **config**(joint jPosition) M0005788.1

Fonction

Cette instruction renvoie la configuration du robot pour la position de l'articulation **jPosition**.

Voir aussi

here

herej

10 - CONTRÔLE DES MOUVEMENTS

M0005688.1

10.1 - CONTRÔLE DE TRAJECTOIRE

M0005689.1

Il ne suffit pas de donner une succession de points pour définir la trajectoire d'un robot. Il faut aussi indiquer le type de trajectoire utilisée entre les points (courbe ou ligne droite), spécifier comment ces trajectoires se raccordent entre elles, et enfin définir les paramètres se rapportant à la vitesse du mouvement. Ce paragraphe présente donc les différents types de mouvements (instructions **movej**, **movel** et **movec**) et explique comment utiliser les paramètres du descripteur de mouvement (type **mdesc**).

10.1.1 - TYPES DE MOUVEMENT : POINT-À-POINT, LIGNE DROITE, CERCLE

M0005690.1

Les mouvements du robot se programment essentiellement à l'aide des instructions **movej, movel** et **movec**. L'instruction **movej** permet de faire des mouvement point-à-point, **movel** s'utilise pour les mouvements en ligne droite et **movec** sur les mouvements circulaires.

Un mouvement de point à point est un mouvement dans lequel seule la destination finale (coordonnées cartésiennes ou position de l'articulation) est importante. Entre le point de départ et le point d'arrivée, le centre outil suit une courbe définie par le système de manière à optimiser la vitesse du mouvement.

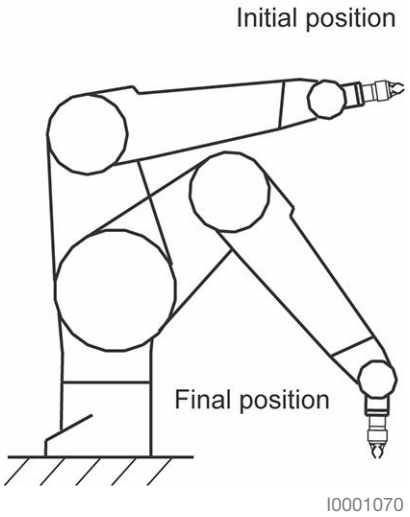


Figure 10.1 : Position initiale et finale

Au contraire, dans un mouvement en ligne droite, le centre outil se déplace le long d'une ligne droite. L'orientation est interpolée linéairement entre l'orientation de départ et l'orientation finale de l'outil.

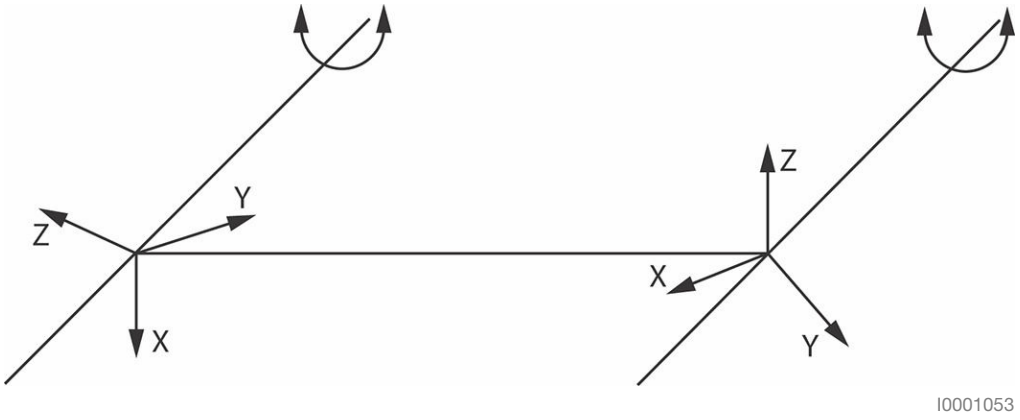
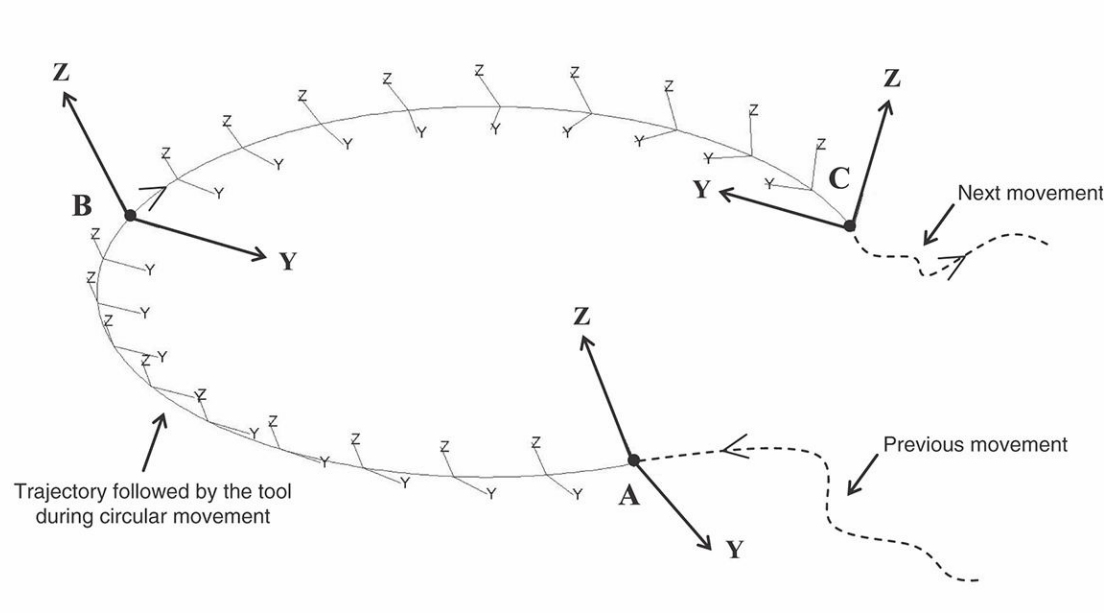


Figure 10.2 : Mouvement en ligne droite

Dans un mouvement circulaire, le centre outil se déplace le long d'un arc de cercle défini par **3** points, et l'orientation outil est interpolée entre l'orientation de départ, l'orientation intermédiaire et l'orientation finale.



Anglais	Traduction
Trajectory followed by the tool during circular movement	Trajectoire décrite par l'outil pendant le mouvement circulaire
Next movement	Mouvement suivant
Previous movement	Mouvement précédent

Figure 10.3 : Mouvement circulaire

Exemples :

Une tâche de manipulation typique consiste à prendre des pièces à un endroit donné, et les déposer à un autre endroit. On suppose qu'on doit prendre les pièces au point **pPick**, et les déposer au point **pPlace**. Pour aller du point **pPick** au point **pPlace**, le robot doit passer par un point de dégagement **pDepart** et un point d'approche **pAppro**.

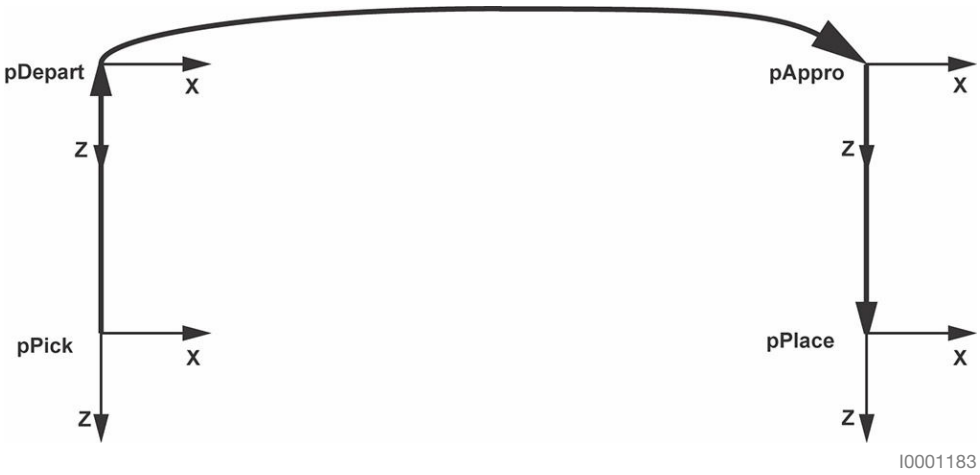


Figure 10.4 : Cycle en : U

On suppose que le robot est initialement au point **pPick**. Le programme pour exécuter le mouvement peut s'écrire :

```
movel(pDepart, tTool, mDesc)
movej(pAppro, tTool, mDesc)
movel(pPlace, tTool, mDesc)
```

On utilise des mouvements en ligne droite pour le dégagement et l'approche. En revanche, le mouvement principal est un mouvement point à point car il n'est pas nécessaire de contrôler précisément la géométrie de cette portion de la trajectoire, l'objectif étant d'aller le plus vite possible.

Pour les deux types de mouvements, la géométrie de la trajectoire ne dépend pas de la vitesse à laquelle les mouvements sont exécutés. Le robot passe toujours au même endroit. Ceci est particulièrement important au moment du développement des applications. On peut commencer par exécuter les mouvements à faible vitesse, puis augmenter progressivement la vitesse sans déformer la trajectoire du robot.

10.1.2 - ENCHAÎNEMENT DE MOUVEMENTS : LISSAGE M0005691.1

10.1.2.1 - Lissage M0005692.1

Reprenons l'exemple de cycle en U du chapitre précédent. Sans gestion particulière de l'enchaînement des mouvements, le robot va s'arrêter aux points **pDepart** et **pAppro**, car la trajectoire présente des angles en ces points. Ceci augmente inutilement la durée de l'opération, et il n'est pas utile de passer exactement par ces points.

Il est possible de réduire de manière importante la durée du mouvement en " lissant " la trajectoire au voisinage des points **pDepart** et **pAppro**. Pour cela, on utilise le champ **blend** du descripteur de mouvement. Lorsque la valeur de ce champ est **off**, la trajectoire du robot s'arrête à chaque point. En revanche, quand le paramètre est réglé sur joint ou **Cartesian**, la trajectoire est lissée au voisinage de chaque point et le robot ne s'arrête plus aux points de passage.

Quand le champ **blend** a la valeur joint ou **Cartesian**, deux autres paramètres doivent être définis : **leave** et **reach**. Ces paramètres déterminent à quelle distance du point d'arrivée on quitte la trajectoire nominale (début du lissage), et à quelle distance du point d'arrivée on la rejoint (fin du lissage).

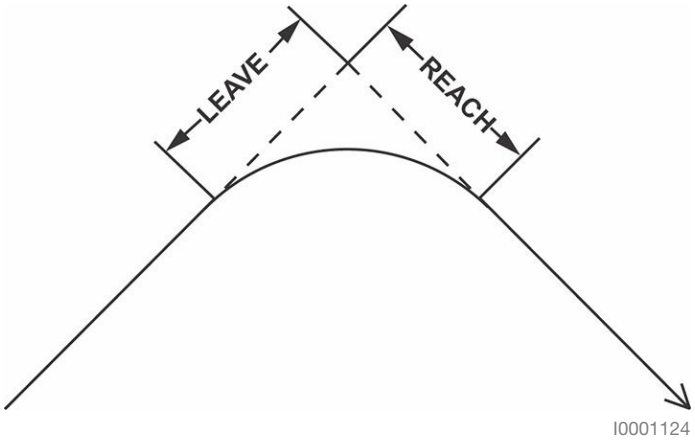


Figure 10.5 : Définition des distances : 'leave' / 'reach'

Exemples :

Reprenons le programme du paragraphe "Types de mouvement : point-à-point ou ligne droite". On peut modifier le programme de mouvement précédent :

```
mDesc.blend = joint
mDesc.leave = 50
mDesc.reach = 200
movel(pDepart, tTool, mDesc)
mDesc.leave = 200
mDesc.reach = 50
movej(pAppro, tTool, mDesc)
mDesc.blend = off
movel(pPlace, tTool, mDesc)
```

On obtient alors la trajectoire suivante :

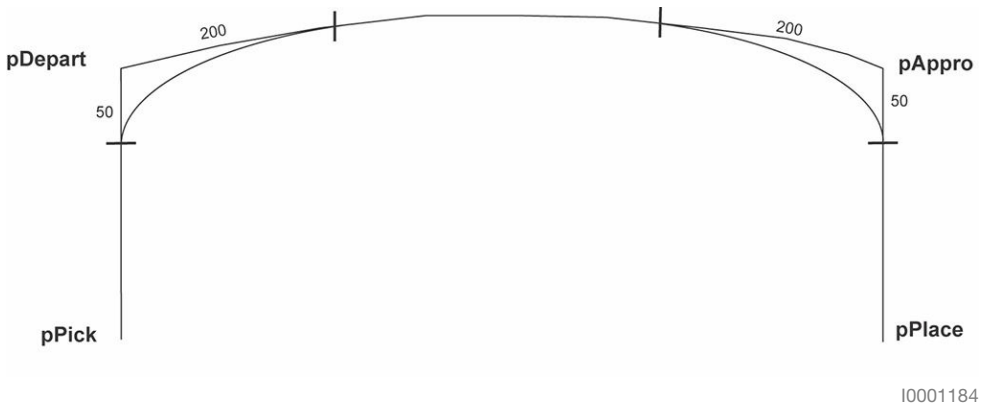


Figure 10.6 : Cycle lissé

Le robot ne s'arrête plus aux points **pDepart** et **pAppro**. Le mouvement est donc plus rapide. Il sera d'ailleurs d'autant plus rapide que les distances **leave** et **reach** sont grandes.

10.1.2.2 - Annulation du lissage

M0005693.1

L'instruction waitEndMove() permet, entre autre, d'annuler l'effet du lissage. Le robot termine alors complètement le dernier mouvement programmé jusqu'à son point d'arrivée, comme si le champ **blend** du descripteur de mouvement avait été mis à la valeur **off**.

Par exemple, considérons le programme suivant :

```
mDesc.blend = joint
mDesc.leave = 10
mDesc.reach = 10
movej(pA, tTool, mDesc)
movej(pB, tTool, mDesc)
waitEndMove()
movej(pC, tTool, mDesc)
movej(pD, tTool, mDesc)
```

etc...

La trajectoire suivie par le robot est alors la suivante :

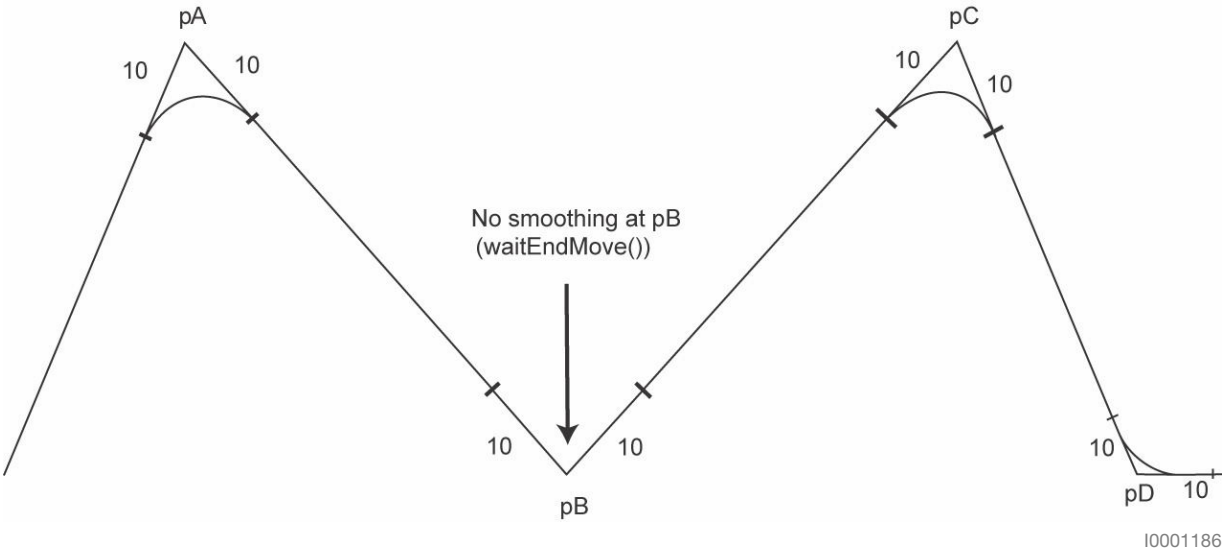


Figure 10.7 : Cycle avec interruption du lissage

10.1.2.3 - Lissage articulaire, lissage cartésien

M0005694.1

Pour simplifier, un lissage articulaire est comme un mouvement de point à point entre les points de départ et de destination et le lissage cartésien comme un déplacement circulaire entre ces points.

- Le lissage articulaire est habituellement plus rapide que le lissage cartésien. Il peut toutefois donner un trajet erratique en cas de changement complexe de l'orientation (habituellement un cercle dans un plan, suivi d'un cercle dans un plan perpendiculaire) ou pour les mouvements de rotation pure.
- Le contrôle de la vitesse et de l'accélération est plus précis avec le lissage cartésien. En outre, on est assuré qu'un lissage cartésien entre deux mouvements dans le même plan se situera dans ce plan.

La forme de lissage optimale dépend de l'application, mais l'interpréteur VAL 3 doit choisir cette forme automatiquement. Le résultat est rarement surprenant mais il peut arriver qu'il le soit...

Le choix peut donner une forme d'une complexité inattendue, avec des distances de départ et d'arrivée très différentes. La forme calculée réduit la courbure de la trajectoire afin d'optimiser la vitesse, mais le résultat peut ne pas être souhaitable pour certaines applications de processus. Quand les distances de départ et d'arrivée sont égales, le lissage cartésien donne toujours une forme simple.

Le lissage cartésien porte sur la position et sur l'orientation. Un changement d'orientaton complexe peut influer sur la forme du lissage et donner des résultats inattendus. Il existe également quelques restrictions concernant le changement d'orientation : dans un cercle, par exemple, les changements importants d'orientation peuvent donner une erreur de mouvement quand plusieurs solutions sont possibles mais le système n'a aucun critère pour en choisir une. Dans une telle situation, un ou plusieurs points intermédiaires supplémentaires sont nécessaires pour aider le système à trouver l'interpolation d'orientation correcte.

10.1.3 - REPRISE DE MOUVEMENT

M0005695.1

Lorsque la puissance du bras est coupée alors que le robot n'a pas fini ses mouvements, suite à un arrêt d'urgence par exemple, une reprise de mouvement doit se faire à la remise sous puissance. Si le bras a été déplacé manuellement pendant l'arrêt, il peut se trouver à une position éloignée de sa trajectoire normale. Il faut alors s'assurer que la reprise de mouvement peut se faire sans collision. Le contrôle de trajectoire du contrôleur VAL 3 donne la possibilité de gérer la reprise de mouvement à l'aide d'un " mouvement de connexion ".

A la reprise du mouvement, le système s'assure que le robot est bien sur la trajectoire programmée : en cas d'écart, même minime, il enregistre automatiquement une commande de mouvement point-à-point vers la position exacte où le robot a quitté sa trajectoire : c'est le " mouvement de connexion ". Ce mouvement se fait à vitesse réduite. Il doit être validé par l'opérateur, sauf en mode automatique, où il peut se faire sans intervention humaine. L'instruction autoConnectMove() permet de préciser le comportement en mode automatique.

L'instruction resetMotion() permet d'annuler le mouvement en cours, et éventuellement de programmer un mouvement de connexion permettant de rejoindre une position à vitesse réduite et sous le contrôle de l'opérateur.

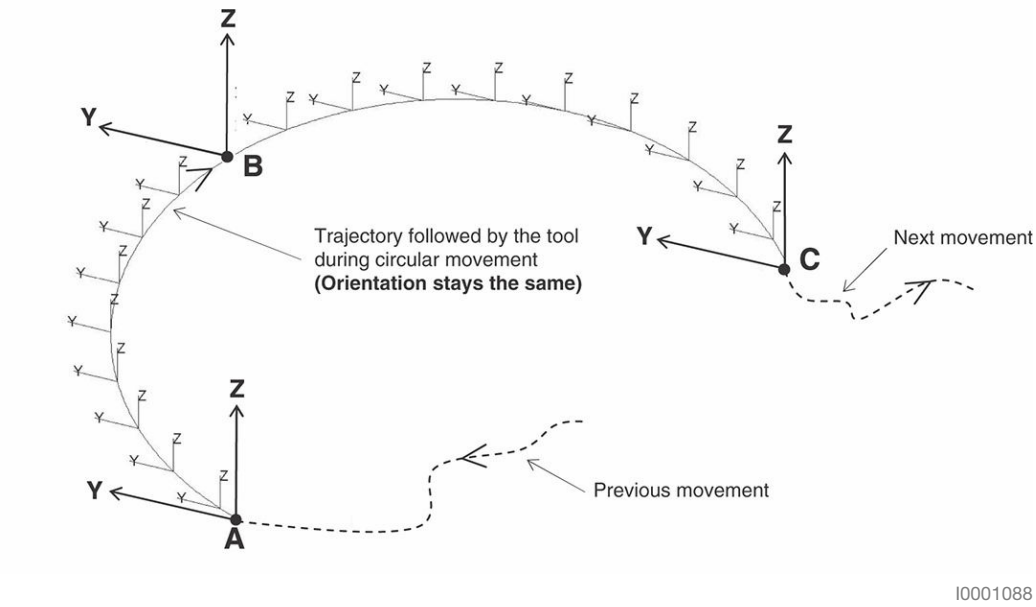
10.1.4 - PARTICULARITÉS DES MOUVEMENTS CARTÉSIENS (LIGNE DROITE, CERCLE) M0005696.1

10.1.4.1 - Interpolation de l'orientation M0005697.1

Le générateur de trajectoire de la VAL 3 minimise toujours l'amplitude des rotations de l'outil pour passer d'une orientation à une autre.

Ceci permet, comme cas particulier, de programmer une orientation constante, en absolu, ou par rapport à la trajectoire, sur tous les mouvements en ligne droite ou circulaires.

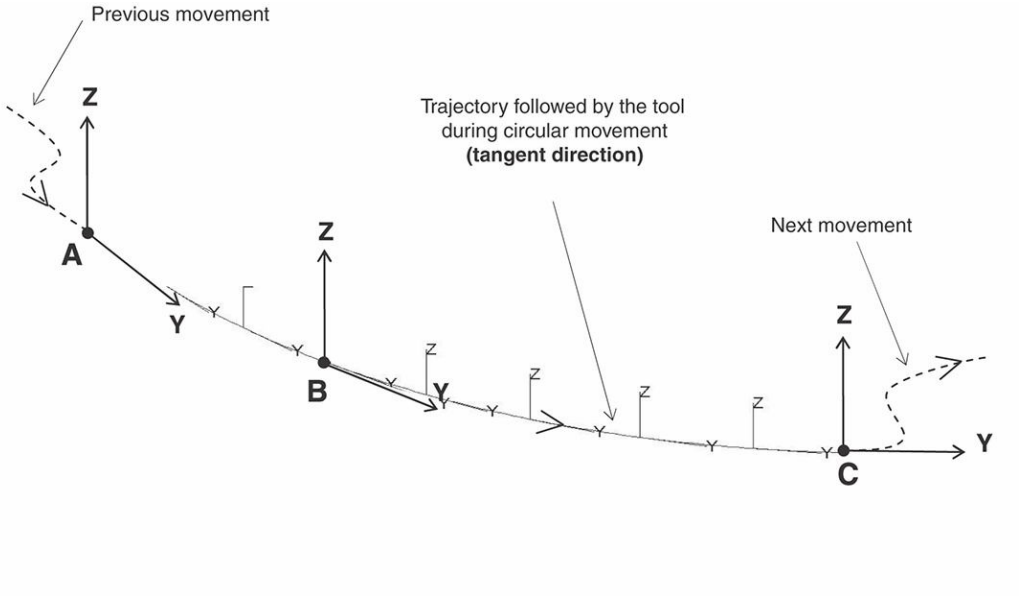
- Pour une orientation constante, les positions de départ, d'arrivée et la position intermédiaire pour un cercle, doivent avoir la même orientation.



Anglais	Traduction
Trajectory followed by the tool during circular movement (Orientation stays the same)	Trajectoire décrite par l'outil pendant le mouvement circulaire (orientation reste constante)
Next movement	Mouvement suivant
Previous movement	Mouvement précédent

Figure 10.8 : Orientation constante en absolu

- Pour une orientation constante par rapport à la trajectoire (par exemple direction Y du repère outil tangente à la trajectoire), les positions de départ, d'arrivée et la position intermédiaire pour un cercle, doivent avoir la même orientation par rapport à la trajectoire.



I0001089

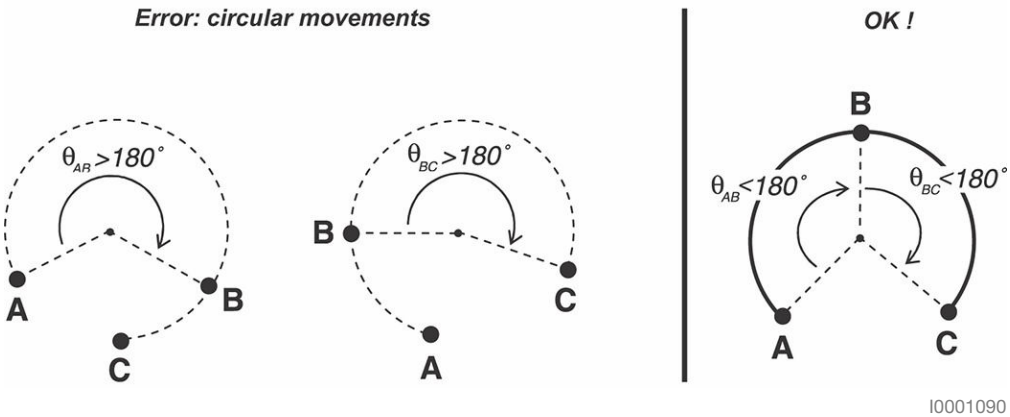
Anglais	Traduction
Trajectory followed by the tool during circular movement (tangent direction)	Trajectoire décrite par l'outil pendant le mouvement circulaire (orientation tangente)
Next movement	Mouvement suivant
Previous movement	Mouvement précédent

Figure 10.9 : Orientation constante par rapport à la trajectoire

Il en découle une limitation pour les mouvements circulaires :

Si le point intermédiaire fait un angle de **180°** ou plus avec le point de départ ou le point d'arrivée, il y a plusieurs solutions d'interpolation de l'orientation, et une erreur est générée.

Il faut alors modifier la position du point intermédiaire pour lever l'ambiguïté sur les orientations intermédaires.

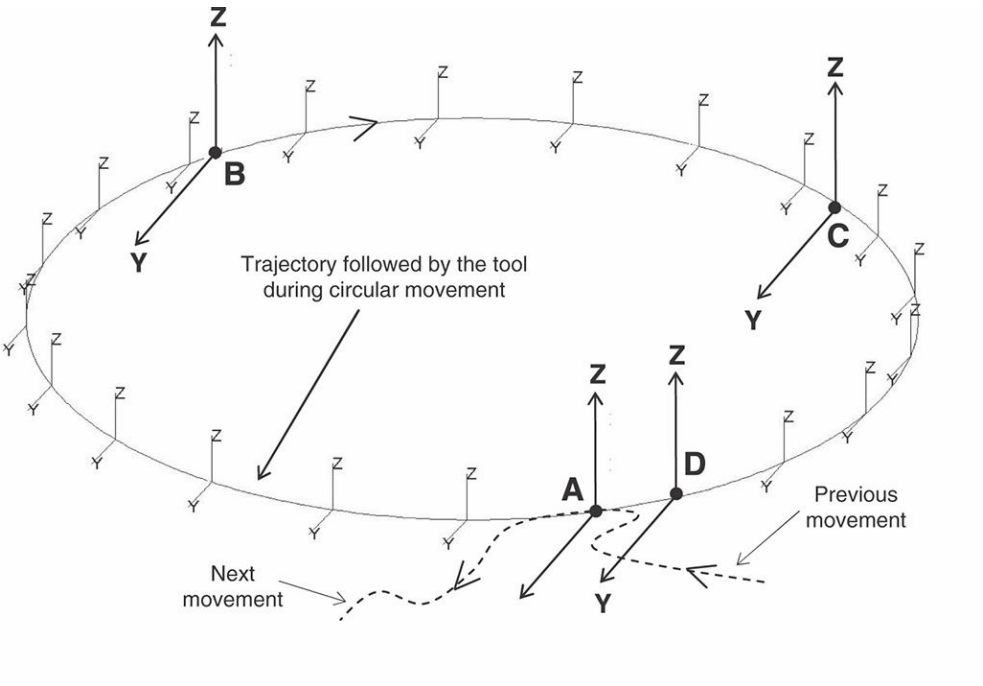


I0001090

Figure 10.10 : Ambiguïté sur l'orientation intermédiaire

En particulier, la programmation d'un cercle complet nécessite **2** instructions **movec** :

```
movec (B, C, tTool, mDesc)
movec (D, A, tTool, mDesc)
```



Anglais	Traduction
Trajectory followed by the tool during circular movement	Trajectoire décrite par l'outil pendant le mouvement circulaire
Next movement	Mouvement suivant
Previous movement	Mouvement précédent

Figure 10.11 : Cercle complet

10.1.4.2 - Changement (Bras TX2)

M0005698.1

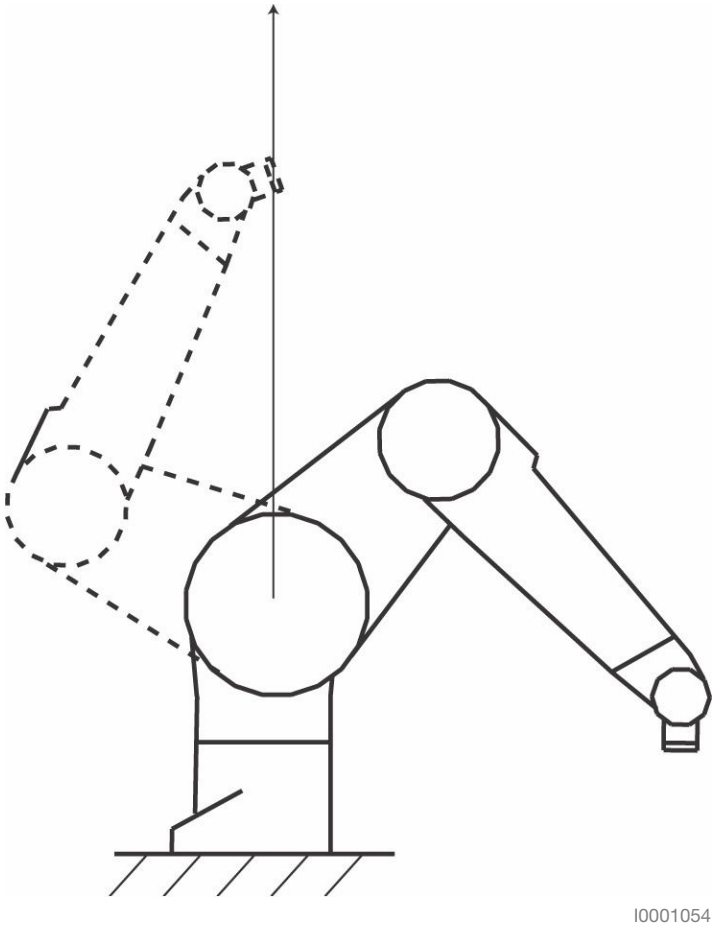


Figure 10.12 : Changement : righty / lefty

Lors d'un changement de la configuration de l'épaule, le centre du poignet du robot passe nécessairement à la verticale de l'axe 1 (pas exactement pour les robots avec déport).

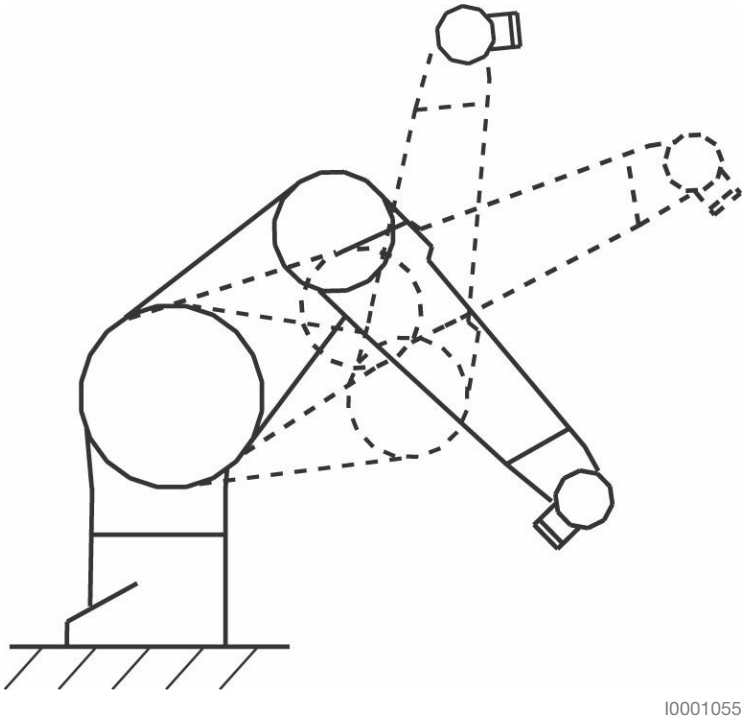


Figure 10.13 : Changement positive/negative du coude

Lors d'un changement de la configuration du coude, le bras passe nécessairement par la position bras tendu **(j3 = 0°)**.

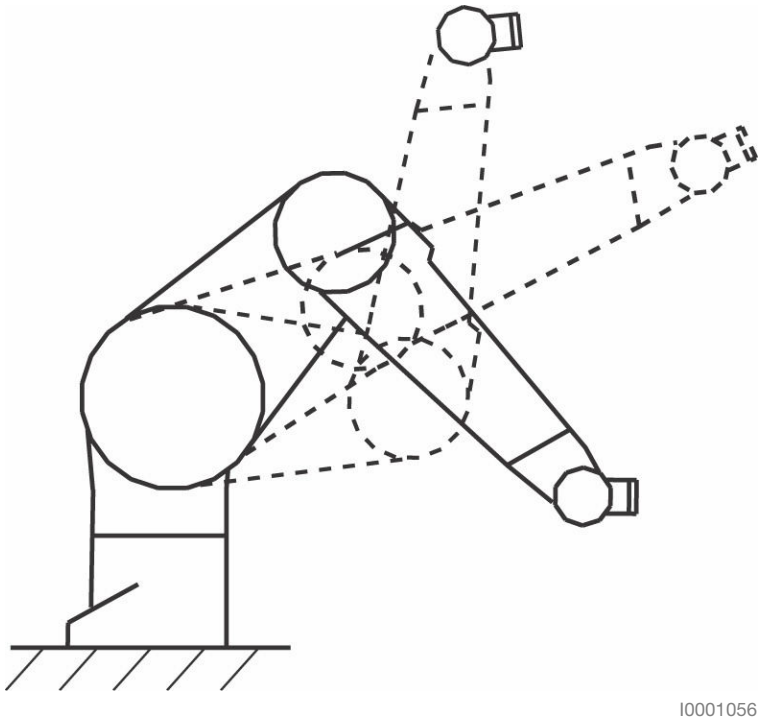


Figure 10.14 : Changement positive/negative du poignet

Lors d'un changement de la configuration du poignet, le bras passe nécessairement par la position poignet tendu **(j5 = 0°)**.

Ainsi pour changer de configuration, le robot doit obligatoirement passer par des positions particulières. Mais on ne peut pas imposer qu'un mouvement en ligne droite ou circulaire passe par ces positions si elles ne sont pas sur la trajectoire désirée! En conséquence, on ne peut pas imposer un changement de configuration dans un mouvement en ligne droite ou circulaire.

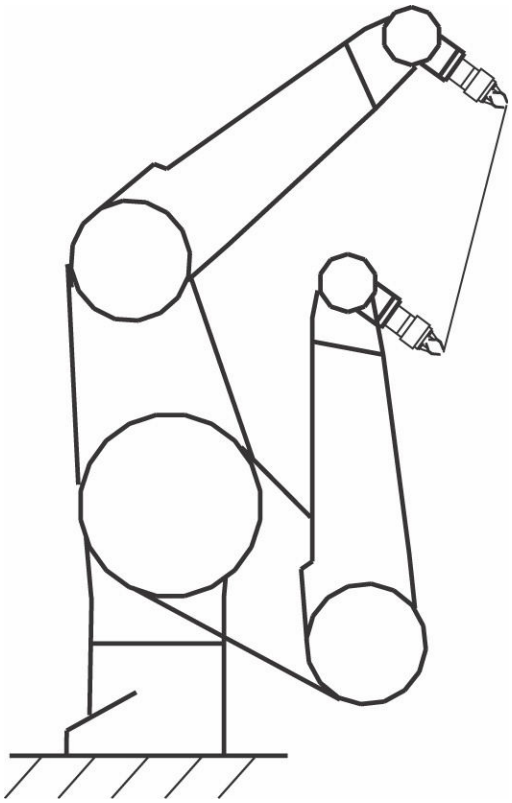
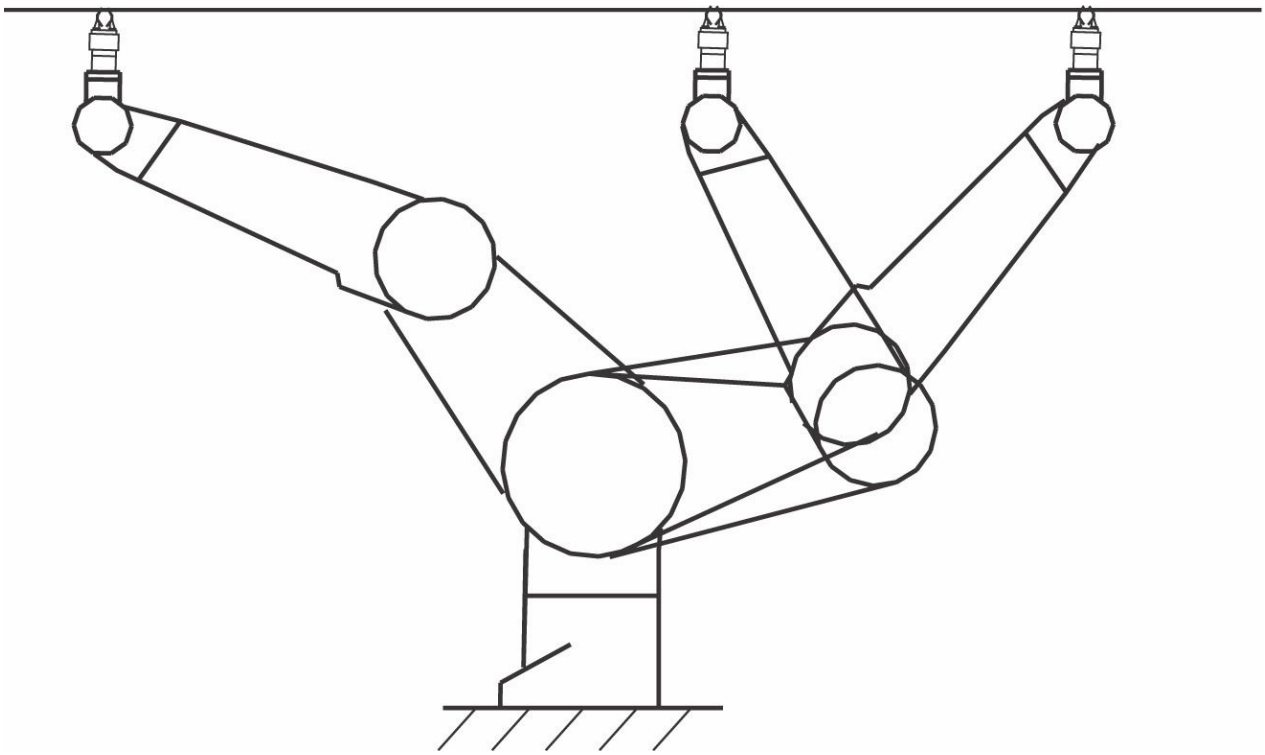


Figure 10.15 : Changement de configuration du coude impossible

Autrement dit, dans un mouvement en ligne droite ou circulaire, on ne peut imposer une configuration que si elle est compatible avec la position de départ : on peut donc toujours spécifier une configuration libre, ou bien identique à celle de départ.

Dans certains cas exceptionnels, la ligne droite ou l'arc de cercle passe effectivement par une position où un changement de configuration est possible. Dans ce cas, si la configuration a été laissée libre, le système peut décider de changer de configuration dans un mouvement en ligne droite ou circulaire.

Pour un mouvement circulaire, la configuration du point intermédiaire n'est pas prise en compte. Seules comptent les configurations des positions de départ et d'arrivée.



I0001058

Figure 10.16 : Changement de configuration de l'épaule possible

10.1.4.3 - Singularités (Bras TX2)

M0005699.1

Les singularités sont une caractéristique inhérente à tous les robots **6** axes. Les singularités peuvent être définies comme les points où le robot change de configuration. Alors certains axes se trouvent alignés : deux axes alignés se comportent comme un seul axe, et le robot **6** axes se comporte donc localement comme un robot **5** axes. Certains mouvements de l'effecteur sont alors impossibles à réaliser. Cela n'est pas gênant dans un mouvement point-à-point : les mouvements générés par le système sont toujours possibles. En revanche, dans un mouvement en ligne droite ou circulaire, on impose la géométrie du mouvement. Si le mouvement est impossible, une erreur est générée lors du mouvement du robot.

10.2 - ANTICIPATION DES MOUVEMENTS

M0005813.1

10.2.1 - PRINCIPE

M0005700.1

Le système contrôle les mouvements du robot un peu comme un pilote qui conduit une voiture. Il adapte la vitesse du robot à la géométrie de la trajectoire. Aussi, plus la trajectoire est connue à l'avance, plus le système peut optimiser la vitesse du mouvement. C'est pourquoi le système n'attend pas que le mouvement en cours du robot soit terminé pour prendre en compte les prochaines instructions de mouvement.

Considérons les lignes de programme suivantes :

```
movej(pA, tTool, mDesc)
movej(pB, tTool, mDesc)
movej(pC, tTool, mDesc)
movej(pD, tTool, mDesc)
```

On suppose que le robot est à l'arrêt lorsque l'exécution du programme atteint ces lignes. Lorsque la première instruction est exécutée, le robot commence son mouvement vers le point **pA**. L'exécution du programme se poursuit alors immédiatement avec la deuxième ligne, bien avant que le robot n'atteigne le point **pA**.

Lorsque le système exécute la deuxième ligne, le robot est en train de commencer son mouvement vers **pA** et le système enregistre, qu'après le point **pA**, le robot devra aller au point **pB**. L'exécution du programme se poursuit alors avec la ligne suivante : alors que le robot continue toujours son mouvement vers **pA**, le système enregistre qu'après le mouvement vers **pB**, le robot devra se diriger en **pC**. Comme l'exécution du programme est beaucoup plus rapide que les mouvements réels du robot, il est probable que le robot soit toujours en train de se diriger vers **pA** au moment où la ligne suivante sera exécutée. Le système enregistrera ainsi les points successifs suivants.

Ainsi, au moment où le robot commence sont mouvement vers **pA**, il 'sait' déjà qu'après **pA**, il devra aller successivement en **pB**, **pC** puis **pD**. Si le lissage a été activé, le système sait que le robot ne s'arrêtera pas avant le point **pD**. Il peut alors accélérer beaucoup plus que s'il devait se préparer à s'arrêter en **pB** ou **pC**.

L'exécution des lignes de commande ne fait qu'enregistrer les commandes de mouvement successives. Le robot les effectue ensuite suivant ses possibilités. La mémoire dans laquelle les mouvements sont enregistrés est importante, pour permettre au système d'optimiser la trajectoire au maximum. Néanmoins, elle est limitée. Lorsqu'elle est pleine, l'exécution du programme s'arrête sur la prochaine instruction de mouvement. L'exécution du programme reprend dès que le robot a terminé le mouvement en cours, libérant ainsi de la place dans la mémoire du système.

10.2.2 - ANTICIPATION ET LISSAGE

M0005701.1

On revient dans ce paragraphe à ce qui se passe en détail lorsque des mouvements sont enchaînés. Reprenons l'exemple précédent :

```
movej(pA, tTool, mDesc)
movej(pB, tTool, mDesc)
movej(pC, tTool, mDesc)
movej(pD, tTool, mDesc)
```

On suppose que le lissage est activé dans le descripteur de mouvement mDesc. Lorsque la première ligne est exécutée, le système ne sait pas encore quel sera le mouvement suivant. Seul le mouvement entre le point de départ et le point **pA leave** est entièrement déterminé, le point **pA leave** étant déterminé par le système à partir de la donnée **leave** du descripteur de mouvements (voir figure suivante).

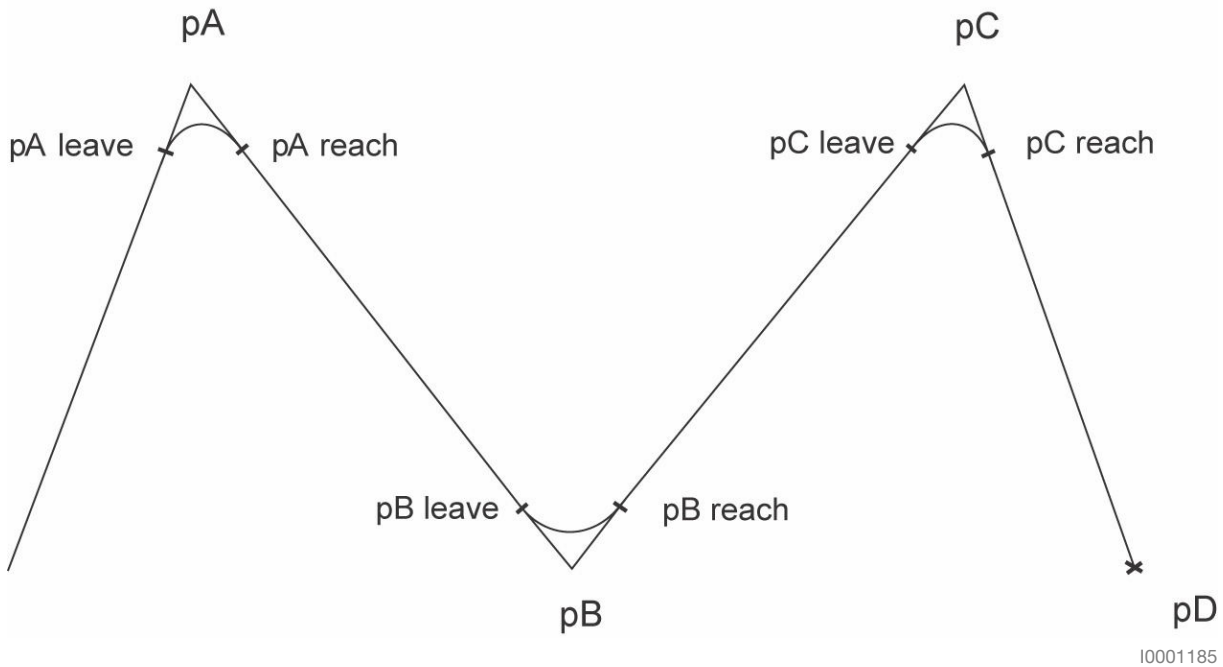


Figure 10.17 : Cycle lissé

Tant que la deuxième ligne n'est pas exécutée, la portion de trajectoire de lissage au voisinage du point **pA** n'est pas entièrement déterminée, puisque le système n'a pas encore pris en compte le mouvement suivant. En mode pas-à-pas, tant que l'utilisateur ne passera pas à l'instruction suivante, le robot n'ira pas plus loin que le point **pA leave**. Au moment où l'instruction suivante est exécutée, la trajectoire de lissage au voisinage du point **pA** (entre **pA leave** et **pA reach**) peut être définie, ainsi que le mouvement jusqu'au point **pB leave**. Le robot peut alors avancer jusqu'à **pB leave**. En mode pas-à-pas, il ne dépassera pas ce point tant que l'utilisateur n'exécute pas la troisième instruction, et ainsi de suite.

L'intérêt de ce mode de fonctionnement est que le robot passe exactement au même endroit en mode pas-à-pas et en exécution normale du programme.

10.2.3 - SYNCHRONISATION

M0005702.1

Le mécanisme d'anticipation induit une désynchronisation entre les lignes d'instructions VAL 3 et les mouvements correspondants du robot : le programme VAL 3 est en avance sur le robot.

Lorsque l'on veut effectuer une action à une position donnée du robot, il faut que le programme attende que le robot ait terminé ses mouvements : l'instruction waitEndMove() permet cette synchronisation. On peut aussi utiliser l'instruction getMoveId() pour détecter la progression du bras sur la trajectoire (voir 10.4, contrôle du mouvement en temps réel).

Ainsi, dans le programme suivant :

```
movej(A, tTool, mDesc)
movej(B, tTool, mDesc)
waitEndMove()
movej(C, tTool, mDesc)
movej(D, tTool, mDesc)
```

etc...

Les deux premières lignes seront exécutées alors que le robot commence sont mouvement vers **A**. Puis, l'exécution du programme sera bloquée sur la troisième ligne jusqu'à ce que le robot soit stabilisé au point **B**. Une fois le mouvement du robot stabilisé en **B**, le programme reprendra son exécution.

Les instructions open() et close() attendent également la fin des mouvements du robot avant d'actionner l'outil.

10.3 - CONTRÔLE DE VITESSE

M0005703.1

10.3.1 - PRINCIPE

M0005704.1

Le principe du contrôle de vitesse sur une trajectoire est le suivant :

A chaque instant, le robot se déplace et accélère au maximum de ses capacités, tout en respectant les contraintes de vitesse et d'accélération données dans la commande de mouvement.

Les commandes de mouvement contiennent deux types de contraintes de vitesse, définies dans une variable de type mdesc :

1. Les contraintes de vitesse (de l'articulation), d'accélération et décélération
2. les contraintes de vitesses cartésiennes du centre outil

L'accélération détermine la rapidité avec laquelle la vitesse augmente au début d'une trajectoire. A l'inverse, la décélération définit la rapidité avec laquelle la vitesse diminue en fin de trajectoire. Lorsqu'on utilise des grandes valeurs d'accélération et de décélération, les mouvements sont plus rapides, mais plus saccadés. Avec des faibles valeurs, les mouvements prennent un peu plus de temps, mais sont plus doux.

10.3.2 - RÉGLAGE SIMPLE

M0005705.1

Lorsque l'outil et l'objet transportés par le robot ne nécessitent pas de précaution de manipulation particulière, les contraintes sur les vitesses cartésiennes sont inutiles. La mise au point de la vitesse sur la trajectoire se fera typiquement de la façon suivante :

1. Mettre les contraintes de vitesse cartésienne à de très grandes valeurs, par exemple les valeurs par défaut, pour qu'elles n'interviennent pas dans la suite du réglage.
2. Initialiser la vitesse, l'accélération et la décélération en utilisant les valeurs nominales **(100%)**.
3. Régler ensuite la vitesse sur la trajectoire à l'aide du paramètre de vitesse.

Afin de conserver un comportement harmonieux du bras, l'accélération et la décélération doivent être modifiées avec la vitesse : les paramètres d'accélération et de décélération doivent correspondre approximativement au carré du paramètre de vitesse. Ainsi, la meilleure adaptation d'une vitesse de 120 % = 1.2 est obtenue avec une accélération et une décélération de 1.2 x 1.2 = 1.44 = 144 %. Les valeurs d'accélération et de décélération plus élevées donnent un comportement du bras plus agressif, mais aussi plus instable.

10.3.3 - RÉGLAGE AVANCÉ

M0005706.1

Lorsque l'on veut maîtriser la vitesse cartésienne au niveau de l'outil, par exemple pour obtenir une trajectoire effectuée à vitesse constante, on procédera plutôt de la manière suivante :

1. Mettre les contraintes de vitesses cartésiennes aux valeurs souhaitées à priori.
2. Initialiser la vitesse, l'accélération et la décélération en utilisant les valeurs nominales (100%).
3. Régler ensuite la vitesse sur la trajectoire à l'aide des seuls paramètres de vitesse cartésienne.
4. Si l'on n'arrive pas à atteindre la vitesse désirée, augmenter les paramètres d'accélération et de décélération.

   Si l'on souhaite freiner automatiquement dans les parties à forte courbure, diminuer les paramètres d'accélération et de décélération.

10.3.4 - ERREUR DE TRAÎNÉE

M0005707.1

Les valeurs nominales de vitesse et d'accélération articulaires sont les valeurs de charge nominale supportées par le robot, quelle que soit la trajectoire.

Mais le robot peut bien souvent aller plus vite : les vitesses maximales atteignables par le robot dépendent de sa charge et de la trajectoire. Dans des cas favorables (faible charge, impact favorable de la gravité) le robot pourra dépasser ses valeurs nominales sans aucun dommage.

Lorsque le robot porte une charge plus lourde que sa charge nominale, ou si l'on utilise de trop grandes valeurs pour les paramètres de vitesse et d'accélération articulaires, le robot ne pourra dans certains cas plus suivre sa consigne de mouvement et s'arrêtera alors sur une erreur de traînée. En spécifiant des paramètres de vitesse et d'accélération articulaires plus faibles, on pourra éviter ces erreurs de traînée.

DANGER

Dans les mouvements en ligne droite, près d'une singularité, un petit déplacement de l'outil nécessite de grands mouvements articulaires. Si la vitesse articulaire est réglée à une valeur trop élevée, le robot ne pourra pas suivre sa consigne et s'arrêtera sur une erreur de traînée.

10.4 - CONTRÔLE DU MOUVEMENT EN TEMPS RÉEL

M0005846.1

Les commandes de mouvement abordées jusqu'à présent n'ont pas un effet immédiat : au moment où chacune d'entre elle est exécutée, un ordre de mouvement est enregistré dans le système. Le robot exécute ensuite les mouvements enregistrés.

Il est possible d'intervenir immédiatement sur le mouvement du robot des manières suivantes :

- La vitesse moniteur modifie la vitesse de tous les déplacements. Elle peut être réglée avec effet immédiat à l'aide de l'instruction setMonitorSpeed(). Cette instruction ne peut cependant pas augmenter la vitesse quand l'opérateur peut aussi régler celle-ci à partir du MCP.
- Les instructions stopMove() et restartMove() permettent l'arrêt et la reprise de mouvement sur la trajectoire.
- L'instruction resetMotion() permet d'arrêter le mouvement en cours et d'annuler les commandes de mouvements enregistrées.
- L'instruction Alter (option) applique sur la trajectoire une transformation géométrique (translation, rotation, rotation au centre outil) qui est immédiatement effective.
- L'instruction getMoveId() permet de suivre, précisément et en temps réel, la position du robot sur sa trajectoire. Chaque instruction de mouvement est identifiée par une valeur numérique renvoyée par l'instruction. L'instruction getMoveId() renvoie une valeur numérique identifiant le mouvement courant (partie entière) et la progression de ce mouvement (partie décimale). Ainsi, un identifiant de mouvement de 17.572 signifie que le mouvement actuel est l'instruction de mouvement qui a renvoyé 17 et que la position du robot a atteint 57.2 % de ce mouvement. La plage des valeurs identifiant le mouvement est [-80 000, +80 000]. Une fois que la valeur maximale de 80 000 est atteinte, les identifiants des mouvements suivants saturent à 80 000. La valeur identifiant le mouvement peut être réinitialisée avec l'instruction setMoveId().

10.5 - TYPE MDESC

M0005708.1

10.5.1 - DÉFINITION

M0005709.1

Le type mdesc permet de définir les paramètres d'un mouvement (vitesse, accélération, lissage).

Le type mdesc est un type structuré dont les champs sont, dans l'ordre :

num accel	Accélération articulaire maximale autorisée, en % de l'accélération nominale du robot.
num vel	Vitesse articulaire maximale autorisée, en % de la vitesse nominale du robot.
num decel	Décélération articulaire maximale autorisée, en % de la décélération nominale du robot.
num tvel	Vitesse maximale autorisée de translation du centre outil, en mm/s ou pouce/s selon l'unité de longueur de l'application.
num rvel	Vitesse maximale autorisée de rotation de l'outil, en degrés par seconde.
blend blend	Mode de lissage : **off** (pas de lissage), **joint** ou **Cartesian** (lissage cartésien).
num leave	Dans les modes de lissage **joint** et **Cartesian**, la distance entre le point cible auquel commence le lissage et le point suivant, en mm ou en pouces, selon l'unité de longueur de l'application.
num reach	Dans les modes de lissage **joint** et **Cartesian**, la distance entre le point cible auquel finit le lissage et le point suivant, en mm ou en pouces, selon l'unité de longueur de l'application.

L'explication détaillée de ces différents paramètres est donnée au début du chapitre "Contrôle des mouvements".

Par défaut, une variable de type **mdesc** est initialisée avec **{100,100,100,9999,9999,joint,50,50}**.

10.5.2 - OPÉRATEURS

M0005710.1

Par ordre de priorité croissant :

mdesc <mdesc& desc1> = <mdesc desc2>	Affecte chaque champ de **desc2** au champ correspondant de la variable **desc1**.
bool <mdesc desc1> != <mdesc desc2>	Renvoie **true** si **desc1** et **desc2** diffèrent par la valeur d'au moins un champ.
bool <mdesc desc1> == <mdesc desc2>	Renvoie **true** si **desc1** et **desc2** ont les mêmes valeurs de champs.

10.6 - INSTRUCTIONS DE MOUVEMENT

M0005711.1

num **movej**(joint ...

M0005829.1

num movej(joint **jPosition**, tool **tTool**, mdesc **mDesc**)

num movej(point **pPosition**, tool **tTool**, mdesc **mDesc**)

Fonction

Cette instruction enregistre une commande pour un mouvement articulaire vers les positions **pPosition** ou **jPosition** à l'aide des paramètres de mouvement **tTool** et **mDesc**. Elle renvoie l'identifiant de mouvement attribué à ce mouvement et incrémente de un l'identifiant de mouvement pour la prochaine commande de mouvement.

DANGER

Le système n'attend pas la fin du mouvement pour passer à l'instruction VAL 3 suivante : plusieurs commandes de mouvements peuvent être enregistrées à l'avance. Lorsque le système n'a plus de mémoire disponible pour enregistrer une nouvelle commande, l'instruction attend jusqu'à ce que l'enregistrement puisse se faire.

L'explication détaillée des différents paramètres de mouvement est donnée au début du chapitre "Contrôle des mouvements".

Une erreur d'exécution est générée si **mDesc** contient des valeurs invalides, si **jPosition** se situe en dehors des butées logicielles, si **pPosition** ne peut pas être atteinte ou si une commande de mouvement précédemment enregistrée ne peut pas être exécutée (position hors d'atteinte).

Le premier paramètre de la fonction movej peut être une valeur de joint (type articulation) ou de point (type point). En raison d'une limitation interne, le compileur n'est pas en mesure de vérifier le type de données en entrée lorsqu'une expression explicite est saisie pour ce paramètre.

Exemples

```
movej({0,0,0,0,0,0},flange,mNomSpeed)
"{0,0,0,0,0,0}": Ambiguous expression
The above expression cannot be used and must be replaced by the following one...
myJoint={0,0,0,0,0,0}
movej(myJoint,flange,mNomSpeed)
```

Voir aussi

movel

isInRange

waitEndMove

movec

Fonction

Cette instruction enregistre une commande de mouvement linéaire vers le point **pPosition** à l'aide de l'outil **tTool** et des paramètres de mouvement **mDesc**. Elle renvoie l'identifiant de mouvement attribué à ce mouvement et incrémente de un l'identifiant de mouvement pour la prochaine commande de mouvement.

DANGER
Le système n'attend pas la fin du mouvement pour passer à l'instruction VAL 3 suivante : plusieurs commandes de mouvements peuvent être enregistrées à l'avance. Lorsque le système n'a plus de mémoire disponible pour enregistrer une nouvelle commande, l'instruction attend jusqu'à ce que l'enregistrement puisse se faire.

L'explication détaillée des différents paramètres de mouvement est donnée au début du chapitre "Contrôle des mouvements".

Une erreur d'exécution est générée si **mDesc** contient des valeurs invalides, si **pPosition** ne peut pas être atteinte, si un mouvement en ligne droite vers **pPosition** n'est pas possible ou si une commande de mouvement précédemment enregistrée ne peut pas être exécutée (destination hors d'atteinte).

Voir aussi

movej

waitEndMove

movec

num **movec**(point pIntermediate, point pTarget, tool tTool, mdesc mDesc) M0005815.1

Fonction

Cette instruction enregistre une commande pour un mouvement circulaire qui part de la destination du mouvement précédent et s'achève au point **pTarget** en passant par le point **pIntermediate**. Elle renvoie l'identifiant de mouvement attribué à ce mouvement et incrémente de un l'identifiant de mouvement pour la prochaine commande de mouvement.

L'orientation de l'outil est interpolée de sorte qu'il est possible de programmer une orientation constante en absolu, ou par rapport à la trajectoire.

DANGER
Le système n'attend pas la fin du mouvement pour passer à l'instruction VAL 3 suivante : plusieurs commandes de mouvements peuvent être enregistrées à l'avance. Lorsque le système n'a plus de mémoire disponible pour enregistrer une nouvelle commande, l'instruction attend jusqu'à ce que l'enregistrement puisse se faire.

L'explication détaillée des différents paramètres de mouvement et de l'interpolation de l'orientation est donnée au début du chapitre "Contrôle des mouvements".

Une erreur d'exécution est générée si **mDesc** a des valeurs invalides, si point **pIntermediate** (ou point **pTarget**) ne peut pas être atteint, si le mouvement circulaire n'est pas possible (voir le chapitre "Contrôle des mouvements - Interpolation de l'orientation") ou si une commande de mouvement précédemment enregistrée ne peut pas être exécutée (destination hors d'atteinte).

Voir aussi

movej

movel

waitEndMove

void **stopMove**() M0005852.1

Fonction

Cette instruction arrête le bras sur la trajectoire et suspend l'autorisation du mouvement programmé.

DANGER
Cette instruction retourne immédiatement : la tâche VAL 3 n'attend pas l'arrêt du bras pour exécuter l'instruction suivante.

Le descripteur de mouvement utilisé pour exécuter l'arrêt est celui utilisé pour le mouvement en cours.

Les mouvements ne peuvent reprendre qu'après exécution d'une instruction restartMove() ou resetMotion().

Les mouvements non programmés (déplacement manuels) restent possibles.

Exemples

```
// waits for a signal
wait(diSignal==true)
// stops movements along the trajectory
stopMove()
wait(diSignal==false)
// restarts movements along the trajectory
restartMove()
```

Voir aussi

restartMove

resetMotion

void **resetMotion**(), void **resetMotion**(joint jStartingPoint) M0005851.1

Fonction

Cette instruction arrête le bras sur sa trajectoire et annule toutes les commandes de mouvement enregistrées.

DANGER
Cette instruction retourne immédiatement : la tâche VAL 3 n'attend pas l'arrêt du bras pour exécuter l'instruction suivante.

L'autorisation de mouvement programmé est restaurée si elle avait été suspendue par l'instruction stopMove().

Si la position de l'articulation **jStartingPoint** est spécifiée, la commande de mouvement suivante ne peut être exécutée qu'à partir de cette position : un mouvement de connexion devra d'abord être effectué pour rejoindre **jStartingPoint**.

Si aucune position articulaire n'est précisée, la commande de mouvement suivante est exécutée à partir de la position courante du bras, quelle que soit celle-ci.

Voir aussi

isEmpty

stopMove

autoConnectMove

setMoveId

resetTurn

void **restartMove**() M0005771.1

Fonction

Cette instruction rétablit l'autorisation de mouvement programmée et relance la trajectoire interrompue par l'instruction stopMove().

Si l'autorisation de mouvement programmé n'avait pas été interrompue par l'instruction stopMove(), cette instruction n'a aucun effet.

Voir aussi

stopMove

resetMotion

void **waitEndMove**() M0005772.1

Fonction

Cette instruction annule le lissage de la dernière commande de mouvement enregistrée et attend l'exécution de la commande.

Cette instruction n'attend pas que le robot soit stabilisé dans sa position finale : elle attend seulement que la commande de position envoyée aux variateurs corresponde à la position finale souhaitée. Lorsqu'il est nécessaire d'attendre la stabilisation complète du mouvement, il faut utiliser l'instruction isSettled().

Une erreur d'exécution est générée si un mouvement enregistré précédemment ne peut pas être exécuté (destination hors d'atteinte).

Exemples

(voir chapitre 10.2)

Voir aussi

isSettled

isEmpty

stopMove

resetMotion

bool **isEmpty**() M0005773.1

Fonction

Cette instruction renvoie **true** si toutes les commandes de mouvement ont été exécutées, et **false** si au moins une commande est encore en cours d'exécution.

Exemples

Ce programme annule les mouvements enregistrés, s'il y en a :

```
//  If commands are in progress
if isEmpty()==false
 // Stop the robot and cancel the commands
 resetMotion()
 sOutput="Movements have been cancelled"
endIf
```

Voir aussi

waitEndMove

resetMotion

bool **isSettled**(), bool **isSettled**(tool tTool, num nTransAccuracy), bool **isSettled**(tool tTool, num nTransAccuracy, num nRotAccuracy, num nTime)

M0005828.1

Fonction

Cette instruction renvoie **true** si le robot est arrêté et **false** si sa position n'est pas encore stabilisée.

DANGER
Le robot peut être arrêté pour différentes raisons et il peut donc être stabilisé avant que tous les mouvements enregistrés soient exécutés. Utilisez isEmpty() pour savoir si le robot est arrêté à la fin de son mouvement programmé.

Sans argument, la position est considérée comme stabilisée si l'erreur de position de chaque articulation est inférieure à 1% de l'erreur maximum admissible de position articulaire pendant le temps de stabilisation par défaut du bras (une demi-période du mode oscillation nomal, qui varie en fonction du type de bras).

Avec 2 arguments (précision de la translation et de l'outil) la position est considérée comme stabilisée lorsque la position cartésienne de la pointe de l'outil conserve la précision par défaut donnée pendant le temps de stabilisation par défaut du bras.

Avec 4 arguments (outil, précision de la translation et de la rotation, durée) la position est considérée comme stabilisée lorsque la position et l'orientation cartésiennes de la pointe de l'outil conservent la précision donnée pendant le temps donné.

Voir aussi

isEmpty

waitEndMove

void **autoConnectMove**(bool bActive), bool **autoConnectMove**() M0005865.1

Fonction

En mode déporté, le mouvement de connexion est automatique si le bras est très proche de sa trajectoire (distance inférieure à l'erreur de traînée maximale autorisée). Si le bras est trop éloigné de sa trajectoire, le mouvement de connexion sera automatique ou sous contrôle manuel selon le mode défini par l'instruction autoConnectMove : automatique si **bActive** vaut **true**, sous contrôle manuel si **bActive** vaut **false**. Appelé sans paramètre, autoConnectMove retourne le mode de mouvement de connexion courant.

Par défaut, le mouvement de connexion en mode déporté est sous contrôle manuel.

DANGER
Dans des conditions normales d'utilisation, le bras s'arrête sur sa trajectoire lors d'un arrêt d'urgence. Par conséquent, en mode déporté, le bras pourra repartir automatiquement quelque soit le mode de mouvement de connexion défini par l'instruction autoConnectMove.

Voir aussi

resetMotion

num **getSpeed**(tool tTool) M0005712.1

Fonction

Cette instruction renvoie la vitesse de translation cartésienne courante au TCPtTool de l'outil **tTool** spécifié. La vitesse est calculée à partir de la commande de vitesse de l'articulation et non du retour de vitesse de l'articulation.

Voir aussi

here

joint **getPositionErr**() M0005808.1

Fonction

Cette instruction renvoie l'erreur de position articulaire actuelle du bras. L'erreur de position articulaire est la différence entre la commande de position articulaire envoyée aux moteurs et le retour de position articulaire mesuré par les codeurs.

Voir aussi

getJointForce

void **getJointForce**(num& nForce) M0005810.1

Fonction

Cette instruction renvoie le couple actuel (N.m pour l'axe de rotation) ou la force (N pour l'axe linéaire) sur l'articulation calculés à partir des courants du moteur.

La force sur l'articulation n'est pas une estimation directe des efforts externes. Elle inclut aussi la gravité, le frottement, la viscosité, l'inertie, le bruit et la précision des capteurs de courant, la relation entre le courant et le couple moteur. Elle ne peut être utilisée pour estimer les efforts externes qu'en enregistrant les forces dans des conditions de référence et en les comparant aux forces mesurées dans des conditions similaires avec des efforts externes supplémentaires.

Elle ne donne qu'un ordre de grandeur des forces. Sa précision n'est pas garantie et doit être évaluée pour chaque application.

Une erreur d'exécution est générée si le paramètre n'est pas un tableau de valeurs numériques d'une taille suffisante.

Voir aussi

getPositionErr

num **getMoveId**() M0005816.1

Fonction

Cette instruction renvoie une valeur numérique indiquant la position actuelle du robot sur la trajectoire. La partie entière identifie le numéro de l'instruction de mouvement en cours d'exécution. Ce nombre entier est renvoyé quand l'instruction de mouvement est exécutée. La partie décimale donne le % de progression de ce mouvement.

Un identifiant de mouvement ne doit jamais être testé avec l'opérateur '==' mais avec l'opérateur '>=' : wait(getMoveId()==12) ne peut jamais être renvoyé parce que l'identifiant de mouvement peut augmenter en un incrément de 11.998 à 12.013 et ne jamais prendre exactement la valeur (12) attendue. Il est préférable d'écrire plutôt wait(getMoveId()>=12).

Si l'identifiant de mouvement a atteint son maximum de 80 000, une erreur d'exécution est générée lors de l'appel de getMoveId().

Exemples

Cet exemple montre comment l'identifiant de mouvement change sur une trajectoire simple :

```
nIdA = movel(pA, tTool, mDesc)
nIdB = movel(pB, tTool, mDesc)
waitEndMove()
nId = getMoveId()
```

Pendant l'exécution de ce programme :

Supposons que la valeur renvoyée nIdA est 15. nIdB est alors 16 : le mouvement est automatiquement incrémenté d'un à chaque instruction de mouvement.

- quand getMoveId() est 15.8, la position du robot se trouve à 80 % du mouvement vers le point pA.
- quand getMoveId() est 16.572, la position du robot se trouve à 57.2 % du mouvement vers le point pB.
- quand getMoveId() est 17, la position du robot se trouve à 100 % du mouvement 16, donc au point pB.

La valeur de nId, après waitEndMove(), est donc de nIdB+1=17.

Voir aussi

movej

movel

movec

num **setMoveId**(num nMoveId) M0005817.1

Fonction

Cette instruction change l'identifiant de mouvement pour l'instruction de mouvement suivante. Elle est utile pour faire en sorte que la même trajectoire utilise toujours les mêmes valeurs d'identifiant de mouvement. Après un resetMotion, l'identifiant de mouvement est automatiquement réinitialisé à 0.

Après l'utilisation de setMoveId() ou resetMotion(), la relation entre un identifiant de mouvement et une instruction de mouvement peut devenir incertaine : plusieurs mouvements enregistrés peuvent alors avoir le même identifiant de mouvement. Il est donc préférable de ne pas attribuer à setMoveId() une valeur qui serait aussi l'identifiant de mouvement d'une commande de mouvement en cours.

La valeur de moveId doit se situer dans la plage [-80 000, +80 000]. Dans le cas contraire, une erreur d'exécution est générée.

Exemples

```
resetMotion()
nId1 = getMoveId()
setMoveId(1000)
nId2 = getMoveId()
nId3 = movel(pA, tTool, mDesc)
nId4 = movel(pB, tTool, mDesc)
waitEndMove()
nId5 = getMoveId()
```

Après l'exécution de ce programme, on a :

- nId1 est 0, parce que l'identifiant de mouvement est mis à 0 après resetMotion()
- nId2 est 1000 : l'identifiant de mouvement vient d'être changé avec setMoveId()
- nId3 est 1000 : une instruction de mouvement renvoie l'identifiant de mouvement défini précédemment et l'incrémente pour le mouvement suivant
- nId4 est 1001 : l'identifiant de mouvement a été incrémenté par l'instruction de mouvement qui précède
- nId5 est 1002 : après waitEndMove(), 100 % du mouvement 1001 est exécuté et l'identifiant du mouvement est donc 1001+1 = 1002

Voir aussi

getMoveId

resetMotion

11 - DÉCLARATION DE LA CHARGE UTILE DU ROBOT

M0005713.1

Il est possible de déclarer les charges réelles déplacées par le bras.

Cela améliore la précision statique du bras obtenue avec l'option "étalonnage absolu" car celle-ci active une compensation de la flexibilité.

Plus généralement, cette fonction améliore le comportement du bras quand l'application exige une dynamique très élevée car l'anticipation du couple est plus précise lorsque la charge réelle est déclarée.

11.1 - PRINCIPE

M0005714.1

Les charges sont définies par un poids et une inertie. Il existe trois ensembles de paramètres pour chaque articulation du bras :

- Les paramètres "système". Définis dans le fichier system.zfx, ils décrivent la charge du bras proprement dite. Ils ne peuvent pas être lus ni modifiés par l'utilisateur.
- Les paramètres "cellule". Ils définissent les charges statiques montées sur le bras. Un préhenseur ou un équipement monté sur l'avant-bras, par exemple, est une charge statique. Ces paramètres sont enregistrés dans le fichier cell.cfx et initialisés une fois pendant le démarrage du contrôleur. Consulter le manuel d'utilisation du contrôleur pour définir les charges de la cellule. Si aucune charge utile n'est déclarée dans cell.cfx, une charge utile par défaut est utilisée au démarrage. Dans ce cas, un message d'avertissement est enregistré dans l'historique des événements. Cette charge utile par défaut est supprimée dès qu'une charge utile est déclarée par la fonction setPayload().
- Les paramètres "utilisateur" représentent la charge utile du robot : Il s'agit habituellement des pièces qui sont saisies et mises en place pendant l'application. La charge utile peut être modifiée de façon dynamique à partir de l'application VAL 3. Ils sont effacés au démarrage du système.

Le modèle dynamique du contrôleur utilise la combinaison de tous ces paramètres.

Cette fonction n'est pas prise en charge par tous les bras. Ceci peut être vérifié par l'application VAL 3 (voir fonctions VAL 3 ci-dessous) ou en contrôlant l'historique des événements où la charge utile réelle du robot a été enregistrée au démarrage du système : si on ne trouve aucune information sur la charge utile dans ce fichier, cette fonction n'est pas prise en charge par le bras.

num **setPayload**(tool tTool, num nMass, trsf trGravityCenter, num& nInertiaMatrix[]) M0005716.1

Fonction

Cette fonction modifie la charge utile du robot.

Paramètres	Description
tTool	L'outil où sont déclarées les coordonnées du centre de gravité.
nMass	La masse de la charge utile en kg.
trGravityCenter	La position du centre de gravité, en coordonnées **tTool**. Seuls les champs x, y et z sont utilisés.
nInertiaMatrix	La matrice d'inertie de la charge utile, à son centre de gravité, dans les coordonnées de l'outil (unité de mesure $kg.m^2$). Différents formats sont possibles : ■ tableau 3x3 2 dims, ■ tableau 9 éléments 1 dim, ■ tableau 3 éléments 1 dim (matrice diagonale), ■ tableau 1 éléments 1 dim (aucune inertie).

Cette fonction renvoie un code d'erreur :

Code	Description
0	Modification effectuée.
-1	La charge utile n'est pas prise en charge pour ce bras.
-3	L'inertie est déjà en cours de modification (le processus de basculement d'une charge utile à une autre dure le temps du boxcar).
-4	La masse de l'entrée est négative.
-5	La matrice d'inertie de l'entrée est asymétrique.

Fonction

Cette fonction acquiert la charge utile réelle.

Les paramètres sont les mêmes que setPayload().

Si nInertiaMatrix est un tableau 1D, il est redimensionné à 9 et rempli avec Ixx, Ixy, Ixz, Iyx, Iyy, Iyz, Izx, Izy et Izz (sauf si l'inertie est symétrique et la taille du tableau est déjà 3. Si nInertiaMatrix est un tableau 2D, il est redimensionné à 3x3 et rempli naturellement avec [Ixx Ixy Ixz; Iyx Iyy Iyz; Izx Izy Izz]. Les tableaux locaux ne pouvant pas être redimensionnés, si nInertiaMatrix est une variable locale avec une taille erronée, une erreur d'exécution est générée.

Codes éventuellement renvoyés :

Code	Description
1	La charge utile du dernier axe (6 pour un robot à 6 axes et 4 pour un robot à 4 axes) est la valeur par défaut. Cela signifie que la charge utile du dernier axe n'a été déclarée ni dans le fichier cell.cfx ni avec la fonction setPayload.
0	Pas d'erreur.
-1	La charge utile n'est pas prise en charge pour ce bras.

12 - EVÈNEMENTS SYSTÈMES

M0005718.1

Codes d'événement du système

Les événements détectés sur le contrôleur sont enregistrés dans le fichier /log/system.log. Ces événements sont identifiés par un numéro unique. Consulter la documentation du contrôleur pour les codes d'événements existants.

void **getIds**(num& nEvtId)

M0005719.1

Fonction

Cette fonction permet de retrouver jusqu'aux 20 derniers événements détectés par le système dans le tableau **nEvtId**.

- num& **nEvtId** est une référence à un tableau où le système stocke les derniers événements détectés. Les événements sont classés selon un ordre chronologique, le dernier événement est stocké à l'index 0 du tableau. Si la taille du tableau est inférieure à 20, seuls les événements les plus récents sont stockés en fonction de la place disponible.

Exemples

L'exemple ci-dessous affiche, chaque seconde, les derniers codes événement reçus dès qu'un nouvel événement se produit.

```
begin
 //
 l_nLastId=-1
 do
  delay(1)
  getIds(nEvtId)
  if nEvtId[0]!=l_nLastId
   sOutput=""
   l_nLastId=nEvtId[0]
   for l_nI=0 to size(nEvtId)-1
    sOutput=sOutput+"; "toString("",nEvtId[l_nI])
   endFor
  endIf
 until false
end
```

num **getEvents**(...)

M0005720.1

num getEvents(num& **x_nEvtNbr,** num& **x_nId**[], num& **x_nType**[], string& **x_sInfo**[])

num getEvents(num& **x_nEvtNbr,** num& **x_nId**[], num& **x_nType**[], string& **x_sInfo**[], num& **nSeverity**[])

Fonction

Cette fonction lit les derniers événements du système qui se sont produits sur le contrôleur. En effet, le système mémorise en continue les 20 derniers événements qui se sont produits. Chaque événement est numéroté par le système, à partir de 1, selon un incrément d'une unité à chaque nouvel événement.

La fonction copie les derniers événements dans les tableaux de paramètres donnés **x_nId, x_nType**, **x_sInfo**, **x_nSeverity** en commençant par le dernier numéro d'événement donné dans **x_nEvtNbr**. Le nombre N d'événements à copier est donné par la taille minimum des tableaux de paramètres, dans la limite de la taille du tampon d'événements interne (20). Dans les tableaux, les événements sont classés du plus ancien au plus récent.

Si **x_nEvtNbr** se réfère à un numéro d'événement qui n'existe plus dans le tampon interne (plus petit que l'événement le plus ancien), la fonction copie les **N** événements disponibles les plus anciens.

Le paramètre **x_nEvtNbr** renvoie le numéro du premier événement réellement lu et la fonction renvoie le numéro du dernier événement réellement lu.

Si **x_nEvtNbr** se réfère à un numéro d'événement qui n'existe plus (supérieur à l'événement le plus récent) dans le tampon interne, la fonction ne copie rien. Le paramètre **x_nEvtNbr** est inchangé, aucune copie n'est effectuée et la fonction renvoie le numéro du dernier événement disponible.

Paramètres des entrées/sorties

x_nEvtNbr :

- Entrée : Numéro du premier événement à lire.
- Sortie : Numéro du premier événement réellement lu.

Paramètres de sortie

- **x_nId** : Tableau d'identifiants d'événement.
- **x_nType** : Tableau de types d'événement (0=Contextuel, 1=Historique, 2=Journal).
- **x_sInfo** : Tableau de description textuelle des événements.
- **x_nSeverity** : Niveau de gravité de l'événement (1=Info, 2=Attention, 3=Erreur, 4=Critique).

Retour arrière

Nombre d'événements transmis depuis le dernier démarrage (voir ci-dessus).

Exemple de mémorisation des événements du système VAL 3

```
eventPolling()
begin
 // First loop read from event number 1
 l_nFirstEvt=-1
 do
  // Get pending events
  l_nEvt=l_nFirstEvt
  l_nLastEvt=getEvents(l_nFirstEvt, l_nId, l_nType, l_sInfo)
  // Compute number of read events
  l_nNbRead=l_nLastEvt-l_nFirstEvt+1
  // IF some events have been read
  if (l_nNbRead>0)
   // IF some events have been lost
   if (l_nEvt!=l_nFirstEvt)
   // TODO add your code here
   endIf
   // TODO add your code here
   // Set event number for next loop
   l_nFirstEvt=l_nLastEvt+1
  // IF rollover
  elseIf (l_nNbRead!=0)
   l_nFirstEvt=1
  endIf
  delay(0)
 until bStopLoop==true
end
```

13 - OPTIONS

M0005721.1

13.1 - ALTER : CONTROLE EN TEMPS REEL D'UNE TRAJECTOIRE

M0005722.1

DANGER
La fonction d'altération utilise une grande quantité de ressources en temps réel. En fonction du type de CPU, il pourra s'avérer nécessaire de réduire le temps de cycle du système.

Alter cartésien

13.1.1 - PRINCIPE

M0005723.1

L'altération cartésienne d'une trajectoire permet d'appliquer à la trajectoire une transformation géométrique (translation, rotation, rotation au centre outil) qui est immédiatement effective.

Cette fonction permet de modifier une trajectoire nominale au moyen d'un capteur externe, afin, par exemple, d'assurer un suivi précis de la forme d'une pièce, ou d'intervenir sur une pièce en mouvement.

13.1.2 - PROGRAMMATION

M0005724.1

La programmation consiste à définir d'abord la trajectoire nominale, puis à spécifier, en temps réél, une déviation sur cette trajectoire.

La trajectoire nominale est programmée de la même manière que les mouvements standards, au moyen des instructions alterMovel(), alterMovej() et alterMovec(). Plusieurs mouvements altérables peuvent se suivre, ou certains mouvements altérables peuvent alterner avec des mouvements non altérables. Nous définirons une trajectoire altérable comme la succession des commandes de mouvement altérables situées entre deux commandes de mouvement non altérables.

L'altération proprement dite est programmée au moyen de l'instruction alter(). Différents modes alter sont possibles en fonction de la transformation géométrique à appliquer ; le mode est défini au moyen de l'instruction alterBegin(). L'instruction alterEnd() sert, enfin, à indiquer de quelle manière doit se terminer l'altération, soit avant la fin du mouvement nominal, de manière à pouvoir séquencer le prochain mouvement non altérable sans provoquer d'arrêt ; soit après, de manière à pouvoir déplacer le bras au moyen d'alter pendant que le mouvement nominal est à l'arrêt.

Les autres instructions de commande de mouvement restent effectives en mode alter.

DANGER
Les instructions waitEndMove, open et close attendent la fin du mouvement nominal, et non pas la fin du mouvement altéré. L'exécution du VAL 3 peut alors reprendre après un waitEndMove même si le bras est encore en mouvement en raison d'une modification de l'altération.

13.1.3 - CONTRAINTES

M0005725.1

Synchronisation, désynchronisation : Comme la commande alter est appliquée immédiatement, la variation de l'altération doit être contrôlée de manière à ce que la trajectoire du bras résultante ne présente aucune discontinuité et aucun bruit :

- Un changement important de l'altération ne peut être appliqué que progressivement avec une commande d'approche spécifique.
- La fin de la modification exige une vitesse d'altération nulle, obtenue progressivement grâce à une commande d'arrêt spécifique.

Commande synchrone : Le contrôleur envoie des commandes de position et de vitesse toutes les 4 ms aux amplificateurs. En conséquence, la commande alter doit être synchronisée avec cette période de communication de manière à ce que la vitesse d'altération reste sous contrôle. Pour ce faire, on utilisera une tâche VAL 3 synchrone (voir chapitre Tâches). De la même manière, il faudra parfois filtrer préalablement une entrée capteur si les données sont bruitées ou si leur période d'échantillonnage n'est pas synchronisée avec la période du contrôleur.

Séquencement progressif : Le premier mouvement non altérable suivant une trajectoire altérable ne peut être calculé qu'après exécution de alterEnd. En conséquence, si alterEnd est exécuté trop peu de temps après la fin du mouvement altérable, le bras risque de ralentir voire de s'arrêter à proximité de ce point tant que le mouvement suivant n'est pas calculé.

De plus, la nécessité de calculer à l'avance le mouvement suivant impose des restrictions à la trajectoire altérée après exécution de alterEnd : Elle doit conserver la même configuration, et il faut s'assurer que toutes les articulations restent dans le même tour d'axe. Il est alors possible qu'une erreur soit générée pendant le mouvement, alors qu'elle n'aurait pas eu lieu si alterEnd n'avait pas été exécuté à l'avance.

13.1.4 - SÉCURITÉ

M0005726.1

A tout moment, l'altération de l'utilisateur peut être invalide : destination hors de portée, vitesse ou accélération trop élevée. Lorsque le système détecte des situations de ce type, une erreur est générée et le bras est arrêté brusquement sur la dernière position valide. Le générateur de mouvement doit être réinitialisé pour repartir.

Lorsque le mouvement du bras est désactivé au cours d'un mouvement (touche 'Move/Hold', demande d'arrêt ou arrêt d'urgence), la commande d'arrêt s'exerce sur les mouvements nominaux comme pour les mouvements standards. Au bout d'un certain temps, le mode alter est lui aussi automatiquement désactivé afin de garantir un arrêt complet du bras. Lorsque l'état d'arrêt disparaît, le mouvement reprend et le mode alter est automatiquement réactivé.

13.1.5 - LIMITATIONS

M0005727.1

Aucun mouvement : Les mouvements nuls (lorsque la destination du mouvement se trouve sur la position de départ) ne sont pas pris en compte par le système. En conséquence, un mouvement non nul est nécessaire pour activer le mode altération. Une distance de mouvement de 0.001 mm est suffisante pour cela.

Configuration du robot : Il n'est pas possible de spécifier la configuration à appliquer à la trajectoire altérée ; le système utilise, en effet, toujours la même configuration. Il est donc impossible de modifier la configuration du bras à l'intérieur d'une trajectoire altérée (même en utilisant l'instruction alterMovej).

Séquencement de la trajectoire altérable : Il n'est pas possible de redémarrer l'altération avec alterBegin() après alterEnd() sans insérer un mouvement standard ((movel, movec, movej)) ou un resetMotion(). Chaque trajectoire altérable continue ne permet qu'un seul alterBegin().

num **alterMovej**(joint...) M0005729.1

num **alterMovej**(joint **jPosition**, tool **tTool**, mdesc **mDesc**)

num **alterMovej**(point **pPosition**, tool **tTool**, mdesc **mDesc**)

Fonction

Cette instruction enregistre une commande de mouvement d'articulation modifiable (une ligne dans l'espace de l'articulation). Elle renvoie l'identifiant de mouvement attribué à ce mouvement et incrémente de un l'identifiant de mouvement pour la prochaine commande de mouvement.

Paramètres

jPosition/pPosition	Expression de type point ou joint définissant la position de fin du mouvement.
tTool	Expression de type outil définissant le centre outil utilisé pendant le mouvement pour le contrôle de la vitesse cartésienne.
mDesc	Expression de type **mDesc** définissant les paramètres de contrôle de vitesse et de lissage pour le mouvement.

Détails

Cette instruction se comporte exactement comme l'instruction movej, à ceci près qu'elle autorise le mode alter pour le mouvement. Voir movej pour plus d'informations.

num **alterMovel**(point pPosition, tool tTool, mdesc mDesc) M0005730.1

Fonction

Cette instruction enregistre une commande de mouvement linéaire modifiable (une ligne dans l'espace cartésien). Elle renvoie l'identifiant de mouvement attribué à ce mouvement et incrémente de un l'identifiant de mouvement pour la prochaine commande de mouvement.

Paramètres

pPosition	Expression de type point définissant la position de fin du mouvement.
tTool	Expression de type outil définissant le centre outil utilisé pendant le mouvement pour le contrôle de la vitesse cartésienne. A la fin du mouvement, le centre outil se trouve à la position cible spécifiée.
mDesc	Expression de type mdesc définissant les paramètres de contrôle de vitesse et de lissage pour le mouvement.

Détails

Cette instruction se comporte exactement comme l'instruction movel, à ceci près qu'elle autorise le mode alter pour le mouvement. Voir movel pour plus d'informations.

num **alterMovec**(point pIntermediate, point pTarget, tool tTool, mdesc mDesc) M0005731.1

Fonction

Cette instruction enregistre une commande de mouvement circulaire modifiable. Elle renvoie l'identifiant de mouvement attribué à ce mouvement et incrémente de un l'identifiant de mouvement pour la prochaine commande de mouvement.

Paramètres

pIntermediate	Expression de type point définissant un point intermédiaire du cercle
pTarget	Expression de type point définissant la position de fin du mouvement.
tTool	Expression de type outil définissant le centre outil utilisé pendant le mouvement pour le contrôle de la vitesse cartésienne. A la fin du mouvement, le centre outil se trouve à la position cible spécifiée.
mDesc	Expression de type mdesc définissant les paramètres de contrôle de vitesse et de lissage pour le mouvement.

Détails

Cette instruction se comporte exactement comme l'instruction movec, à ceci près qu'elle autorise le mode alter pour le mouvement. Voir movec pour plus d'informations.

num **alterBegin**(... M0005801.1

num **alterBegin**(frame **fAlterReference**, mdesc **mMaxVelocity**)

num **alterBegin**(tool **tAlterReference**, mdesc **mMaxVelocity**)

Fonction

Cette instruction initialise le mode alter pour le trajet modifiable en cours d'exécution.

Paramètres

fAlterReference/tAlterReference	Expression de type frame ou tool définissant la référence de la déviation alter.
mMaxVelocity	Expression de type mdesc définissant les paramètres de vérification de sécurité de la déviation alter.

Détails

Le mode alter initié avec alterBegin ne se termine que par une commande alterEnd, ou par un resetMotion. Lorsque la fin d'une trajectoire altérable est atteinte, le mode alter demeure actif jusqu'à l'exécution de alterEnd.

L'expression trsf de la commande alter définit une transformation de la trajectoire entière autour de alterReference :

- La trajectoire pivote autour du centre du repère ou de l'outil en utilisant la partie de rotation de la trsf.
- La trajectoire est ensuite soumise à une translation par la partie de translation de la trsf.

Les coordonnées trsf de la commande alter sont définies dans le repère alterReference.

Lorsqu'un repère (frame) est utilisé comme référence, alterReference est fixe dans l'espace (World). Ce mode doit être utilisé lorsque l'altération de la trajectoire connue ou mesurée dans l'espace cartésien (pièce mobile telle que le suivi de convoyeur).

Lorsqu'un outil est utilisé comme référence, alterReference est fixe par rapport au centre outil. Ce mode doit être utilisé lorsque l'altération de la trajectoire est connue ou mesurée par rapport au centre outil (par exemple capteur de forme de pièce monté sur l'outil).

Le descripteur de mouvement est utilisé pour définir la vitesse articulaire et la vitesse cartésienne maximum sur la trajectoire altérée (à l'aide des champs vel, tvel et rvel du descripteur de mouvement). Une erreur est générée et le bras est arrêté sur la trajectoire si la vitesse altérée dépasse les limites spécifiées.

Les champs accel et decel du descripteur de mouvement contrôlent la durée de l'arrêt lorsqu'une condition d'arrêt survient (eStop, touche 'Move/Hold', VAL 3 stopMove()) : L'altération de la trajectoire doit être arrêtée en utilisant ces paramètres de décélération (voir alterStopTime).

alterBegin renvoie une valeur numérique indiquant le résultat de l'instruction :

1	alterBegin a été exécuté avec succès
0	alterBegin attend le début du mouvement altérable
-1	alterBegin n'a pas été pris en compte parce que le mode alter a déjà démarré
-2	alterBegin a été refusé (l'option alter n'est pas activée)
-3	alterBegin a été refusé parce que le mouvement est en erreur. Un resetMotion est requis.

Voir aussi

alterEnd

alter

alterStopTime

num **alterEnd**() M0005791.1

Fonction

Cette instruction quitte le mode alter et rend le mouvement en cours non modifiable.

Détails

Si alterEnd est exécuté alors que la fin de la trajectoire altérable est atteinte, le mouvement non altérable suivant (s'il en existe un) est démarré immédiatement.

Si alterEnd est exécuté avant la fin du mouvement altérable, la valeur actuelle de la déviation alter s'applique au reste de la trajectoire altérable, jusqu'au mouvement non altérable suivant. Il n'est pas possible de passer à nouveau en mode alter sur la même trajectoire altérable.

Le mouvement non altérable suivant, s'il en existe un, est calculé dès l'exécution de alterEnd afin que la transition entre la trajectoire altérable et le mouvement non altérable suivant se fasse sans arrêt.

alterEnd renvoie une valeur numérique indiquant le résultat de l'instruction :

1	alterEnd a été exécuté avec succès
-1	alterEnd n'a pas été pris en compte parce que le mode alter n'a pas encore démarré
-3	alterEnd a été refusé parce que le mouvement est en erreur. Un resetMotion est requis.

Voir aussi

alterBegin

num **alter**(trsf trAlteration) M0005792.1

Fonction

Cette instruction définit une nouvelle modification de la trajectoire modifiable.

Détails

La transformation induite par la trsf d'altération dépend du mode alter sélectionné par l'instruction alterBegin. Les coordonnées de l'altération sont définies dans le repère ou dans l'outil spécifié par l'instruction alterBegin.

L'altération est appliquée par le système toutes les 4 ms : Lorsque plusieurs instructions alter sont exécutées en moins de 4 ms, c'est la dernière qui s'applique. Le plus souvent, l'instruction alter doit être exécutée dans une tâche synchrone pour forcer un rafraîchissement de l'altération toutes les 4 ms.

L'altération doit être calculée minutieusement de manière à ce que les commandes de position et de vitesse du bras qui en résultent soient exécutées sans discontinuité et sans bruit. Il faudra parfois filtrer préalablement l'entrée capteur de manière à atteindre la qualité recherchée sur la trajectoire et le comportement du bras.

Lorsque le mouvement est arrêté (touche 'Move/Hold', arrêt d'urgence, instruction stopMove()), l'altération de la trajectoire est verrouillée jusqu'à ce que toutes les conditions d'arrêt soient annulées.

Lorsque l'altération de la trajectoire n'est pas valide (position hors d'atteinte, limites de vitesse dépassées), le bras s'arrête d'un coup sur la dernière position valide et le mode alter est verrouillé en position d'erreur. Un resetMotion est requis pour pouvoir repartir. Les limites de vitesse du mouvement alter sont définies par l'instruction alterBegin.

Alter renvoie une valeur numérique indiquant le résultat de l'instruction :

1	alter a été exécuté avec succès.
0	alter attend le redémarrage du mouvement (alterStopTime est nul).
-1	alter n'a pas été pris en compte parce que le mode alter n'a pas démarré ou est déjà terminé.
-2	alter a été refusé (l'option alter n'est pas activée).
-3	alter a été refusé parce que le mouvement est en erreur. Un resetMotion est requis.

Voir aussi

alterBegin

taskCreateSync

num **alterStopTime**() M0005793.1

Fonction

Cette instruction renvoie le temps restant avant que la déviation alter soit verrouillée quand une condition d'arrêt se présente.

Détails

Lorsqu'une condition d'arrêt survient, le système évalue le temps nécessaire à l'arrêt du bras si les paramètres accel et decel du descripteur de mouvement spécifiés par alterBegin sont utilisés. Le mimimum entre ce temps et le temps imposé par le système (généralement 0.5s lorsqu'un eStop survient) est renvoyé par alterStopTime.

Lorsque alterStopTime renvoie une valeur négative, il n'y a pas de condition d'arrêt active. Lorsque alterStopTime renvoie une valeur nulle, la commande alter est verrouillée jusqu'à ce que toutes les conditions d'arrêt aient été supprimées.

alterStopTime renvoie une valeur nulle lorsque le mode alter n'est pas activé.

Voir aussi

alterBegin

alter

13.2 - CONTRÔLE DE LICENCE OEM

M0005732.1

13.2.1 - PRINCIPES

M0005733.1

Une licence OEM est un code spécifique du contrôleur qui permet de limiter l'utilisation d'un projet VAL 3 à certains contrôleurs de robots sélectionnés.

Un outil de Stäubli Robotics Suite(*) permet de coder un mot de passe OEM secret en licence OEM publique propre au contrôleur, qui peut ensuite être installée sur le contrôleur sous la forme d'une option logicielle. A l'aide de l'instruction getLicence(), une application ou une librairie peut vérifier si la licence OEM est installée et s'assurer ainsi qu'elle est utilisée uniquement par le robot choisi.

Pour tenir secret le mot de passe OEM et protéger le code pendant le test de la licence, l'instruction getLicence() doit être utilisée dans une bibliothèque cryptée.

Le mode de démonstration des licences OEM est pris en charge ; dans ce cas, le contrôleur est simplement configuré avec le code "demo" et l'instruction getLicence() en informe l'origine de l'appel. Avec l'émulateur VAL 3, il suffit de sélectionner le mode "demo" pour activer complètement la licence OEM.

L'instruction getLicence() est une option VAL 3 qui nécessite l'installation d'une licence d'exécution (runtime) sur le contrôleur. Si cette licence d'exécution n'est pas définie, getLicence() renvoie un code d'erreur.

(*) Cet outil, un exécutable encrypttools.exe, ne peut être utilisé qu'avec une licence spécifique dans la Stäubli Robotics Suite.

13.2.2 - INSTRUCTIONS

M0005734.1

string **getLicence**(string sOemLicenceName, string sOemPassword) M0005804.1

Fonction

Cette instruction renvoie le statut de la licence OEM spécifiée :

"oemLicenceDisabled"	La licence de temps d'exécution VAL 3 "oemLicence" n'est pas activée sur le contrôleur : la licence OEM ne peut pas être testée.
""	La licence OEM sOemLicenceName n'est pas activée (mot de passe non défini ou invalide).
"demo"	La licence OEM sOemLicenceName est activée en mode démonstration.
"enabled"	La licence OEM sOemLicenceName est activée.

Voir aussi

Encryption

13.3 - ROBOT ABSOLU

M0005735.1

13.3.1 - PRINCIPE

M0005736.1

Un "robot absolu" est un robot qui utilise une identification des paramètres géométriques spécifiques au bras (souvent appelés "paramètres DH"). Ces paramètres sont spécifiques à chaque robot et correspondent à l'orientation et aux dimensions réelles de chaque articulation. Un robot absolu possède une précision accrue pour atteindre les positions cartésiennes définies par un outil de CAO ou calculées dans VAL 3 (par exemple des positions dans une palette). Les courbes cartésiennes (lignes longues, cercles) sont en outre plus précises. Le calibrage absolu ne modifie pas la répétabilité du bras.

Les paramètres DH se composent d'un ensemble de translations (a, b et d sur les axes X, Y, Z,) et de rotations (alpha, beta, theta autour des axes X, Y et Z)

La séquence de ces translations et rotations est définie de telle manière que la position de l'articulation {j1, j2, j3, j4, j5, j6} corresponde à la position cartésienne pCart au centre de la bride, avec :

```
pCart.trsf = {0,0,0,0,0, j1+theta[0]}
* {a[0], b[0], d[0], alpha[0], beta[0], j2+theta[1]}
* {a[1], b[1], d[1], alpha[1], beta[1], j3+theta[2]}
* {a[2], b[2], d[2], alpha[2], beta[2], j4+theta[3]}
* {a[3], b[3], d[3], alpha[3], beta[3], j5+theta[4]}
* {a[4], b[4], d[4], alpha[4], beta[4], j6+theta[5]}
* {a[5], b[5], d[5], alpha[5], beta[5], 0}
* {0, 0, d[6], 0, 0, 0}
```

Le paramètre d[6] n'est requis que pour certains poignets de bras.

13.3.2 - FONCTIONNEMENT

M0005737.1

Les paramètres géométriques doivent être identifiés à l'aide d'un outil de mesure séparé (par exemple d'un appareil de poursuite laser). Le langage VAL 3 ne comprend pas d'outil permettant cette opération de mesure, mais il fournit un moyen pour appliquer les paramètres géométriques mesurés au robot, avec effet immédiat. Les paramètres sont en outre sauvegardés dans le fichier de configuration du bras, de sorte qu'ils sont automatiquement récupérés au redémarrage suivant.

Les paramètres géométriques peuvent être modifiés dans une application VAL 3 en cours d'exécuton. On peut ainsi ajuster la géométrie du bras pendant le cycle de production, si la cellule comporte un outil de mesure adéquat.

13.3.3 - LIMITATIONS

M0005738.1

Les paramètres géométriques ne peuvent être modifiés dans une application VAL 3 en cours d'exécution que si le générateur de mouvements est vide (pas de mouvements en cours).

La géométrie modifiée d'un robot absolu a des propriétés mathématiques plus complexes qu'une géométrie standard. La notion de configuration du bras (type de configuration) ne peut plus être rigoureuse. La conversion d'une position cartésienne en position de l'articulation correspondante peut échouer près des limites de l'articulation ou dans des positions singulières.

13.3.4 - INSTRUCTIONS

M0005739.1

void **getDH**(num& theta[],... / **getDefaultDH**(num& theta[],... M0005814.1

---

void **getDH**(num& **theta**[],num& **d**[],num& **a**[],num& **alpha**[],num& **beta**[])void

void **getDefaultDH**(num& **theta**[],num& **d**[],num& **a**[],num& **alpha**[],num& **beta**[])

Fonction

Ces instructions renvoient les paramètres DH du bras dans les tableaux spécifiés. Les paramètres d et a sont des translations en mm ; les paramètres alpha, beta et theta sont des angles en degrés. getDH() renvoie les paramètres DH actuels du bras connecté au contrôleur. getDefaultDH() renvoie les paramètres DH standard pour le type de bras.

Voir aussi

setDH

bool **setDH**(num& theta[], num& d[], num& a[], num& b[], num& alpha[], num& beta[]) M0005825.1

Fonction

Cette instruction n'est activée qu'avec la licence d'exécution spécifique. Elle modifie la géométrie du bras avec les paramètres DH. Les paramètres d, a et b sont des translations en millimètres ; les paramètres alpha, beta et theta sont des angles en degrés. La modification est immédiate et s'applique aussi au fichier de configuration du bras, de sorte que la géométrie modifiée prend effet au redémarrage suivant.

L'instruction renvoie true si le changement a bien été effectué ou false si la nouvelle géométrie n'a pas été appliquée parce qu'elle est trop différente des paramètres de la géométrie standard. setDH() attend que le mouvement soit vide avant d'effectuer son opération.

La taille des tableaux DH doit correspondre au nombre d'axes du robot. Une entrée supplémentaire peut être nécessaire dans le tableau d pour modifier la dimension de la bride : quand il est absent, setDH() renvoie false et un message de diagnostic est envoyé à l'enregistreur.

Voir aussi

getDH

13.4 - AXE CONTINU

M0005740.1

13.4.1 - PRINCIPE

M0005741.1

L'axe 6 (pour les bras TX2) ou 4 (pour les bras TS2) peut être "continu" dans une application robotique quand seule sa position dans un tour est importante : le nombre de tours qu'il a effectués au cours du cycle est sans importance. Ce principe s'applique, par exemple, aux applications dans lesquelles le robot tient une pièce qui est usinée par un outil fixe.

L'option continuousAxis permet de réinitialiser automatiquement le nombre de tours effectués dans le cycle précédent avant le début d'un nouveau cycle. Par exemple, si l'axe commence le cycle de l'application dans la position 0° et le finit dans la position 720° (2 tours), le cycle suivant peut remettre instantanément la position à 0° et commence un nouveau cycle, sans qu'il faille ramener l'axe de 720° à 0°.

13.4.2 - INSTRUCTIONS

M0005742.1

joint **resetTurn**(joint jReference)

M0005743.1

Fonction

Cette instruction se comporte comme l'instruction resetMotion() standard, attend l'arrêt du bras et ajuste la position de l'axe continu afin de la rapprocher le plus possible de la position de référence spécifiée. Elle renvoie le modulo 180° de la position effective du bras après l'exécution de l'instruction. L'opération d'ajustement s'effectue indifféremment avec le bras sous tension ou hors tension et prend environ 50 ms. L'instruction resetTurn() modifie les données de calibration du bras. Au redémarrage suivant du contrôleur, la position du zéro sur l'axe est réinitialisée automatiquement (actualisation des données spécifiques du bras, dans le bras et sur le disque du contrôleur).

Cette instruction est sans effet en mode de fonctionnement manuel quand le mode de déplacement manuel (jog) est activé.

Voir aussi

resetMotion

14 - OPC UA SERVEUR

M0005744.1

14.1 - INTRODUCTION

M0005745.1

Le contrôleur peut recevoir un serveur OPC UA. Celui-ci est protégé par une licence et la clé "OpcUAServer" doit être activée. Il peut fonctionner pendant deux heures en mode démo. Le contrôleur ne comporte pas de client OPC UA. Il existe de nombreux clients gratuits disponibles sur Internet.

14.2 - CONNEXION AU SERVEUR

M0005746.1

14.2.1 - CONFIGURATION SIMPLE

M0005747.1

Quand le démarrage est terminé, deux messages sont enregistrés dans l'historique :

- Démarrage de l'OPC UA réussi.
- L'OPC UA est accessible à l'adresse : 'opc.tcp://[IP of controller]:[port]/Staubli'

Par défaut, le port est réglé à 4880 mais il est possible d'utiliser un autre port si celui-ci est déjà utilisé dans l'usine. La marche à suivre est indiquée dans les paragraphes qui suivent. Si ces deux messages ne s'affichent pas, le serveur OPC UA n'a pas démarré et aucune connexion n'est possible.

Attention : le port utilisé par l'OPC UA ne doit servir pour aucune autre application dans le réseau.

La deuxième entrée d'historique indique l'URL qui doit être saisie dans le client OPC UA pour établir la communication.

14.2.2 - CONFIGURATION AVANCÉE

M0005748.1

Le serveur utilise un protocole SSL pour crypter la communication. Par défaut, il possède un certificat auto-signé. La partition /usr/configs/ contient une infrastructure de clé publique (PKI). Si vous voulez utiliser votre propre certificat (ou certificat d'entreprise), vous pouvez enregistrer la clé publique et la clé privée dans le répertoire /usr/configs/pki/own et ajouter les informations suivantes dans network.cfx :

La clé privée doit être un fichier "pem" et la clé publique un fichier "der".

```
<opcUaServer>
  <String name="publicKey" value="name or path to the public key" />
  <String name="privateKey" value="name or path to the private key" />
  <String name="password" value="password to open private key" />
  <Uint name="portOpc" value="4880" />
</opcUaServer>
```

Le chemin des clés publiques ou privées part du répertoire pki. Par exemple, si vous mettez votre propre certificat dans le répertoire "own", vous devez saisir dans network.cfx "own/user_certificate.der".

Attention : le nom de l'hôte dans le certificat donné par l'utilisateur doit être le même que celui du robot.

Lors de la première connexion, le certificat du client OPC UA sera refusé et enregistré dans un répertoire de rejet (/usr/configs/pki/rejected/). Si vous déplacez ce fichier dans le répertoire de confiance, vous pourrez vous connecter au serveur lors de l'essai suivant.

14.2.3 - POLITIQUE DE SÉCURITÉ

M0005749.1

Plusieurs politiques de sécurité sont implémentées. L'utilisateur doit en choisir une pendant la procédure de connexion. Protocoles connus :

- Basic128Rsa15
- Basic256
- Basic256Sha256

Pour chaque protocole, vous pouvez choisir si les messages sont seulement signés ou signés et cryptés :

- Sign : vérifier si le message est reçu tel qu'il a été envoyé.
- Cryptage : le message est crypté au niveau 128 ou 256 bits.

Un mode sans sécurité est disponible sur l'émulateur pour le débogage.

14.2.4 - PROFIL UTILISATEUR ET DROITS DES UTILISATEURS

M0005750.1

Le serveur demande à l'utilisateur de s'identifier par un login. Il est possible de définir des droits d'accès pour restreindre ou étendre les capacités des utilisateurs. Cette fonction se base sur les profils d'utilisateur de CS9. Le mot de passe utilisé par le profil est celui défini dans le champ "password" de l'éditeur de profils.

Deux autres champs sont utilisés dans ce fichier :

- readAccess : l'utilisateur peut lire les variables et l'historique d'OPC UA.
- writeAccess : l'utilisateur peut écrire les variables, l'historique et l'exécutable d'OPC UA.

Les autres droits de CS9 ne sont pas utilisés sur le serveur. Il est également possible de se connecter en mode anonyme. Celui-ci permet un accès en lecture seule à tous les noeuds.

Un accès avec certificat est également possible. L'utilisateur doit mettre son certificat public dans USR://configs/pki/user/ et peut ensuite écrire des noeuds.

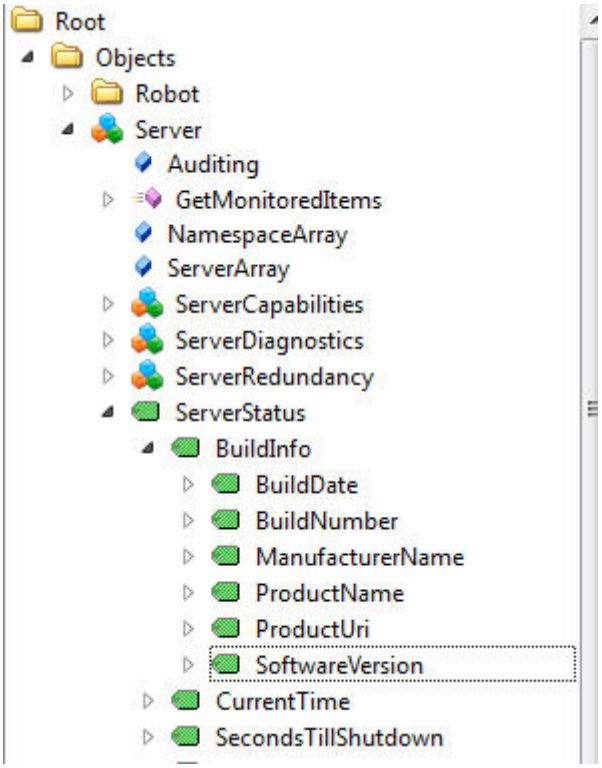
14.3 - OPC UA SERVEUR

M0005751.1

14.3.1 - GÉNÉRALITÉS

M0005752.1

Quel que soit le client OPC UA utilisé, les informations du serveur seront affichées sous la forme d'une arborescence. Le client peut récupérer les informations communes concernant le serveur dans le menu suivant. Sur le serveur OPC UA, faire attention à la version de l'interface. Si un changement survient et que la compatibilité n'est plus assurée, nous changerons le plus grand nombre. Le numéro de version est stocké dans la variable "SoftwareVersion".



I0005252

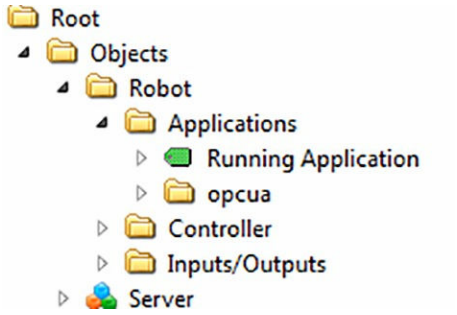
Le client OPC UA peut aussi changer l'intervalle de publication pour l'actualisation des données. Selon le client, la valeur par défaut peut être 500 ms ou même 1000 ms. CS9 accepte une valeur plus basse mais il peut y avoir un dépassement. Un message de journal arrête alors l'utilisateur et lui recommande d'augmenter la valeur.

Le menu Robot contient toutes les informations et actions concernant le robot. Il existe, par exemple, trois catégories : Application, Contrôleur et Entrées/Sorties.

14.3.2 - NOEUD APPLICATIONS

M0005753.1

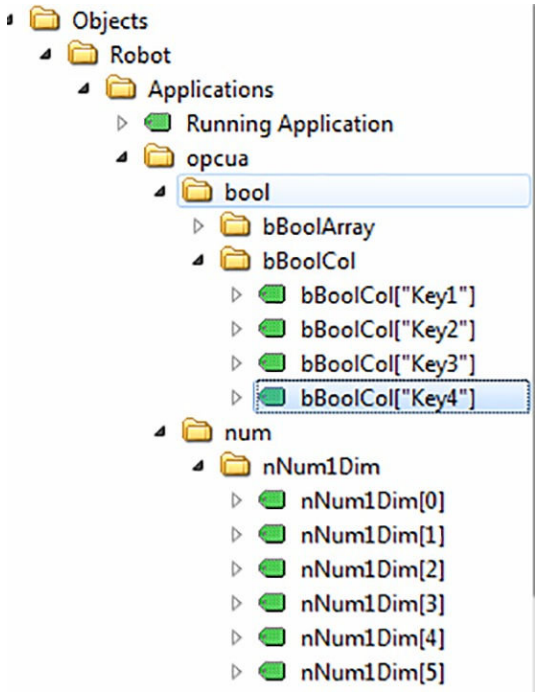
L'utilisateur peut trouver ici toutes les applications chargées. Une application VAL 3 est ouverte par une configuration "autoload/autostart", le panneau VAL 3 du SP2, les librairies des applications VAL 3 ou les instructions VAL 3 LibOpen, LibLoad.



I0005253

Il existe aussi une variable pour voir l'application active. La chaîne de caractères est vide si aucune application n'est active.

Les applications chargées sont visibles en dessous de l'application active. Dans l'image ci-dessus, on voit que l'application "opcua" est chargée. En déroulant le répertoire, le client affiche les variables VAL 3. Les collections et les tableaux sont affichés : si le profil connecté a un writeAccess, il peut modifier la valeur de chaque variable affichée.



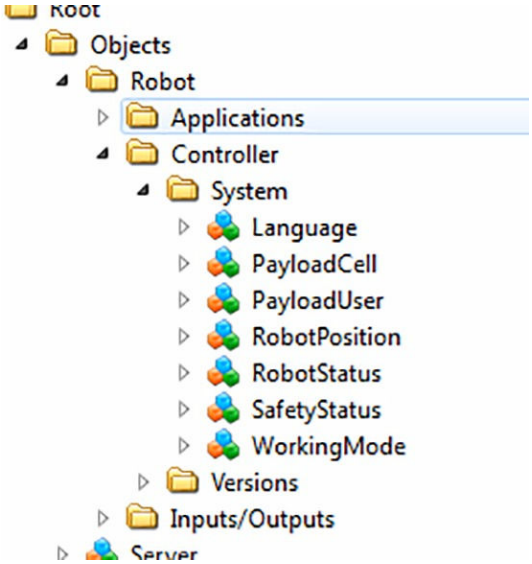
I0005254

Par exemple, l'application OPC UA contient deux variables booléennes et un tableau de num.

14.3.3 - NOEUD DE CONTRÔLEUR

M0005754.1

Le menu Contrôleur comporte également deux répertoires : Système et Versions.



I0005255

14.3.4 - NOEUD SYSTÈME

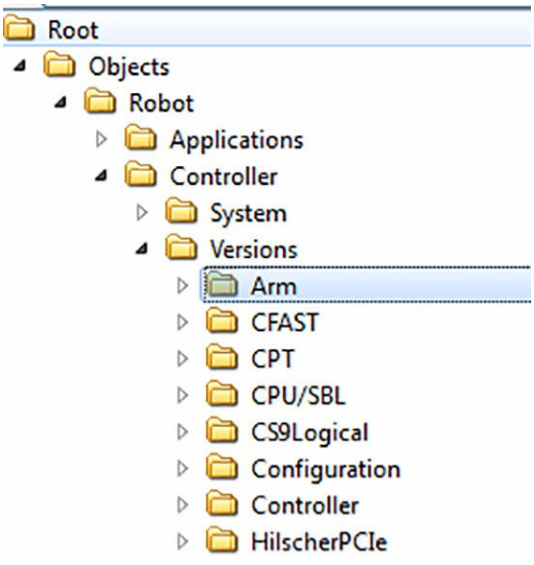
M0005755.1

Dans ce menu, l'utilisateur peut voir certaines informations, comme l'état du robot, sa position, son statut de sécurité, son mode de fonctionnement. Il peut lire et/ou modifier le langage et la charge utile du robot. Ces deux entrées nécessitent un droit sur les exécutables OPC UA, ce qui signifie que les droits du profil CS9 "writeAccess" doivent être réglés sur true.

14.3.5 - NOEUD VERSIONS

M0005756.1

Toutes les versions affichées sur le SP2 dans Robot/Information sont également disponibles dans ce menu. Les versions sont rangées par famille.

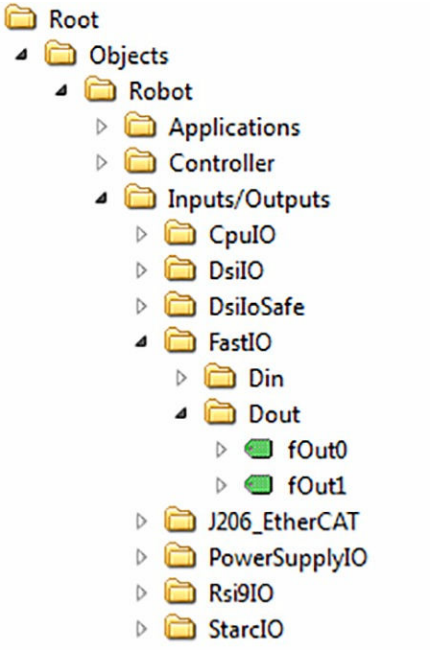


I0005256

14.3.6 - ENTRÉES ET SORTIES

M0005757.1

Toutes les E/S du robot sont affichées ici. L'utilisateur peut écrire sur les sorties qui ne sont pas "system", par exemple la sortie rapide, s'il détient les droits correspondants.



I0005257

14.4 - EXPORTATION DE VARIABLES VAL 3

M0005758.1

14.4.1 - PRINCIPES

M0005759.1

Les variables VAL 3 peuvent être exportées vers le module OPC-UA du contrôleur et être visibles et modifiables par les clients OPC-UA. Pour pouvoir être exportée, une variable VAL 3 doit être globale et du type num, bool ou string.

Une variable peut être exportée en "lecture" ou "lecture et écriture". Quand une donnée est exportée en "écriture", sa valeur peut être changée à partir d'OPC-UA.

La table des données exportées est réinitialisée chaque fois qu'une nouvelle application démarre. Si une application est supprimée (manuellement ou avec VAL 3), ses variables exportées sont effacées de la table.

14.4.2 - INSTRUCTIONS

M0005760.1

void **opcuaExport**(...

M0005761.1

void opcuaExport(num & **nVar**, bool **writable**)

void opcuaExport(bool & **bVar**, bool **writable**)

void opcuaExport(string & **sVar**, bool **writable**)

Fonction

Ces instructions exécutent l'exportation de la variable du premier champ de paramètre. Le deuxième paramètre est utilisé pour choisir s'il faut exporter la donnée VAL 3 avec l'attribut "écriture".

Variables exportables

Les variables exportables doivent être globales et du type num, bool ou string. Les structures d'utilisateur, les structures VAL 3 géométriques (par ex. repère, articulation...), les données externes, les paramètres de fonctions et les données locales ne peuvent pas être exportés. Une erreur d'exécution se produit si des données invalides sont exportées.

void **opcuaExportAll**() M0005762.1

Fonction

Cette instruction exporte toutes les données publiques globales de type num, bool ou string de l'application qui en est propriétaire. Toutes les variables sont exportées en "lecture et écriture". Cette fonction est appelée depuis une librairie afin d'exporter les variables publiques de cette librairie.

void **opcuaReset**() M0005763.1

Fonction

Cette instruction réinitialise toutes les variables précédemment exportées de l'application qui en est propriétaire.

15 - ANNEXES

M0005764.1

15.1 - CODES D'ERREUR D'EXÉCUTION

M0005765.1

Code	Description
-1	Aucune tâche portant le nom spécifié n'a été créée par cette librairie ou application
0	La tâche est suspendue sans erreur d'exécution (instruction taskSuspend() ou mode débogage)
1	La tâche spécifiée est en cours d'exécution
10	Calcul numérique non valide (division par zéro).
11	Calcul numérique non valide (par exemple ln(-1))
20	Accès à un tableau ayant un index supérieur à la taille du tableau.
21	Accès à un tableau ayant un index négatif.
23	Utilisation d'une collection avec une clé non valable. Une clé de collection doit être une valeur de String non nulle.
25	Accès à une collection avec une clé non valable.
29	Nom de tâche invalide. Voir instruction taskCreate().
30	Le nom spécifié ne correspond à aucune tâche VAL 3.
31	Une tâche de même nom existe déjà. Voir instruction taskCreate.
32	Seules 2 périodes différentes pour les tâches synchrones sont supportées. Modifier la période d'exécution.
33	Le nombre maximum de lignes de tâche synchrones VAL 3 a été atteint. Le nombre de lignes de code VAL 3 à exécuter pendant un cycle est limité à 3000 pour les tâches synchrones VAL 3.
34	L'instruction en cours ne peut pas être appelée simultanément par 2 tâches.
35	Une seule tâche à la fois peut accéder au mouvement.
40	Pas assez de place mémoire système.
41	Pas assez de mémoire d'exécution pour la tâche. Voir Taille de mémoire d'exécution.
45	Watchdog a été actualisé par wdgRefresh() mais le système n'a pas été configuré pour l'utiliser : **val3WatchdogPeriod <=0**
60	Temps maximal d'exécution de l'instruction dépassé.
61	Erreur interne à l'interpréteur VAL 3
70	Valeur de paramètre d'instruction invalide. Voir l'instruction correspondante.
71	Opération non valable sur une variable locale.
80	Utilisation d'une donnée ou d'un programme d'une librairie non chargée en mémoire.
81	Cinématique incompatible : Utilisation d'une point/joint/config incompatible avec la cinématique du bras.
82	Le Frame ou Tool de référence d'une variable fait partie d'une librairie qui n'est pas accessible depuis l'application (librairie non déclarée dans l'application de la variable ou variable avec attribut privée).
90	La tâche ne peut pas reprendre à l'endroit spécifié. Voir instruction taskResume().
91	Liaison invalide dans la page de l'utilisateur (voir les détails dans le journal).

Code	Description
92	Option 'userPage' indisponible.
93	Option 'posErrJntFrc' indisponible.
94	Option 'useCryptedLib' indisponible.
95	Option 'OpcUAServer' indisponible.
100	La vitesse spécifiée dans le descripteur de mouvement est invalide (négative ou trop grande).
101	L'accélération spécifiée dans le descripteur de mouvement est invalide (négative ou trop grande).
102	La décélération spécifiée dans le descripteur de mouvement est invalide (négative, trop grande, ou inférieure à la vélocité).
103	La vitesse de translation spécifiée dans le descripteur de mouvement est invalide (négative ou trop grande).
104	La vitesse de rotation spécifiée dans le descripteur de mouvement est invalide (négative ou trop grande).
105	Le paramètre reach spécifié dans le descripteur de mouvement est invalide (négatif).
106	Le paramètre leave spécifié dans le descripteur de mouvement est invalide (négatif).
122	Tentative d'écriture sur une entrée du système.
123	Utilisation d'une entrée-sortie dio, aio ou sio non reliée à une entrée-sortie du système.
125	Erreur de lecture ou d'écriture sur une dio, aio ou sio (erreur sur le bus de terrain)
126	Addon : Format entrée/sortie non valable (float/integer/signed/unsigned) Une variable VAL 3 de type aio a été utilisée avec un format non valable. Par exemple, une variable VAL 3aio avec un format flottant est utilisée avec une instruction VAL 3 alors qu'un nombre entier est attendu.
150	Impossible d'exécuter cette instruction de mouvement : un mouvement demandé précédemment n'a pas pu être exécuté (point hors d'atteinte, singularité, problème de configuration...)
153	Commande de mouvement non supportée
154	Instruction de mouvement invalide : vérifier le descripteur de mouvement.
156	L'identifiant de mouvement se situe en dehors des limites [-80 000, +80 000].
160	Coordonnées du repère flange non valides
161	Coordonnées du repère world non valides
162	Utilisation d'un point sans repère de référence. Voir Définition.
163	Utilisation d'un repère sans repère de référence. Voir Définition.
164	Utilisation d'un outil sans outil de référence. Voir Définition.
165	Repère ou outil de référence invalide (variable globale liée à une variable locale).
170	Addon : Impossible de lier un référentiel mobile à un autre référentiel mobile.
171	Addon : Un référentiel mobile est attendu pour cette instruction.
250	Pas de licence d'exécution (runtime) pour cette instruction, ou validité de la licence démo dépassée.
270	Addon : Chemin de fichier non valable.
271	Addon : Impossible de lire le fichier.
290	Addon : Type de trajectoire non valable.

Code	Description
301	Addon : Position de l'axe externe nécessaire. Cette erreur d'exécution est liée à la fonction de contrôle de l'axe externe.
513	Erreur du générateur de mouvement interne.
514	Valeur de paramètre non valable.
515	État du générateur de mouvement non valable.
516	Mouvement non stabilisé. Certains paramètres de mouvement ne peuvent pas être modifiés pendant un mouvement (par exemple la fréquence de Boxcar).
520	Position d'articulation en dehors des limites logicielles.
525	Position de bras interdite : La position de l'articulation donnée place certains points de contrôle à l'intérieur d'une zone interdite.
530	Aucune solution d'articulation trouvée (pas de convergence). Position cartésienne probablement hors de portée.
531	Position cartésienne hors des limites de l'articulation.
532	Position cartésienne hors espace de travail.
533	Position cartésienne hors d'atteinte avec la configuration spécifiée.
534	Orientation hors d'atteinte avec ce bras.
535	La position du bras identifiée se trouve dans une zone interdite : Les solutions d'articulation identifiées pour atteindre la position cartésienne donnée placent certains points de contrôle à l'intérieur d'une zone interdite.
536	Les positions du bras identifiées sont soit en dehors des limites soit à l'intérieur de zones interdites : Plusieurs solutions d'articulation atteignant la position cartésienne ont été identifiées (différentes configurations) mais aucune n'est à la fois en dehors des zones interdites et à l'intérieur de la plage de l'articulation.
540	Impossible de croiser la singularité dans ce cas.
541	Erreur interne : Discontinuité de la trajectoire cartésienne.
542	Discontinuité de la configuration : La position atteinte par l'articulation ne correspond pas à la configuration du bras nécessaire pour le mouvement suivant.
543	Le robot ne se trouve pas dans la position de départ de la trajectoire.
544	La trajectoire cartésienne entraîne une articulation multitour en dehors de sa plage.
550	Arc circulaire : L'angle entre point de départ et point intermédiaire dépasse 180 degrés.
551	Arc circulaire : L'angle entre point intermédiaire et point final dépasse 180 degrés.
552	Arc circulaire : Les points de départ, intermédiaire et final sont trop proches.
553	Arc circulaire : Le changement d'orientation entre point de départ et point intermédiaire, ou entre point intermédiaire et final est exactement de 180 degrés. La direction du mouvement est indéterminée.
554	lissage cartésien : La rotation est excessive. La direction du mouvement peut être ambigüe.
560	Réservé (Fonction obsolète).
561	Addon : Aucun lissage n'est compatible avec movejSync.
562	Addon : Le lissage de ’joint’ n'est pas pris en charge avec le suivi des mouvements. Utiliser plutôt un lissage ’Cartesian’.
563	Addon : Aucun lissage n'est compatible avec trajMove.
564	Addon : Aucun lissage n'est compatible avec splines.

Code	Description
565	Addon : Lissage non valable dans mdesc : Le lissage 'Cartesian' est impossible en cas de passage entre cinématique { axes externes + bras } et { bras uniquement }.
570	Données non valables pour ce bras (nombre d'articulation erroné ou cinématique erronée).
571	Composant de chaine cinématique non valable.
572	Addon : Erreur de montage d'un composant : Un resetMotion() est nécessaire pour passer en mode trajectoire de composant porté.
573	Addon : Erreur de montage d'un composant : Un resetMotion() est nécessaire pour passer au mode trajectoire d'outil porté.
574	Addon : Erreur de montage d'un composant : Le mode composant porté n'est pas compatible avec ce mouvement.
580	Addon : Un mouvement 'trackOn' est nécessaire pour démarrer une séquence de suivi du convoyeur.
581	Addon : Un mouvement 'trackOff' est nécessaire pour terminer une séquence de suivi de convoyeur.
582	Addon : Le référentiel mobile est différent du précédent. Utiliser un mouvement 'trackon' pour basculer entre les référentiels mobiles.
590	Alter: État non valable, commande ignorée.
600	Addon : Générateur de mouvement verrouillé pour libérer un référentiel mobile.
610	Réservé (Fonction obsolète).
611	Addon : Fréquence de Boxcar trop faible.
999	Vérifier l'historique des événements pour obtenir une description de l'erreur.

DANGER
Les étiquettes de code d'erreur d'exécution avec "Addon" ne sont pas stables et peuvent être modifiées ou supprimées dans la version future.

les codes d'erreur d'exécution compris entre 512 et 768 correspondent à des erreurs de générateur de mouvement.

15.2 - ATTRIBUTS D'OBJET GRAPHIQUE DE LA PAGE UTILISATEUR M0005766.1

La description des attributs ci-dessous est fournie dans la notation DOM (nom de propriété dans JavaScript https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Properties_Reference).

Les catégories définies sont les suivantes :

Catégorie	Description
forbidden	Ne doit pas être utilisé et risque d'entraîner un comportement inattendu.
supported	Testé et pris en charge.
unsupported	Non testé, peut fonctionner ou non.
xor_attributes	Pour un composant, un seul de ces attributs maximum doit être lié.
writable_attributes	Les attributs de cette catégorie peuvent être liés avec l'instruction "w"
def	Attribut d'objet graphique par défaut pour l'élément correspondant

Une liste des attributs des objets graphiques est fournie ci-dessous. Cette liste étant liée à la version du SRC, il est recommandé de consulter la liste correspondant à la version de votre SRC dans le fichier : sys/sp2/app/www-2.0/attributesConfig.json.

L'élément "*" dans l'élément représente tous les types d'objets.

```
{
    "elements": {
        "*": {
            "def": "innerHTML",
            "forbidden": [
                "id",
                "data-*",
                "hidden",
                "for",
                "name",
                "style.height",
                "style.left",
                "style.margin",
                "style.padding",
                "style.width",
                "style.top",
                "onabort",
                "onautocomplete",
                "onautocompleteerror",
                "onblur",
                "oncancel",
                "oncanplay",
                "oncanplaythrough",
                "onchange",
                "onclick",
                "onclose",
                "oncontextmenu",
                "oncuechange",
                "ondblclick",
                "ondrag",
                "ondragend",
                "ondragenter",
                "ondragexit",
                "ondragleave",
                "ondragover",
                "ondragstart",
                "ondrop",
                "ondurationchange",
                "onemptied",
                "onended",
                "onerror",
                "onfocus",
                "oninput",
                "oninvalid",
                "onkeydown",
                "onkeypress",
                "onkeyup",
                "onload",
                "onloadeddata",
                "onloadedmetadata",
                "onloadstart",
                "onmousedown",
                "onmouseenter",
                "onmouseleave",
                "onmousemove",
                "onmouseout",
                "onmouseover",
                "onmouseup",
                "onmousewheel",
                "onpause",
                "onplay",
```

```
            "onplaying",
            "onprogress",
            "onratechange",
            "onreset",
            "onresize",
            "onscroll",
            "onseeked",
            "onseeking",
            "onselect",
            "onshow",
            "onsort",
            "onstalled",
            "onsubmit",
            "onsuspend",
            "ontimeupdate",
            "ontoggle",
            "onvolumechange",
            "onwaiting",
        ],
        "supported": [
            "className",
            "dataset.*",
            "style.visibility"


    "a": {
        "def": "href",
        "forbidden": [],
        "supported": [
            "href",
            "innerHTML",
            "style.backgroundColor",
            "style.borderColor",
            "style.borderStyle",
            "style.borderWidth",
            "style.borderRadius",
            "style.color",
            "style.fontFamily",
            "style.fontSize",
            "style.fontStyle",
            "style.fontWeight",
            "style.textAlign",
            "style.textDecoration",
        ]
    },
    "button": {
        "def": "innerHTML",
        "forbidden": [],
        "supported": [
            "disabled",
            "innerHTML",
            "style.backgroundColor",
            "style.borderColor",
            "style.color",
            "style.fontFamily",
            "style.fontSize",
            "style.fontStyle",
            "style.fontWeight",
            "style.fontFamily",
            "style.textAlign",
        ]
    },
```

```
    "hr": {
        "def": "style.borderColor",
        "forbidden": [],
        "supported": [
            "style.backgroundColor"
            "style.borderColor"
            "style.borderWidth"
        ]
    },
    "img": {
        "def": "src",
        "forbidden": [],
        "supported": [
            "style.backgroundColor"
            "style.borderColor"
            "style.borderStyle"
            "style.borderWidth"
            "style.src"
        ]
    },
    "input": {
        "def": "value",
        "forbidden": [],
        "supported": [
            "disabled"
            "min"
            "max"
            "step"
            "style.backgroundColor",
            "style.borderColor",
            "style.borderStyle",
            "style.borderWidth",
            "style.color",
            "style.fontFamily",
            "style.fontSize",
            "style.fontStyle",
            "style.fontWeight",
            "style.textAlign",
            "value",
        ]
    },
    "label": {
        "def": "innerHTML",
        "forbidden": [],
        "supported": [
            "innerHTML",
            "style.backgroundColor",
            "style.borderColor",
            "style.borderStyle",
            "style.borderWidth",
            "style.color",
            "style.fontFamily",
            "style.fontSize",
            "style.fontStyle",
            "style.fontWeight",
            "style.textAlign",
        ]
    },
    "pre": {
        "def": "innerHTML",
        "forbidden": [],
        "supported": ["innerHTML"]
    },
```

```
        "range": {
            "def": "value",
            "forbidden": [],
            "supported": [
                "disabled",
                "min",
                "max",
                "step",
                "value",
            ]
        },
        "select": {
            "def": "value",
            "forbidden": [],
            "supported": [
                "disabled",
                "options",
                "selectedIndex",
                "style.backgroundColor",
                "style.borderColor",
                "style.borderStyle",
                "style.borderWidth",
                "style.color",
                "style.fontFamily",
                "style.fontSize",
                "style.fontStyle",
                "style.textAlign",
                "value",
            ]
        }
    },
    "writable_attributes": ["value"],
    "xor_attributes": {
        "warning": [
            ["selectedIndex", "value"]
        ]
    },
}
```

LIST OF FIGURES

INDEX

J

L

M

N

O

P

R

S

T

U

V

W