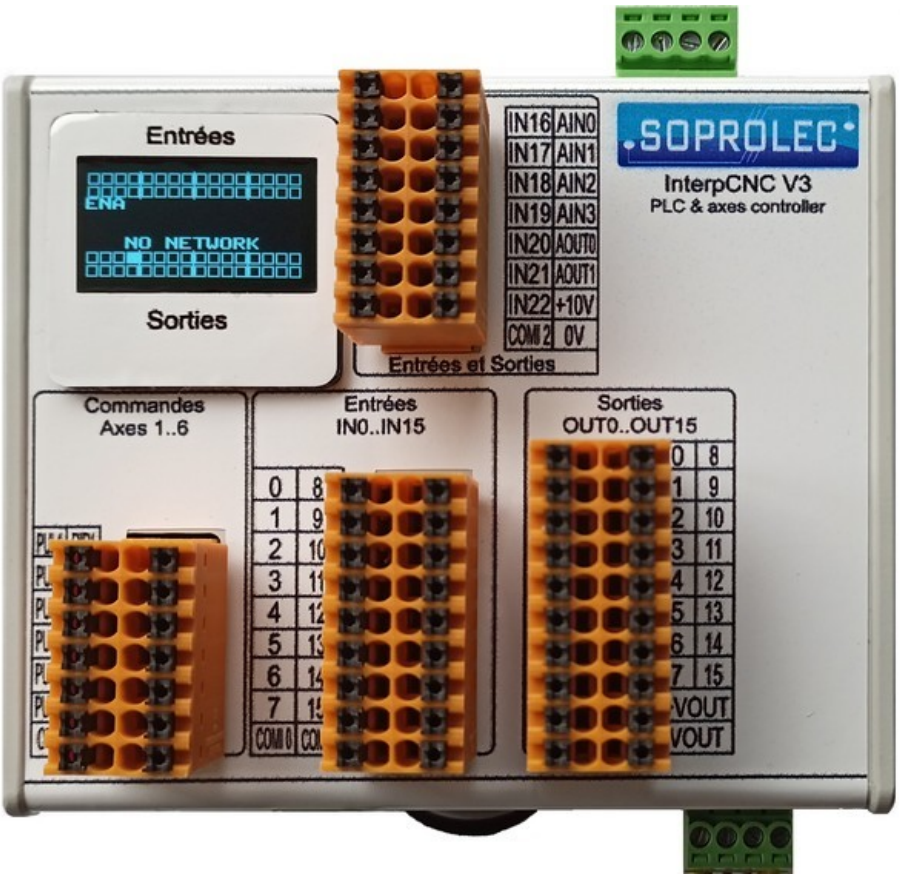
**SOPROLEC**
ZAC DE L'EPINE
72460 SAVIGNE L'EVEQUE
Tél : +33 (0)2 4376 4476

# Carte d'axe
# SOPROLEC
# InterpCNC V3




**Interpréteur langage PLC Basic intégré**

# Table des matières

# InterpCNC V3 – Guide d'utilisation rapide

## Expressions Littérales / Variables Utilisateur

### Expression Littérales

Les chaînes de caractères sont contenus dans des guillemets, ex : "InterpCNC".
Les nombres peuvent être décimaux ou représentés par:
&Hnn Hex Literal, ex : &H3C (60)
&Bnn... Binaire Literal, ex : &B00100011 (35)
n.nE+n Scientifique, e.g. 1.6E+4 (16000)

### Variables Utilisateur

Les noms de variables commencent par un caractère alphanumérique ou un underscore et peuvent contenir n'importe quel caractère alpha ou numérique, point (.) et underscore (_); la longueur maximum est de 32 caractères. Les noms de chaînes de caractères sont terminés par le symbole. Les noms des variables numériques ne sont pas terminés par le symbole $.

## Opérateurs

### Arithmétiques

^ * / Exponentiation, Multiplication, Division
MOD \ Modulus (reste), Division entière
+ + - Addition, Concaténation de chaîne, Soustraction

### Logiques

NOT Inverse logique
= <> Égalité, Inégalité
> < Plus grand que, plus petit que
<= or =< Inférieur ou égal à
>= or => Supérieur ou égal à
AND OR Conjonction, Disjonction
XOR Ou Exclusif

## Formatage de chaînes

**% [flags] [width] [.prec] type**
**flags**: **-** Justifie à gauche
**0** Utilise 0 comme caractère de décalage (pas Espace).
**+** Le signe + désigne des valeurs positives.
**space** Espace comme signe, sauf si négatif.

**width**: nombre minimal de caractères en sortie, moins cause du décalage, plus cause de l'expansion.
**.prec**: nombre de chiffres de fraction pour les types e, ou f, ou le max. de chiffres significatifs pour le type g. Doit être précédé par un point(.) si utilisé.
**Type**: **g ou G** format pour la meilleure présentation.
**f ou F** Format décimal avec point décimal et chiffres
**e ou E** Format exponentiel

## Commandes / Déclarations

Déclaration de tableau :
DIM variable(éléments…)

### Contrôle d'exécution

CONTINUE
DO <déclarations> LOOP
DO WHILE expression <déclarations> LOOP
DO <déclarations> LOOP UNTIL expression
ELSE
ELSEIF expression THEN
ENDIF
END
EXIT
EXIT FOR
FOR compteur = début TO fin [STEP increment]
GOSUB
GOTO
IF expression THEN
IRETURN
NEXT [compteur-variable] [, compteur-variable]...
RUN
STOP
LIST
NEW
SELECT CASE Variable
CASE Value
CASE ValueFrom TO ValueTo
CASE Value IS < Limite
CASE ELSE
END SELECT

### Chaînes / Caractères

ASC (str$)
CHR$ (nbr)
FORMAT$ (nbr [,format$])
INSTR ([start,] search$, pattern$)
LEFT$ (str$, nbr)
LEN (str$)
LCASE$ (str$)
MID$ (str$, start [,nbr])
RIGHT$ (str$, nbr)
SPACE$ (nbr)
SPC (nbr)
STRING$(nbr, val|str$)
TAB(nbr)
UCASE$ (str$)
VAL (str$)
INKEY$

## Fonctions

### Maths / Nombres

ABS (nbr)
ATN (nbr)
CINT (nbr)
COS (nbr)
EXP (nbr)
FIX (nbr)
HEX$ (nbr)
INT (nbr)
LOG (nbr)
OCT$ (nbr)
RND (nbr)
SGN (nbr)
SIN (nbr)
SQR (nbr)
STR$ (nbr)

# Index des Fonctions et Commandes spécifiques à **InterpCNC V3** (avec hyperlien)

## Fonctions



## Commandes

# Introduction

L'InterpCNC V3 dispose d'un puissant interpréteur de langage PLC Basic intégré. Cet interpréteur permet de développer des applications d'automatisme autonomes ou en association avec une interface homme/machine.
L'InterpCNC V3 communique via une connexion Ethernet (protocole Modbus TCP ou Modbus UDP), ou Serie (RS485, protocole Modbus RTU), ou USB (port Com virtuel, protocole Modbus RTU).

Cet interpréteur fonctionne en parallèle des autres fonctions de la carte.
Il est donc possible d'utiliser l'InterpCNC dans des applications de commandes numériques pilotées par un PC tout en exécutant le programme Basic pour des traitements d'actions particulières (par exemple, pupitre déporté de contrôle manuel de la machine).
La programmation se fait à l'aide du programme ICNCStudio.

# Généralités

D’un point de vue matériel, l’InterpCNC est une carte Automate dont les principales caractéristiques sont :

- Commande de 6 axes (Pulses / direction)
- 16 entrées compatibles NPN et PNP
- 16 sorties PNP 500 mA
- 4 entrées Analogiques 0 à 10V
- 2 sorties Analogiques 0 à 10V
- 7 entrées rapides
- Connectivité Ethernet, USB, et 2 x RS485
- 1 Ecran OLED 0,96"
- Microcontrôleur 32 bits
- Tension d'alimentation : 24V

Elle a pour avantage d’intégrer un interpréteur PLC Basic, ce qui en fait comme pour toutes les cartes SOPROLEC, l’une des plus accessibles en termes de programmation.
Cet interpréteur travaille avec des variables qui peuvent être des chaînes de caractères ou des nombres réels.
Toutes les valeurs numériques sont de type réel codés sur 32 bits simple précision. La valeur maximale est de 17549435e-38 et la valeur minimale est de 3.40282347e+38.
Les nombres entiers pouvant êtres manipulés sans perte de précision doivent être compris dans intervalle de ±16777100.

# Gestion des erreurs

S'agissant d'un langage interprété, les erreurs dans le programme sont détectées en cours de fonctionnement.

Il est donc important de pouvoir mettre en place un gestionnaire d'erreur pour interrompre d'éventuels mouvements lorsqu'une erreur survient.

L'interpréteur PLC Basic réalisera automatiquement un GoTo au label OnError si une erreur d'exécution se présente.

Si ce label n'est pas présent dans le programme, aucun traitement d'erreur particulier ne sera réalisé.

Dans l'exemple qui suit, le programme principal est placé dans une une boucle

DO … LOOP.

Si une erreur de traitement, survient, il y aura un GOTO automatique au label OnError :

Dans le traitement de l'erreur, on arrêt tous les éventuels déplacements en cours et on désactive toutes les sorties. Le programme est ensuite interrompu.

Il est bien entendu possible d'ajouter dans le traitement d'erreur un saut de type GOTO pour retourner au début du programme ou reprendre l'exécution à un label particulier.

```
' Programme principal
Do
   ' Code de l'application
Loop

' Traitement d'erreur
OnError:
   StopAxes &H3F   ' Arrêt de tous les axes
   OUTPort 0, 0  'Mise à 0 des sorties 0 à 7
   OUTPort 1, 0 'Mise à 0 des sorties 8 à 15
```

Vous pouvez donc mettre en place un tel gestionnaire dans votre programme PLC Basic à l'aide du label réservé OnError.

# LES FONCTIONS

Les fonctions se distinguent des commandes par le fait qu'elles retournent une réponse, un résultat.

## *Fonctions PLC Basic spécifiques à l' InterpCNC V3*

**Accès aux Registres et Bits utilisateur en Lecture (Holding Registers et Coils)**

| | |
|---|---|
| **GetMB(**Memo**)**<br>ou<br>**GetMB(**Registre, Bit**)** | Lecture d'un Memo bit(Coil) dans l'espace des Memos (Coils utilisateurs), ou un bit particulier dans le domaine des registres.<br>Exemple :<br>if G**etMB**(*MBB_ONOFF_CONVOYEUR*) then …<br>*'si le bit ON/OFF du convoyeur est à 1, alors…*<br>if **GetMB**(3100, 15) then…<br>*'si le 15ème bit de l'adresse 3100 (U16, ici en Ram) est à 1, alors...*<br>if **GetMB**(3100, 31) then…<br>*'si le 31ème bit de l'adresse 3100(U32, ici en Ram) est à 1, alors...* |
| **GetMW**(Registre) | Lecture d'un registre 16 bit non signé (U16)<br>Exemple :<br>ResolutionAxe2 = **GetMW**(EE_RESO_AXE2) /10<br>*'Affectation dans une variable, de la valeur stockée dans le registre correspondant (U16) en EEprom* |
| **GetMDW**(Registre) | Lecture d'un registre 32 bits non signé (U32)<br>Exemple :<br>CompteurTotal = **GetMDW**(EE_CPT_TOTAL)<br>*'Affectation dans une variable, de la valeur stockée dans le registre correspondant (U32) en EEprom* |
| **GetMI**(Registre) | Lecture d'un registre 16 bits signé (I16)<br>Exemple :<br>Offset = **GetMI**(*RCP_OFFSET_BOUTEILLE*) *ResOrienteur/360 |
| **GetMDI**(Registre) | Lecture d'un registre 32 bits signé (I32)<br>Exemple :<br>Cible = GetMDI(POSITION_SPOT) + Offset |
| **GetMF**(Registre) | Lecture d'un registre 32 bits traité comme un Float (FLOAT)<br>Exemple :<br>ResolConvoyeur = **GetMF**(*EE_RES_CONVOYEUR*)<br>*'Affectation dans une variable, de la valeur stockée dans le registre correspondant (FLOAT) en EEprom* |

**Accès aux Registres et Bits système en Lecture seule (Input Registers et Input Bits)**

| | |
|---|---|
| **StsBit(**BitNo)<br><br>BitNo : Numéro du bit de status de 0 à 359 | Lecture de l'état d'un des bits de registre de la carte.<br>Exemple :<br>If not StsBit(256) then…<br>*'Si l'Axe 1 n'est plus en mouvement, alors...* |
| **GetInputMB(**BitNo) | Lecture de l'état d'un des bits de registre de la carte (Input Bits, de 0 à 359, en lecture seule). Idem StsBit.<br>If not **GetInputMB**(256) then…<br>*'Si l'Axe 1 n'est plus en mouvement...* |
| **GetInputMW**(RegisterNo) | Lecture de l'état d'un des registres de la carte (Input Registers, de 1000 à 1143, en lecture seule).<br>Exemple :<br>FirmVerHigh = **GetInputMW(**1115)<br>FirmVerLow = **GetInputMW(**1116)<br>? « Firmware Version : », FirmVerHigh, " ", FirmVerLow<br>*'Affiche la version du Firmware de la carte dans le Moniteur* |
| **GetPrm**(ParameterNumber)<br><br>ParameterNumber : ID paramètre de 0 à 999 | Lecture de la valeur d'un paramètre de la carte en connaissant son identifiant.<br>? GetPrm (20) *'Affiche dans le moniteur, la valeur du paramètre 20.* |
| **IsEEdataChanged** | Retourne 1 si le contenu de la mémoire utilisateur Sauvegardée a été modifié (NB : l'état de **IsEEdataChanged** est aussitôt remis à zéro après avoir été lu/testé).<br>Exemple :<br>If **IsEEdataChanged** then CalculParametres() |
| **IsRCPChanged** | Retourne 1 si le contenu de la mémoire utilisateur EEProm dédiée aux recettes, a été modifié (NB : l'état de **IsRCPChanged** est aussitôt remis à zéro après avoir été lu/testé).<br>Exemple :<br>If **IsRCPChanged** then ? « Recette modifiée ! » |
| **IsMBPrmChanged** | Retourne 1 si le contenu de la mémoire utilisateur EEProm stockant les paramètres de la carte, a été modifié (NB : l'état de **IsMBPrmChanged** est aussitôt remis à zéro après avoir été lu/testé).<br>Exemple :<br>If **IsMBPrmChanged** then CalculParametres() |

**Variantes pour l'accès aux registres en lecture seule (Input Registers):**

Pour mémoire, les Input Registers sont situés entre les adresses 1000 et 2512, et entre les adresses 11000 et 12220 pour les buffers (voir la documentation Modbus pour la correspondance de ces registres).

L'utilisation des mêmes fonctions de lecture que pour les Holding Registers est possible, à condition d'ajouter 1xxxxx à l'adresse du registre.

Exemples :

| | |
|---|---|
| **GetMW**(Registre+100000) | Lecture d'un Input Register 16 bit non signé (U16)<br>Exemple :<br>FirmVerLow = **GetMW**(101115)<br>FirmVerHigh = **GetMW**(101116)<br>*'Affectation dans une variable, de la valeur stockée dans le registre correspondant (U16)* |
| **GetMDW**(Registre+100000) | Lecture d'un Input Register 32 bits non signé (U32)<br>Exemple :<br>NbFramesDMXRecues = **GetMDW**(101998)<br>*'Affectation dans une variable, de la valeur stockée dans le registre correspondant (U32)* |
| **GetMI**(Registre+100000) | Lecture d'un Input Register 16 bits signé (I16)<br>Exemple :<br>I16Value = **GetMI**(1xxxxx)<br>*'Affectation dans une variable, de la valeur stockée dans le registre correspondant (I16)* |
| **GetMDI**(Registre+100000) | Lecture d'un registre 32 bits signé (I32)<br>Exemple :<br>PositionAxe1 = **GetMDI**(101030)<br>*'Affectation dans une variable, de la valeur stockée dans le registre correspondant (I32)* |
| **GetMF**(Registre+100000) | Lecture d'un Input Register 32 bits traité comme un Float (FLOAT)<br>Exemple :<br>FloatValue = **GetMF**(1xxxxx)<br>*'Affectation dans une variable, de la valeur stockée dans le registre correspondant (FLOAT)* |

NB : **GetInputMW**(1115) est équivalent à **GetMW**(101115)
mais dans un soucis d'uniformité avec les autres commandes, il est préférable d'utiliser **GetMW**(Registre+100000)

## ***Fonctions effectuant des calculs***

| | |
|---|---|
| **Min(**Valeur1, Valeur2) | Retourne la valeur minimale entre les valeurs (ou variables) Valeur1 et Valeur 2.<br>Attention : Valeur 1 et Valeur 2 sont traités comme des entiers.<br>Exemple :<br>TensionMin = **Min**(AinV(1), AinV(2))<br>*'TensionMin relève la plus petite valeur (tension) sur 2 entrées analogiques* |
| **Max(**Valeur1, Valeur2) | Retourne la valeur maximale entre les valeurs (ou variables) Valeur1 et Valeur 2.<br>Attention : Valeur 1 et Valeur 2 sont traités comme des entiers.<br>Exemple :<br>TensionMax = **Max**(Ain(3), Ain(4))<br>*'TensionMax relève la plus grande valeur (points) sur les 2 entrées analogiques* |
| **Limit(**Valeur, Mini, Maxi**)** | Retourne Valeur, bornée entre Mini et Maxi. Très utile pour tester une variable en évitant une succession de « if… then… else... »<br>Exemple :<br>**NewPosition = Limit(PositionActuelle, 10000, 30000)**<br>*'Quelle que soit la valeur de PositionActuelle, NewPosition restera limitée entre 10000 et 30000* |

## ***Fonctions liées aux Mouvements d'Axes***

| | |
|---|---|
| **GetPos**(AxeNumber)<br>AxeNumber : Numéro de l'axe, 1 à 5 | Lecture du compteur de position d'un axe. Le retour est un entier 32 bits signé.<br>Exemple :<br>PositionAxe_1 = **GetPos**(1) |

### ***Fonctions liées aux Timers***

| | |
|---|---|
| **GetTimer**(VariableTimer) | Lecture d'un timer initialisé avec SetTimer. Retourne le temps restant sur une temporisation initialisée avec la commande SetTimer. Lorsque le timer est écoulé, cette fonction retourne 0.<br>Exemple : SetTimer Tempo1, 100<br>if **GetTimer**(Tempo1) then<br>out 1,1<br>endif |
| **Toc**(numéro instance de 0 à 15) | Retourne la période de temps écoulée depuis la commande Tic (en micro-secondes)<br>Exemple :<br>? **Toc(**1**)** |
| **Cyclestat(**numéro instance de 0 à 15**)** | Mise à jour de l'instance de mesure de temps. Sert à mesurer un temps de cycle, valeurs à récupérer aux 3 registres MW définis lors de la commande CyclestatInit.<br>Les temps sont donnés en 1/10 de ms<br>Exemple :<br>**CycleStat**(1)<br>*'Actualise les stats dans les 3 registres désignés* |

## ***Fonctions liées aux Entrées et Sorties***

| | |
|---|---|
| **IN(**InputNumber)<br><br>InputNumber : Numéro d'entrée 0 à 255 | Lecture de l'état d'une entrée. Résultat = 0 ou 1.<br>Exemple :<br>if not **IN**(8) then ? « Pression d'air insuffisante »<br>*'Si l'entrée 8 est à 0, alors print « Pression d'air insuffisante »* |
| **Ain(**CanalNumber)<br><br>CanalNumber : Numéro entrée analogique 0 à 3 | Lecture de l'état d'une entrée analogique en millivolts.<br>Exemple :<br>If **Ain**(1) > 5000 then<br>Out 1, 1<br>else<br>Out 1, 0<br>endif |
| **AinV(**CanalNumber)<br><br>CanalNumber : Numéro entrée analogique 0 à 3 | Lecture de l'état d'une entrée analogique en Volts<br>Exemple :<br>If **AinV**(1) > 3,30 then<br>Out 1, 1<br>else<br>Out 1, 0<br>endif |
| **Out(**SortieNo**)**<br><br>SortieNo = Numéro 0 à 15 de la sortie | Lecture de l'état actuel d'une sortie.<br>Exemple :<br>? **Out(**15**)** 'Affiche dans le moniteur, l'état de la sortie 15 |

## ***Fonctions pour la détection de fronts sur Entrées ou Bits utilisateurs***

| | |
|---|---|
| **DF**(n° ou nom de l'Entrée) | Détection front montant ou descendant d'une entrée.<br>Cette fonction permet de détecter le passage de l'état 0 à l'état 1 entre deux appels à cette fonction.<br>Exemples :<br>if **DF**(2) then SetAlarme()<br>'Si l'entrée 2 change d'état, alors…<br>ou<br>if **DF**(IN_VERROU) then<br>? « Intrusion ! »<br>SetAlarme()<br>endif<br>*'si l'entrée détectant le verrou change d'état, alors on exécute notre fonction d'alarme* |
| **DFM**(Numero ou nom de l'entrée) | Détection front montant ou descendant d'une entrée.<br>Exemple :<br>if **DFM**(CELLULE_OBJET) then ? « Objet détecté » ... |
| **DFD**(Numero ou nom de l'entrée) | Détection front descendant d'une entrée. Cette fonction permet de détecter le passage de l'état 1 à l'état 0 entre deux appels à cette fonction.<br>Exemple :<br>if **DFD**(PRESSOSTAT) then ClearAlarme() |
| **DFMBit(**Numero d'instance(1 à 64), Valeur du bit testé) | Détection d'un front montant sur un bit testé.<br>Exemple :<br>if **DFMBit**(2, GetMB(MBB_ALARME)) then ? « Interruption des cycles sur Alarme » ... |
| **DFDBit(**Numero d'instance(1 à 64), Valeur du bit testé) | Détection d'un front descendant sur un bit testé.<br>Exemple :<br>if **DFDBit**(3, GetMB(MBB_ALARME)) then ? « Acquitement des Alarmes » |

# LES COMMANDES

## *Commandes issues du Basic standard*

| | |
|---|---|
| **Run** | Exécute le programme PLC Basic |
| **Stop** | Arrêt de l'exécution du programme PLC Basic |
| **New** | Efface le programme PLC Basic en Ram |
| **List** | Affiche la liste du programme PLC Basic présent en Ram, dans la fenêtre du moniteur |

## *Commandes de gestion du programme*

| | |
|---|---|
| **New** | Efface le programme PLC Basic en Ram |
| **SaveProgram** | Sauvegarde le programme présent en RAM dans une mémoire Flash |
| **LoadProgram** | Charge le programme présent en mémoire Flash pour le placer dans la RAM. Cette commande est appelée automatiquement à la mise sous tension si le paramètre « Démarrage PLC automatique » est actif. Elle sera alors suivie d'une commande RUN également appelée automatiquement. |

## *Commandes PLC Basic spécifiques à l' InterpCNC V3*

**Accès aux Registres et Bits utilisateur en Ecriture (Holding Registers et Coils)**

| | |
|---|---|
| **SetMB** Memo, Value<br>ou<br>**SetMB** Registre, Bit, Value | Set ou Reset d'un Memo bit (Coil) dans l'espace des Memos (Coils utilisateurs), ou un bit particulier dans le domaine des registres.<br>Exemple :<br>**SetMB** *MBB_ALARME, 1*<br>*'mise à 1 du bit d'alarme*<br>**SetMB** 3100,15,0<br>*'mise à 0 du 16ème bit du registre 3100 (Ram)*<br>**SetMB** 3100,31,1<br>*'mise à 1 du 32ème bit du registre 3100 (Ram)* |

| | |
|---|---|
| **SetMW** Registre, Value | Ecriture dans un registre 16 bits non signé (U16).<br>Exemple :<br>**SetMW** GCYCLE, 10 ‘écrit la valeur 10, à l’adresse nommée GCYCLE |
| **SetMDW** Registre, Value | Ecriture dans un registre 32 bits non signé (U32)<br>Exemple :<br>**SetMDW** *EE_COMPTEUR_TOTAL*, **GetMDW**(*EE_COMPTEUR_TOTAL*)+1<br>‘*incrémentation de 1 du registre EEprom stockant la valeur du compteur total* |
| **SetMI** Registre, Valeur | Ecriture dans un registre 16 bits signé (I16)<br>Exemple :<br>**SetMI** 3010, -32767<br>*‘écrit la valeur -32767, à l’adresse 3010 (ici définie en I16, en Ram)* |
| **SetMDI** Registre, Valeur | Ecriture dans un registre 32 bits signé (I32)<br>Exemple :<br>**SetMI** 3010, -999999<br>*‘écrit la valeur -999999, à l’adresse 3010 (ici définie en I32, en Ram)* |
| **SetMF** Registre, Valeur | Ecriture d'un nombre de type FLOAT dans dans un registre 32 bits (FLOAT)<br>Exemple :<br>**SetMF** 3200, 3.14159<br>*‘écrit la valeur 3.14159, à l’adresse 3200 (ici définie en FLOAT, en Ram)* |
| **IncMDW** Registre[, +/-Valeur de l’incrément =1] | Permet d’incrémenter un registre 32 bits de la valeur indiquée ou de +1 si le second paramètre n’est pas précisé.<br>Exemple 1 : **IncMDW** 3020, 5 ‘ Incrément de 5<br>Exemple 2 : IncMDW 3030 ‘ incrément de 1 |
| **CopyReg** Adresse 1er registre Source (0 à 65535), Adresse Destination(Holding Register de 0 à 65535), Nombre de registres (0 à 65535) | Copie une zone de mémoire contenant le nombre de registres indiqués, vers une adresse de destination.<br>Exemple avec des Holding Registers :<br>**CopyReg** 3000, 3200, 16<br>‘*copie les registres 3000 à 3015, vers les registres 3200 à 3215’*<br>Exemple avec des Input Registers :<br>**CopyReg** 101030, 3000, 12<br>*‘copie les registres de positions des 6 axes (Input Registers 1030 à 1041), vers les adresses 3000 à 3011 (Holding Registers)’* |

## ***Commandes spécifiques au Hardware de la carte InterpCNC 6 axes***

| | |
|---|---|
| **Unlock** | **Déverrouillage** de la carte.<br>Lorsque la carte est verrouillée, les commandes de déplacement ou d'activation des sorties sont inactives. |
| **Lock** | **Verrouillage** de la carte.<br>Lorsque la carte est verrouillée, les commandes de déplacement ou d'activation des sorties sont inactives. |
| **ListFlash** | Affiche la liste du programme PLC Basic présent en mémoire Flash, dans la fenêtre du moniteur |
| **SaveProgram** | Sauve le programme PLC Basic actuellement en Ram, dans la mémoire Flash (même action que le click sur l'icone Carte SD) |
| **LoadProgram** | Charge le programme de l'éditeur PLC Basic dans la Ram. |
| **SetPrm** Numéro, Valeur<br><br>Numéro : ID paramètre de 0 à 999 | Écriture de la valeur d'un paramètre InterpCNC en connaissant son identifiant.<br>Exemple :<br>**SetPrm** 20, 1300 ' Fréquence initiale des mouvements Axe 1, réglée sur 1300hz |

## ***Commandes liées à la gestion de Recettes***

| | |
|---|---|
| **CopyRCP** Recette Source, Recette Cible | Fonction recopiant une Recette, vers un autre emplacement Recette.<br>NB : ***RCP_SIZE*** (à l'adresse 9995 -> nombre de registres alloués pour chaque recette) doit avoir été défini au préalable.<br>Exemple :<br>**CopyRCP** 0,1 *'Recopie la recette 0 dans la recette 1* |
| **InsertRCP** Position Destination, Recette Source | Insère la copie d'une recette à l'emplacement précisé. Les recettes suivantes sont décalées.<br>Exemple :<br>**InsertRCP** 1,4 *'Insère une copie de la recette 4, en position 1* |
| **RemoveRCP** NumeroRecette | Supprime une recette. Les recettes suivantes sont décalées.<br>Exemple :<br>**RemoveRCP** 1 *'Supprime la recette n°1* |

## ***Commandes liées aux Mouvements d'Axes***

| | |
|---|---|
| **SetPos** AxeNumber, Value<br><br>AxeNumber : Numéro de l'axe 1 à 5<br>Value : Valeur position | Écriture du compteur de position d'un axe. Cette fonction ne doit pas être appelée durant la rotation de l'axe. Le paramètre AxeNumber permet d'indiquer l'axe concerné par la commande.<br>Exemple :<br>**SetPos** 3, 1000 *'Écrire 1000 dans le compteur axe Y* |
| **MoveAxe** AxeID, Accel, Vitesse, Decel, Position<br>AxeID : Identifiant de l'Axe (1 à 6)<br>Accel : Accélération en Hz/s<br>Vitesse : Fréquence en Hz des pulses<br>Decel : Décélération en Hz/s<br>Position : Cible, en pas moteur | Met l'axe en mouvement jusqu'à une position cible.<br>Exemple :<br>**MoveAxe** 1, 1500, 10000, 1500, 50936 *'Déplacement X* |
| **MoveSpeed** AxeID, accel ou decel (selon la cible), vitesse (signée) | Rotation jusqu'à une vitesse spécifique (vitesse signée)<br>Exemple :<br>**MoveSpeed** 1, 1500, 10000 |
| **Home** AxeID, InputNumber, InputState, Accel, HighSpeed, Decel, (+/-)MaxStep, LowSpeed, [HomePosition], [MoveStepAfterHome]<br><br>AxeID : Axe à initialiser (1 à 6)<br>InputNumber: Numéro de l'entrée utilisée pour le référencement<br>InputState : Etat de l'entrée lorsque le contact est activé (0 ou 1)<br>Accel : accélération mouvement rapide vers le capteur<br>HighSpeed : Vitesse rapide vers le capteur<br>Decel : décélération mouvement rapide vers le capteur<br>(+/-)MaxStep : donne le sens de déplacement et la course Maxi pour le Homing<br>LowSpeed : vitesse dégagement du capteur<br>[HomePosition] valeur à laquelle est initialisé le compteur de position avant le dégagement<br>[MoveStepAfterHome] position cible pour le dégagement (relative à la home position) | Démarre une séquence de Homing. Plusieurs commandes peuvent être lancées simultanément sur différents axes. Cette fonction est utilisée pour trouver une position d'origine à l'aide d'un capteur placé sur la course d'un axe.<br>Exemple :<br>Home 2, 2, 0, 25, 20000, 25, 1000000, 1000, 0<br>*'Origine axe 2 sur entrée N°2 type NC sur une course maxi de 1000000 pas. Position d'origine initialisée à 0* |

| | |
|---|---|
| **Probe** AxeID, InputNumber, InputState, Accel, Speed, Decel, (+/-)MaxStep<br><br>AxeID : Axe à initialiser (1 à 6)<br>InputNumber: Numéro de l'entrée utilisée pour le référencement<br>InputState : Etat de l'entrée lorsque le contact est activé (0 ou 1)<br>Accel : accélération mouvement vers le capteur<br>Speed : Vitesse vers le capteur<br>Decel : décélération mouvement vers le capteur<br>(+/-)MaxStep : donne le sens de déplacement (valeur signée) et la course Maxi pour le Palpage | Lancement d'un Palpage.<br>Exemple :<br><br>**Probe** 1, 3, 0, 25, 50000, 25, 1000000<br><br>*'Palpage sur l'Axe 1, capteur sur l'entrée 3, de type NC, accélération et décélération de 25kHz, vitesse de 50000 Hz, sur 1000000 de pas Maxi '* |
| **StopAxes** Valeur<br>Valeur : binaire (axe sélectionné=Bit à 1) ou Hexadécimal, ou décimal | Arrêt d'un ou plusieurs axes.<br>Exemple :<br>**StopAxes** &B111111 'Arrêt pour les 6 axes (sélection binaire des axes)<br>**StopAxes** &H3F 'Arrêt pour les 6 axes (valeur 3F en Hexa pour les bits sélectionnés)<br>**StopAxes** 63 'Arrêt pour les 6 axes (valeur décimale pour les bits sélectionnés) |
| **StopAxeID** Valeur | Arrêt d'un axe par son Identifiant (numéro où nom).<br>Exemple :<br>**StopAxesID 2**<br>ou **StopAxesID** AXE_ETIQUETTE |

## ***Commandes liées aux Timers***

| | |
|---|---|
| **Pause** ppp<br><br>ppp = durée en ms | Pause de ppp milliseconde dans l'exécution du programme.<br>Les traitement d'interruption ne sont en revanche pas interrompu durant une pause.<br>Cette fonction peut également être utilisé dans les traitements d'interruption.<br>Exemple :<br>*'activation de la sortie 1 pendant une seconde:*<br>OUT 1,1 'Activation sortie 1<br>PAUSE 1000 'Pause 1000ms<br>OUT 1,0 'Désactivation sortie 1 |
| **SetTimer** NomVariable, Durée_ms | Initialisation d'une variable timer. La durée est exprimée en milliseconde. Commande à utiliser avec la fonction GetTimer.<br>Exemple :<br>**SetTimer** Tempo1, 500 |
| **Timer** | Retourne le temps écoulé (en ms) depuis la mise sous tension de la carte.<br>Exemple :<br>print Timer/1000 ; " secondes écoulées " |
| **Time$** | Retourne une chaîne du temps écoulé depuis la mise sous tension. Format : hh:mn:s,ms<br>Exemple :<br>? « Dernière mise en marche à : », **Time$** |
| **Tic** numéro d'instance de 0 à 15 | Démarre une capture de temps, en micro-secondes.<br>Exemple : **Tic 1** |
| **CycleStatInit** N° d'instance (1 à 5), temps mini, temps actuel, temps maxi | Initialise une fonction mesure de temps. Les valeurs seront stockées dans les 3 registres MW (16bits) indiqués. Les temps sont donnés en 1/10 de ms.<br>Exemple :<br>**CycleStatInit** 1, 3015, 3016, 3017<br>ou<br>**CycleStatInit** 1, TPLC_MIN, TPLC_ACTUEL, TPLC_MAX |

## ***Commandes liées aux Entrées et Sorties***

| | |
|---|---|
| **SetIN** N° Entrée, Etat<br>Entrée : 0 à 255<br>Etat : -1 pas de forçage, forçage à 0, forçage à 1 | Forçage de l'état d'une entrée.<br>Exemple :<br>**SetIN** 3, 1 'forçage à 1 de l'entrée 3 |
| **OUT** SortieNumero, Valeur<br>Numéro : 0 à 95<br>Valeur = 0 ou 1 | Activation/désactivation d'une sortie tout ou rien (OUT 0 à OUT 15).<br>Les sorties OUT 0 à OUT 15 sont des sorties physiques. Les sorties 16 à 95 sont des sorties qui peuvent êtres utilisées via un dispositif externe (interfaces Modbus ou USB)<br>Exemple :<br>**OUT** 4,1 *'Passe à 1 l'état de la sortie 4* |
| **OUTPort** NumPort, Valeur<br>NumPort : 0 à 11<br>Valeur : 0 à 255 | Sert à définir l'état (0 ou 1) sur un port de 8 sorties (8 bits).<br>Exemple :<br>for i = 0 to 11 : **OUTPort** i, 0 : Next i *'Remise à zéro de toutes les sorties* |
| **SetAna** Canal, Valeur<br>Canal = 0 ou 1 (AOUT0 ou AOUT1)<br>Valeur = 0 à 10000 | Définit le niveau d'une sortie analogique en point mV (0 à 10000).<br>Exemple :<br>**SetAna** 1, 5000 *'Sortie analogique AOUT1 à 5V* |
| **SetAnaV** Canal, Tension<br>Canal = 0 ou 1 (AOUT0 ou AOUT1)<br>Tension = 0 à 10V | Définit le niveau d'une sortie analogique en Volt (0 à 10v).<br>Exemple :<br>**SetAnaV** 1, 5.51 *'Sortie analogique AOUT1 à 5,51V* |

# CREATION D'UN GRAFCET AVEC L'INSTRUCTION « Select Case »

L'instruction « **Select Case** » permet aisément la création d'un **Grafcet**, et donne une plus grande lisibilité à votre programme, comparée à une série de « If Etape=10 then… Else… Endif »

**Les instructions associées disponibles sont** :
**Select Case** Variable
**Case** Value
**Case** ValueFrom **to** ValueTo
**Case** Value **is** < Limite
**Case Else**
**End Select**

## *Exemple d'utilisation*

Supposons que nous souhaitions créer un cycle automate de 4 étapes (étape 0, 10, 20, et 30) dans lequel nous allons activer/désactiver la sortie 1 selon l'état de l'entrée 1, puis activer/désactiver la sortie 2 avec une temporisation de 1000ms.
Nous utiliserons par exemple une variable nommée « GCycle1 » , qui prendra tour à tour la valeur d'étape en cours. Celle-ci sera réaffectée en fin de chaque étape, ce qui permettra de passer à la suivante au prochain tour de la boucle DO … LOOP.

```
GCycle1 = 0         ' Initialisation du Cycle à l'étape 0
DO
        Select Case GCycle1
        Case 0
          if IN(1) then
            Out 1,1
            GCycle1 = 10
          Endif

        Case 10
          if not IN(1) then
            Out 1,0
            GCycle1 = 20
          Endif

        Case 20
          Out 2,1
          SetTimer tempo1, 1000
          GCycle1 = 30

        Case 30
          if GetTimer(tempo1)=0 then
            Out 2,0
            GCycle1 = 0
          Endif

        Case Else
          ? "Erreur de programmation"
        End Select
LOOP
```

Ainsi, chaque instruction "Case .." permettra de traiter, pour chaque valeur d'étape du cycle GCycle1, le code à exécuter.

# GESTION DES INTERRUPTIONS

Vous avez la possibilité de programmer trois types d'interruption.

- Les interruptions périodiques
- Les interruptions liées à l'état des entrées
- Les interruptions sur positions d’axes

Le temps de latence pour le traitement d'une interruption est < 5µs.

Vous pouvez mettre en place jusqu'à 32 traitements d'interruption différents (périodiques ou liées aux entrées).

Le traitement d'interruption doit se terminer par l'instruction **iReturn**

Lorsqu'un traitement d'interruption n'a plus lieu d'être, vous pouvez le désactiver en indiquant :

Une période de 0 pour les interruption Periodique,

Un numéro d'entrée = 0 pour une interruption liée aux entrées.

## ***Interruptions périodiques***

Mise en place à l'aide de la commande SetTick.
Un saut au label indiqué sera réalisé suivant la période précisée lors de la mise en place de l'interruption.

**SetTick** InterruptionNo, Période, Label

InterruptionNo : Numéro de l'interruption de 1 à 32
Période : Période d'interruption en milliseconde
Label : Label où trouver le traitement d'interruption.

Exemple d'interruption périodique :

```
SetTick 1, 500, OnInt1 ' Mise en place d'une interruption périodique de 500ms
do
… ' Code de l'application
Loop
OnInt1 :
Out 5, not Out(5)  ' Changer l'état de la sortie 5
iReturn
```

## ***Interruptions Liées à l'état des entrées***

Afin d'alléger l'écriture de l'application et réagir rapidement à un changement d'état d'entrée, vous pouvez mettre en place un traitement d'interruption qui réalisera ce contrôle et exécutera le code souhaité.
Ce type de traitement peut prendre en charge :
• Le changement d'état d'une entrée (passage à 0 ou à 1),
• Le passage de l'état 0 à l'état 1 d'une entrée (Front montant),
• Le passage de l'état 1 à l'état 0 d'une entrée (front descendant).
La mise en place de ce traitement se fait à l'aide de la commande **SetInputInt**.
**SetInputInt** :
Syntaxe : InterruptionNo, EntreeNo, Type, Label

InterruptionNo : Numéro de l'interruption de 1 à 32
EntreeNo : Numéro de l'entrée à surveiller (0 à 255)
Type : Type de contrôle
1 pour contrôler tous les changements d'état (INPUT_INT_DF)
2 pour contrôler le passage de l'état 0 à l'état 1 de l'entrée (INPUT_INT_DFM)
3 pour contrôler le passage de l'état 1 à l'état 0 de l'entrée (INPUT_INT_DFD)
4 pour contrôler tous les changements d'état, 1 seule fois (INPUT_INT_DF_ONESHOT)
5 pour contrôler le passage de l'état 0 à l'état 1 de l'entrée, 1 seule fois (INPUT_INT_DFM_ONESHOT)
6 pour contrôler le passage de l'état 1 à l'état 0 de l'entrée, 1 seule fois (INPUT_INT_DFD_ONESHOT)
Label : Label où trouver le traitement d'interruption.

NB : pour annuler une interruption, il suffit de relancer exactement la même commande SetInputInt, mais en utilisant « -1 » comme numéro de l'entrée.

Exemple : Gestion d'une entrée fin de course

```
' Fin de course de type Normalement Fermé sur l'entrée 8
SetInputInt 1, 8, 3, OnFDC1
do
… 'Code de l'application
loop
' traitement entrée fin de course
OnFDC1:
StopAxeID 1    'Arrêt de l'axe 1
iReturn
```

## ***Interruptions sur positions d' Axe***

**SetCaptureInt** : Interruption sur position axe atteinte. L'interruption est automatiquement effacée. AxeNumber=0 pour annuler l'interruption.
Syntaxe : **SetCaptureInt** IntNumber, AxeNumber, CapturePosition, Label
IntNumber : numéro d'instance 1 à 32
AxeNumber : 1 à 6 (sur Axes motorisés),7 à 9 (sur Entrées codeurs), 10 à 16 (sur Compteurs sur entrées rapides)
CapturePosition : position cible (en pas moteur)
Label : Label de l'interruption
Exemple : **SetCaptureInt** 10, 2, CibleEtiquette, OnPosEtiquette

**SetCaptureIDOnInputInt :** Configuration des captures de position sur interruption entrées rapides (16 à 22). S'utilise avec GetCapturePos(AxeID)
L'intérêt de cette commande est de permettre la capture de positions sur les axes, indépendamment de l'exécution du programme PLC Basic. Ainsi les captures étant prioritaires sur l'exécution du programme PLC Basic, celles-ci ont lieu avec une parfaite régularité sans variation dans les temps de réponse.
Syntaxe : **SetCaptureIDOnInputInt** EntréeNo, AxeID
Exemple : **SetCaptureIDOnInputInt** *IN_CEL_ETIQUETTE, AXE_ETIQUETTE*

NB : Ces interruptions ne fonctionnent pas sur un forçage manuel des entrées, mais seulement sur un signal physique.
L'entrée doit également avoir été configurée en mode IT ou compteur (paramètres 220 à 226).

Une seule entrée peur être affectée à un axe. Toute réaffectation d'un axe sur une entrée, annule et remplace l'interruption précédemment déclarée.

**GetCapturePos(** : Lecture de la position capturée sur un axe pendant l'interruption. (Configurée avec SetCaptureIDOnInputInt).
Syntaxe : **GetCapturePos**(*AxeID*)
Exemple : ? "Position Capturée = ", **GetCapturePos**(*AXE_ORIENTEUR_ENTREE*)

# UTILISATION DES ENTRÉES RAPIDES (16 à 22)

## *En mode Codeur*

Nous pouvons disposer de 3 codeurs car 2 entrées rapides sont requises pour chacun des codeurs.
Canal 0 : entrées 21 et 22. Fréquence jusqu'à 1Mhz
Canal 1 : entrées 19 et 20. Fréquence jusqu'à 50 khz
Canal 2 : entrées 17 et 18. Fréquence jusqu'à 50 khz
Les paramètres 221 à 226 de la carte sont à configurer en Mode 4 (4X, tous les fronts sont pris en compte), ou en Mode 3 (2X, 1 front sur 2 est pris en compte).

**GetEncoder(** : Renvoie la position d'une entrée codeur
Syntaxe : **GetEncoder**(Canal) '*Canal : de 1 à 3.*
Exemple : SetMDW 3018, **GetEncoder**(3)

**SetEncoder**: Affectation d'une valeur à une entrée codeur
Syntaxe : S**etEncoder** Canal, Valeur '*Canal : de 1 à 3.*
Exemple : S**etEncoder** 3, 0 'M*ise à 0 de l'entrée codeur n°3*

## *En mode Compteur*

On peut donc disposer de 7 compteurs. Les paramètres 220 à 226 de la carte sont à configurer :
En mode 0 (standard), le rafraîchissement du tableau stockant l'état des entrées a lieu toutes les millisecondes.
En mode 1 (sur interruption « IT »), le rafraîchissement du tableau stockant l'état des entrées a lieu à chaque interruption y accédant.
En mode 2 (Mode compteur), le rafraîchissement du tableau stockant l'état des entrées a lieu à chaque front (montant ou descendant, selon configuration de la polarité des entrées (paramètre 200 de la carte)).
La période entre 2 évènements peut être lue dans les registres modbus 32 bits (Input registers) 1130 à 1142.

**GetCnt(** : Lecture d'un des compteurs associés aux entrées rapides 16 à 22.
Le résultat est un entier non signé.
Syntaxe : **GetCnt(**CompteurNo)
CompteurNo = 1 à 7
Exemple : SetMDW 3018, **GetCnt**(4) 'Ecrit à l'adresse 3018 la valeur lue au compteur n°4

**SetCnt** : Écriture dans les compteurs des entrées rapides 16 à 22. L'interpCNC dispose de 7 entrées rapides qui peuvent être utilisées comme entrées de comptage.
Syntaxe : **SetCnt** CompteurNo, Valeur
Compteur No = 1 à 7
Valeur = Valeur du compteur
Exemple : **SetCnt** 4, 0 'Remet à 0 le compteur 4

# COMMUNICATION VIA MODBUS AVEC DES DRIVERS OU VARIATEURS

Chacun des port COM1 et COM2 peut être utilisé pour le pilotage en MODBUS, de drivers ou variateurs. Dans ce ce cas, la carte InterpCNC V3 sera « Maître ».
Au préalable il sera nécessaire d'étudier la documentation de votre driver/variateur, pour connaître les adresses Modbus de ses principaux registres à utiliser:

## *Communication de base*

→ Mode, Baud Rate, Bits de données, Parité, Bits de Stop, ID en tant qu'esclave

En principe, il faudra définir ces réglages soit depuis le clavier en façade du variateur, soit à l'aide d'un logiciel de paramétrage et connexion au PC.
D'autre réglages peuvent être nécessaires, tels que mode de fonctionnement des entrées, sens de rotation, etc...

Il faudra bien évidemment saisir des paramètres identiques du côté de la carte InterpCNC V3 (registres 400 à 405 pour l'utilisation du COM1, registres 420 à 425 pour l'utilisation du COM2).
Paramètres supplémentaires :
COM1 : registre 408 (délai mini entre chaque trame)
registre 409 (nombre de tentatives avant retour d'une erreur de communication)

COM2 : registre 426 (délai mini entre chaque trame)
registre 427 (nombre de tentatives avant retour d'une erreur de communication)

## *Utilisation des Commandes MBRTU.*

La commande **MBRTU.Send** enverra une trame Modbus via le port COM1 ou 2, destinée à l'ID du variateur (Esclave), avec un type de donnée spécifié (exemple : Holding Register).
La valeur à envoyer sera lue depuis un registre ou bit source (ou plusieurs consécutifs) chez le Maître (la carte InterpCNC V3), pour être écrit(s) vers le registre ou bit spécifié (ou plusieurs consécutifs) de l'esclave.
Ainsi par exemple, on pourra écrire au registre de vitesse du variateur pour faire varier cette dernière.

Structure de la commande **MBRTU.Send** :
(les membres d'une commande peuvent être soit des variables, soit des valeurs directes)

**MBRTU.Send** COMPort, JobType, SlaveID, MasterAddress, SlaveAddress, DataSize, MIRegisterNumber

**COMPort** est le numéro du port COM utilisé (1 ou 2)
**JobType** est le type d'action (Read ou Write) selon le type d'accès de la donnée (Lecture/Ecriture, Lecture seule):

***MB_RTU_WRITE_HOLDING_REG*** *ou «5»*, ***et MB_RTU_READ_HOLDING_REG*** *ou «2»*
***MB_RTU_WRITE_COIL*** *ou «4»*, ***MB_RTU_READ_COIL*** *ou «0»*
***MB_RTU_READ_INPUT*** *ou «1»*, ***MB_RTU_READ_INPUT_REG*** *ou «3»*

**MasterAddress** est l'adresse source chez le Maître
**SlaveAddress** est l'adresse de destination chez l' esclave
**DataSize** est le nombre de données consécutives (registres ou bits)
**MIRegisterNumber** est un registre dans lequel sera retourné un code erreur (-14 = pas d'erreur)

Il est à noter que le mode d'accès aux Input Registers ou Input Bits en ajoutant +100000 à l'adresse, fonctionne également.
Il peut permettre par exemple de les lire directement sur le Maître dans le mode ***MB_RTU_READ_HOLDING_REG*** (au lieu de ***MB_RTU_READ_INPUT_REG***) afin de copier directement la valeur d'un Input Register chez le Maître, vers un Holding Register chez l'esclave. Exemple :

```
const INPUTS_SHADOWS = 3022
const RESULT_COM_VAR5 = 3036

MBRTU.Push 2, MB_RTU_WRITE_HOLDING_REG, 5, 101000, INPUTS_SHADOWS, 2, RESULT_COM_VAR5
```

Cette commande copiera donc via le COM2 , l'état des entrées physiques 0 à 31 de la carte InterpCNC vers les registre 3022 et 3023 de l'esclave. (Le résultat de la communication sera stocké chez le Maître en 3036).

**La commande MBRTU.Push** est identique, à ceci près que les commandes successives se trouveront dans une file d'attente, exécutées dans l'ordre d'arrivée.

**MBRTU.Push** COMPort, JobType, SlaveID, MasterAddress, SlaveAddress, DataSize, MIRegisterNumber

Exemple :
**MBRTU.Push** COMVariateurs, MB_RTU_WRITE_HOLDING_REG, VAR_CONVOYEUR, HZ_TAPIS_CONVOYEUR, REGISTRE_SPEED_VAR, 1, RESULT_COM_VAR5
pourrait aussi s'écrire

```
MBRTU.Push 2, 5, 5, HZ_TAPIS_CONVOYEUR, 3, 1, 3030
```

La commande **MBRTU.Flush(**PortCOM**)** permet de vider la file d'attente des commandes **MBRTU.** non encore traitées. Exemple : **MBRTU.Flush(**1**)**

La fonction **MBRTU.Count(**PortCOM**)** permet de retourner le nombre de commandes actuellement dans la file d'attente.

Exemple :

```
if MBRTU.Count(COMVariateurs)>25 then
   ? "Erreur envoi commande variateur"
endif
```

Elle peut être utilisée par exemple pour détecter un problème de communication. En effet, un nombre de commandes non traitées qui s'accumulent dans la file d'attente, sont révélateurs.

# COMMUNICATION PAR ETHERNET ENTRE CARTES INTERPCNC V3

Chaque carte InterpCNC V3 est équipée d'un port Ethernet, qui lui permet non seulement d'être utilisée à distance par ICNCStudio, mais également de communiquer entre elles.
L'intérêt peut-être par exemple de constituer un réseau de plusieurs postes automates sur une ligne de production, n'ayant besoin que d'un seul programme sur la carte dite « Maître » (ou « serveur »).

En Ethernet, 2 protocoles coexistent : l'UDP et le TCP.
L'UDP est le plus simple à utiliser. Le TCP quant à lui, s'il est plus lourd à mettre en œuvre, est beaucoup plus fiable et plus sécurisé.
De ce fait c'est le TCP qui a été retenu pour la communication entre cartes InterpCNC.
Ainsi nous pourrons accéder aux registres, en lecture ou lecture/écriture, de chacune des cartes connectées, et même leur transmettre des commandes Modbus.

## ***Utilisation, commandes et fonctions TCP***

Voici un programme de démonstration simple :

Commande:

```
1  ' SOPROLEC InterpCNC V3 PLCBasic
2
3  ' Copie état des entrées ICNC vers registre robot adresse 0
4  ' Copie Entrées robot adresse 10000 vers sorties ICNC
5  const ROBOT_OUTPUT_ADDR = 0        ' Adresse pour écriture des sorties sur robot
6  const ROBOT_INPUT_ADDR = 10000     ' Adresse pour lecture des entrées robot
7  const SCAN_RATE=1                  ' Periode de rafraichissement (ms)
8
9  const ICNC_INPUT_ADDR=1000        ' Addresse modbus de l'état des entrées 0..15 de l'ICNC (Input Register)
10 const ICNC_OUTPUT_ADDR=2160       ' Addresse modbus de l'état des sorties OUT0..15 de l'ICNC (Holding register)
11
12 const IP_ROBOT="192.168.10.11"
13
14 ' Pour statistique communication (Adresses de registres internes ICNC)
15 const ETHERNET_RX_FPS= 1198
16 const ETHERNET_TX_FPS= 1199
17 const ETHERNET_TCP_CLIENT_ERROR=1201
18 const ETHERNET_TCP_CLIENT_SUCCESS=1203
19
20 ' Création socket de communication
21 SockRobot = MBClient.Open(IP_ROBOT, 502, MBB_SOCK_BUSY)
22
23 do
24    ' Quand le socket est disponible
25    if not getMB(MBB_SOCK_BUSY) then
26       ' Copy IN0..15 dans holding Register
27       SetMW  REG_ICNC_INPUT_COPY, GetInputMW(ICNC_INPUT_ADDR)
28       ' Envoie entrées ICNC 0..15 sur sorties robot
29       MBClient.WriteVar SockRobot, MB_WRITE_HOLDING_REG_TO_HOLDING_REG, REG_ICNC_INPUT_COPY, 1, ROBOT_OUTPUT_ADDR,0
30       'Copie entrées robot vers sorties 0..15 de ICNC_INPUT_ADDR
31       MBClient.ReadVar SockRobot, MB_READ_HOLDING_REG,ROBOT_INPUT_ADDR, 1,ICNC_OUTPUT_ADDR,SCAN_RATE
32    endif
33
34    ' Pour statistique communication
35    SetMW REG_ETHERNET_RX_FPS, GetInputMW(ETHERNET_RX_FPS)
36    SetMW REG_ETHERNET_TX_FPS, GetInputMW(ETHERNET_TX_FPS)
37    SetMW REG_TCP_CLIENT_ERROR, GetInputMW(ETHERNET_TCP_CLIENT_ERROR)
38    SetMW REG_TCP_CLIENT_SUCCESS , GetInputMW(ETHERNET_TCP_CLIENT_SUCCESS)
39
40 loop
41
```

Nous supposons ici que notre carte Serveur s'adresse à un Robot équipé d'une carte Client dont l'adresse IP est 192.168.10.11

→ `const` ***IP_ROBOT***=*"192.168.10.11"*

Nous définissons tout d'abord en tant que constantes les adresses des zones mémoire où écrire et lire sur la carte Client (`const` ***ROBOT_OUTPUT_ADDR*** = 0 et `const` ***ROBOT_INPUT_ADDR*** = 10000), de même pour la carte Serveur (`const` ***ICNC_INPUT_ADDR***=1000 et `const` ***ICNC_OUTPUT_ADDR***=2160). Les constantes qui suivent ne sont pas indispensables, elles servent ici juste à calculer des statistiques de connexion.

**1) <u>Ouvrir un « Socket »</u>**
Prérequis : choisir un bit utilisateur (coil), qui fera état de la disponibilité du socket pour recevoir les requêtes.
Exemple : MBB_SOCK_BUSY, à l'adresse Modbus 96

`SockRobot = MBClient.Open(`***IP_ROBOT***, 502, ***MBB_SOCK_BUSY***)
La fonction **MBClient.Open** permet d'ouvrir un Socket, et retourne un n° de Socket que l'on stockera dans une variable (ici : SockRobot)
→ paramètres : adresse IP du client, n° port, nom du bit d'Etat

**<u>2) Envoyer des requêtes en lecture ou écriture</u>**
<u>NB</u> : Le Socket est disponible pour recevoir des requêtes en lecture ou écriture, lorsque son bit d'état (ici, MBB_SOCK_BUSY), est à 0.
→ `if not` getMB(***MBB_SOCK_BUSY***) `then`

***REG_ICNC_INPUT_COPY*** est un registre utilisateur 16 bits, disons à l'adresse 3000,
et ***ICNC_INPUT_ADDR*** (adresse 1000 en lecture seule) correspond au mappage sur 16 bits des entrées numériques 0 à 15 de la carte Serveur).
Ainsi :
SetMW ***REG_ICNC_INPUT_COPY***, GetInputMW(***ICNC_INPUT_ADDR***)
va recopier l'état des entrées 0 à 15 du Serveur, à l'adresse 3000.

<u>Requête en écriture</u> :
MBClient.WriteVar SockRobot, ***MB_WRITE_HOLDING_REG_TO_HOLDING_REG***, ***REG_ICNC_INPUT_COPY***, 1, ***ROBOT_OUTPUT_ADDR***, 0

La fonction **MBClient.WriteVar** permet d'envoyer une requête en écriture au Client :
→ paramètres :
adresse IP du client, la commande ***MB_WRITE_HOLDING_REG_TO_HOLDING_REG***, adresse de début à copier (côté Serveur), nombre de registres à copier, adresse de destination (côté Client), temps alloué en ms.

<u>Requête en lecture</u> :
MBClient.ReadVar SockRobot, ***MB_READ_HOLDING_REG***, ***ROBOT_INPUT_ADDR***, 1, ***ICNC_OUTPUT_ADDR***, ***SCAN_RATE***
La fonction **MBClient.ReadVar** permet d'envoyer une requête en lecture au Client :
→ paramètres :
adresse IP du client, la commande ***MB_READ_HOLDING_REG***, adresse de début à copier (côté Client), nombre de registres à copier, adresse de destination (côté Serveur), temps alloué en ms.

NB : On peut empiler jusqu'à 10 requêtes (Read ou Write).
Le Bit d'état  MBB_SOCK_BUSY retombe à zéro après les avoir envoyées.

### 3) Fermer un Socket

On utilise la fonction **MBClient.Close**
→ paramètres : n° de socket
exemple : **MBClient.Close** SockRobot

NB : Les Sockets sont automatiquement fermés lors de l'arrêt du programme automate.

# UTILISATION EN MODE DMX

**Introduction** :
Le DMX est un protocole de communication très répandu dans le monde du spectacle, permettant le pilotage de différents appareils (jeux de lumière, actuateurs motorisés (patiences, tournettes, etc...).

Le DMX 512 existe en mode Device (pour les variateurs de lumières ou des projecteurs) ou en mode Master pour piloter les Devices (exemple : console de lumières).

L'interpCNC vous autorise les 2 modes de fonctionnement :
Le mode Device disponible sur le port COM1
Le mode Master disponible sur le port COM2.

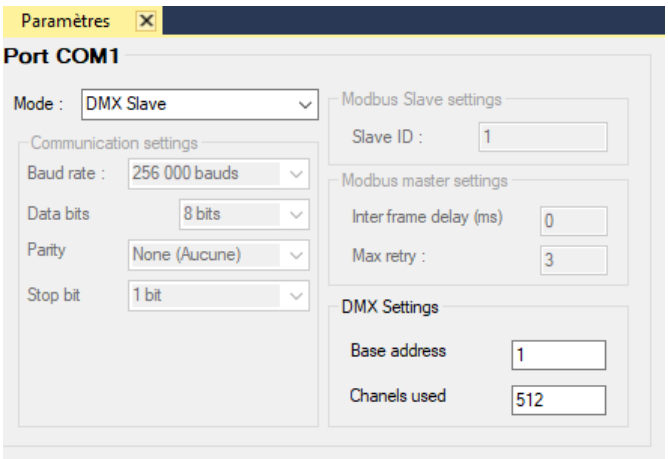
En mode Device, vous pouvez donc piloter les sorties ou des moteurs à partir d'une console DMX.
En mode master, vous pouvez programmer des séquences de contrôle d'appareil en interagissant avec les entrées de l'automate.

## I) Raccordement :

Connecter votre console DMX512 à la carte InterpCNC V3 est très simple. Il suffit de raccorder les signaux D+ (fil vert) et D- (fil jaune) issus de la prise DMX de votre console,
sur les entrées D+ et D- du port COM1 ou COM2 de la carte InterpCNC.

## II) Configuration du mode DMX Device :

Dans le tableau des paramètres, il s'agit de configurer le port COM1 en mode DMX :




L'adresse Base est le n° du 1er canal DMX à partir duquel on veut travailler, et le nombre de canaux utilisés indiquera la plage des canaux utilisés depuis l'Adresse Base.

Le paramètre « Base address » permet de modifier l'adresse DMX de l'automate sans renuméroter les canaux utilisés dans le programme automate.

**III) Utilisation du DMX Device** dans votre programme :
Les fonctions disponibles sont :

**StsBit**(***STS_DMX_CONNECTED***) → Ce bit de status indique si une liaison DMX est établie (Si à 1).

**IsDMXReceived** → Indique qu'une trame DMX vient d'être reçue (Si à 1). Il est automatiquement effacé après sa lecture,
Exemples :

```
if IsDMXReceived then
        …
endif
```

ou encore

```
SetMB DMXReceived, IsDMXReceived
```

(→ recopie de l'état du bit système, dans un bit utilisateur nommé DMXReceived)

Lorsque ce bit est à 1, c'est le moment de lire les canaux à l'aide des fonctions suivantes :

**GetDMX(**n° canal (1 à 512)) → Récupère la valeur entre 0 et 255 présente sur ce canal.

Remarque : Le numéro de canal réellement exploité dépend du paramètre « Base address ». Si le paramètre vaut 1, la commande GetDMX(1) retourne bien la valeur du premier canal DMX.
Si le paramètre « Base address » vaut 10, la commande GetDMX(1) retournera en réalité la valeur du canal 10.

exemples :

```
vitesse = GetDMX(2)
        ou
if GetDMX(1)>127 then
 Out Sortie1 = 1
else
 Out Sortie1 = 0
endif
```

ou encore

**GetDMX16(**n° du 1er canal (1 à 511)) → Récupère les valeurs de 0 à 255 de 2 canaux consécutifs, et reconstitue une valeur de 0 à 65535
Formule : valeur du 1er canal + valeur du 2e canal*256

**III) Utilisation du DMX Master** dans votre programme :

SetDMX N° canal, ValeurDMX → Ecriture d'une valeur 8 bits sur un canal DMX
SetDMX16 N° canal, ValeurDMX → Ecriture d'une valeur 16 bits sur 2 canaux DMX consécutifs.

SetDMXMaster MasterLevel → Pondération des l'ensemble des canaux DMX.
Exemple :

```
SetDMXMaster 255    ‘ les canaux DMX seront transmis tels que donnés par les commandes SetDMX ou SetDMX16
SetDMXMaster 0      ‘ Les canaux seront envoyés avec une valeur 0 (black out)
SetDMXMaster 127    ‘ Les canaux seront envoyés avec une valeur divisée par 2
```

L'ensemble des canaux et également, la valeur de DMX Master sont accessibles en modbus dans les registres en lecture/écriture des adresses 5000 à 5512.

# UTILISATION DE L'HORLOGE RTC

**Fonctions Real Time Clock (RTC)**

L'interpCNC dispose d'une horloge interne permettant de gérer la date et l'heure. Cette horloge n'est cependant pas sauvegardée lors de la mise hors tension. Il convient donc de l'initialiser avant d'en exploiter les fonctions.
L'initialisation peut se faire par :

- Les commandes modbus 112, 113 et 114 (détaillées dans Modbus InterpCNC)
- Le programme automate à l'aide de la commande RTC
- Automatiquement via un serveur SNTP si l'interpCNC dispose d'un accès internet.

**Synchronisation automatique :**
Pour la mise à jour automatique par STNP, les paramètres 546 et 547 doivent être correctement réglés. Le serveur SNTP utilisé est « sntp.pool.org ».
L'horloge sera alors initialisée en tenant compte de fuseau horaire indiqué dans le paramètre 547 et de l'heure d'été/hiver si le bit b1 du paramètre 547 est actif.
Vous disposez de 2 bits de status qui permettent de déterminer l'état de la synchronisation :
stsBit(STS_RTC_SYNCHRONIZED) qui indique que l'horloge a été reglée,
stsBit(STS_SNTP_CONNECTED) qui indique qu'une connection SNTP est établie.
La synchronisation par serveur SNTP, si elle est activée, est renouvelée automatiquement toutes les heures.

**Commande et fonctions PLCBasic**
Vous disposez de 3 instructions pour exploiter l'horloge RTC. Pour chacune d'entre elles, plusieurs codes de fonctions seront détaillés ci-dessous.

RTC RTC_SubCommandeCode, ... pour les commandes qui ne retournent pas de réponse,
RTC(RTC_SubFunction, ...) pour les fonctions qui retournent une valeur numérique,
RTC$(RTC_SubFunction, …) pour les fonctions qui retournent une chaîne de caractères

Les codes de fonctions sont des constantes pré-définies dans ICNCStudio (System\RTC)

Notez également que plusieurs des fonctions ci-dessous détaillées utilisent des variables de type UnixTime. Il s'agit d'un format interne qui ne peut donc pas être manipulé directement par des fonction arithmétiques (addition, soustraction). Pour les opérations sur ces variables, veuillez utiliser les fonctions proposées.

Les informations gérées par la RTC (date et heure) sont également disponibles dans les registres modbus (Input registers) 1987 à 1995
Vous pouvez donc y accéder depuis le PLC avec la commande GetMW( aux adresses 101987 à 101995

**Détails de la commande RTC :**
Code de commande RTC (Syntaxe : RTC RTC_SubCommand, .....)
RTC RTC_SetTime, hh, mn, ss
=> Réglage manuel de l'heure de l'horloge. Si la synchronisation SNTP est active,
cette opération n'est pas nécessaire car l'heure sera automatiquement synchronisé.

RTC RTC_SetDate, jj, mm, aa
=> Réglage manuel de la date de l'horloge. Si la synchronisation SNTP est active,
cette opération n'est pas nécéssaire car la date sera automatiquement synchronisé.

RTC RTC_SetAlarmeA, hh, mn, s, dayOfMonth_or_DayOfWeek[, d_meaning=0][, mask=0]
=> Programmation de l'alarme A. Le déclenchement de l'alarme A provoque un appel à la sub routine onAlarmA()
qui doit exister dans votre programme.
- dayOfMonth_or_DayOfWeek peut être un jour dans le mois de 1 à 31 si d_meaning = RTC_DAY_OF_MONTH
ou un jour de la semaine de 1 à 7 si d_meaning = RTC_DAY_OF_WEEK
- mask permet une programmation avancée de l'alarme.
bit 0 de mask permet de masquer les secondes de l'heure
bit 1 de mask permet de masquer les minutes de l'heure
bit 2 de mask permet de masquer les heure de l'heure
bit 3 de mask permet de masquer le jour de la date
La mise à 1 d'un bit de masque permet de déclencher l'alarme quelque soit la valeur du champs concerné.
exemple :
mask = &b1000, jour est masqué => l'alarme se déclenchera tous les jours à hh:mn:ss (quelque soit jj)
mask = &b1100, jour et heure sont masqués => Alarme périodique toutes les heures à mn:ss (quelque soit jj,hh)
mask = &b1110, jour, heure et mn sont masqués => Alarme périodique toutes les minutes à ss (quelque soit jj,hh,mn)
mask = &b1111, jour, heure et mn et seconde sont masqués => Alarme périodique toutes les secondes (quelque soit jj,hh,mn,s)

RTC RTC_SetAlarmeA, hh, mn, s, dayOfMonth_or_DayOfWeek[, d_meaning=0][, mask=0]
=> indentique à l'alarme A mais appel à une sub routine appelée onAlarmB

RTC RTC_StopAlarmeA
=> Désactiver l'alarme A
Les alarmes sont automatiquement désactivées lorsque le PLC passe en mode STOP
RTC RTC_StopAlarmeB
=> Désactiver l'alarme B
Les alarmes sont automatiquement désactivées lorsque le PLC passe en mode STOP

RTC RTC_SetAlarmAutime, utime[, mask=0]
=> fonctionnement identique à RTC_SetAlarmeA mais la date et l'heure d'alarme sont donnés par une instance de type UnixTime.

Les instance de type UnixTime sont obtenue avec les fonction RTC(...) et permettent de faire des opération simples sur les temps.

RTC RTC_SetAlarmButime, utime[, mask=0]
=> indentique à l'alarme A mais appel à une sub routine appelée onAlarmB

RTC RTC_CalcSunPosition, utime, latitude, longitude, MF_AZIMUTH, MF_ELEVATION => Calcul la position du soleil à un instant donné. Les résultats sont donnée dans des registres de type Float (numéros de registres donnés dans MF_AZIMUTH et MF_ELEVATION
Exemple :

```
const MY_LATTITUDE = 48.064422
const MY_LONGITUDE = 0.28127
Actualutime = RTC(RTC_GetActualUTime) ‘ Date et heure actuelle
RTC RTC_GetSunPosition, Actualutime, MY_LATTITUDE, MY_LONGITUDE, SUN_AZIMUT, SUN_ELEVATION
```

**Détails de la fonction RTC(**
utime = RTC(**RTC_GetActualUTime**)
=> retourne une instance de type UnixTime représentant la date et l'heure actuels
utime = RTC(**RTC_GetMakeUTime**, d,m,y,h,mn,s)
=> création d'une instance de type UnixTime correspondant à la date et heure indiqués
Delta_s = RTC(**RTC_DiffTime**, utime1, utime2)
=> retourne l'écart de temps en seconde entre deux instances de type UnixTime. Soit, utime2-utime1 en seconde.
sum_utime = RTC(**RTC_AddToUtime**, utime, +/-seconde
=> retourne une instance de type UnixTime correspondant à un décallage temporaire de l'instance utime. Le décallage est exprimé en seconde et peut être positif ou négatif.

DayLen = RTC(**RTC_DayLength**, day, month, year, lon, lat[, phase=0]) => Retourne la durée en heure du jour.

```
sunriseHour = RTC(RTC_SunRise, day, month, year, long, lat[, phase=0][,
do_adjustDST=1]) => Heure de levé du soleil à la longitude et la latitude indiquées
(en heure)
```

sunsetHour = RTC(**RTC_Sunset**, day, month, year, long, lat[, phase=0][, do_adjustDST=1]) => Heure de couché du soleil (en heure) à la longitude et la latitude indiquées.

SunRiseUtime = RTC(**RTC_SunRiseUTime**, UTime, longitude, latitude[, type=RTC_SUNSET_TIME][,do_adjustDST=1]) => Heure de levé du soleil au format UnixTime. Peut être utilisé pour réaliser des calculs sur les heures.

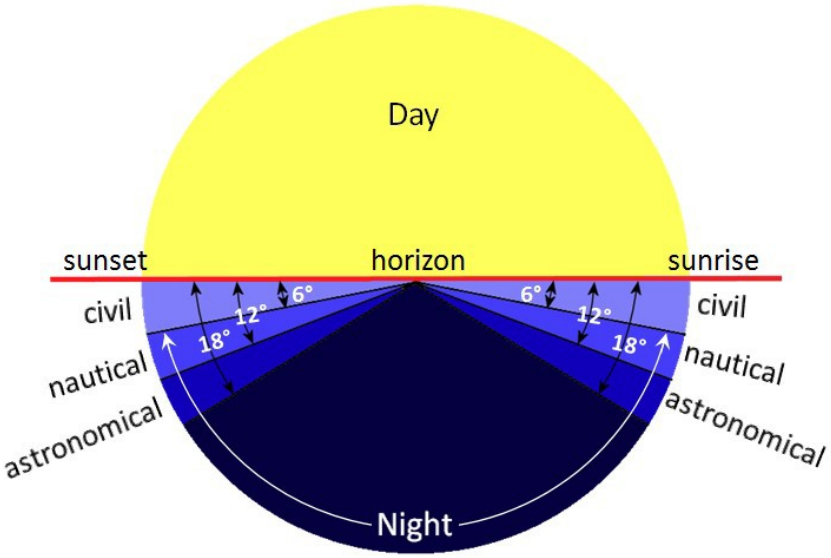
SunSetUTime = RTC(**RTC_SunsetUTime**, UTime, longitude, latitude[, type=RTC_SUNSET_TIME][,do_adjustDST=1]) => Heure de couché du soleil au format UnixTime. Peut être utilisé pour réaliser des calculs sur les heures.

**Détails de la fonction RTC$(**

RTC$(RTC_GetTimeStr)
  => Retourne une chaine de caractère représentant l'heure RTC actuelle au format "hh:mn:ss "
  Exemple : ? RTC$(RTC_GetTimeStr); "Message d'information horodaté"

RTC$(RTC_GetDateStr)
  => Retourne une chaine de caractère représentant l'heure RTC actuelle au format "jj/mm/aa "
  Exemple : ? RTC$(RTC_GetDateStr); "Message d'information horodaté"

RTC$(RTC_GetDateTimeStr)
  => Retourne une chaine de caractère représentant l'heure RTC actuelle au format "jj/mm/aahh:mn:ss "
Exemple : ? RTC$(RTC_GetDateTimeStr); "Message d'information horodaté"



## Historique des modifications

Le 20/10/2022 : Ajout commandes et fonctions RTC

Le 27/06/2022 : Ajout commande DMX master

Le 30/03/2021 : Informations complémentaires sur **SetCaptureIDOnInputInt.**
Ajout Création d'un **Grafcet** avec l'instruction **SELECT CASE**
Ajout des commandes **MBRTU**.Send, .Push, .Count, et .Flush

Le 03/12/2020 : Ajout de la commande **CopyReg** (à partir du firmware 1.19)

Le 01/12/2020 : Ajout des fonctions de Lecture des Input Registers, à l'aide des mêmes fonctions que pour les Holding Registers, mais avec leur **adresse+100000** (à partir du firmware 1.19)

Le 14/10/2020 : Ajout de la commande **Probe**, et de la méthode de désactivation d'une interruption avec **SetInputInt**.

Le 15/09/2020 : Ajout des rubriques pour l'utilisation en **mode DMX** et pour la **communication par Ethernet** entre cartes.

Le 26/03/2020 : Ajout des informations pour l'**utilisation des Recettes** et des **Entrées Rapides** (en mode Codeur et Compteur).

Le 03/03/2020 : Ajout de la commande **SetCaptureInt**.

Le 31/01/2020 : Version initiale.