PC 3000

Fonctions

CE

HA 174 518 - Indice 1

EUROTHERM
AUTOMATION

PREFACE

Ce manuel de référence définit les fonctions disponibles sur la station de programmation PC3000, utilisables dans les instructions en Texte structuré pour définir un 'câblage par soft' entre les blocs fonctions et dans les définitions des étapes et des transitions. Le lecteur doit être familiarisé avec la station de programmation PC3000 et le langage Texte structuré. Le guide utilisateur du PC3000 présente le langage et les concepts de programmation.

Les informations contenues dans ce document donnent des définitions et des exemples pour toutes les fonctions du PC3000 et peuvent servir de référence pour le développement d'applications PC3000.

Pour avoir plus d'informations sur les documents de référence PC3000 :

Manuel du matériel PC3000 : fournit des informations détaillées sur l'ensemble des modules matériels du PC3000, en particulier l'étalonnage, le câblage et les détails de configuration physique. Référence HA174 383.

Manuel du système d'exploitation temps réel PC3000 : décrit le fonctionnement et les fonctions du logiciel du système PC3000 qui gère le chargement, l'initialisation et l'exécution d'un programme utilisateur. Référence HA174 519.

Manuel des blocs fonctions PC3000 : décrit les nombreux blocs fonctions disponibles incorporables dans le programme de régulation pour la régulation PIC, les rampes, les compteurs, les filtres, les timers, etc. Référence HA174 415.

SOMMAIRE

COMPATIBILITÉ MICROLOGICIELLE

	Versions de microprogramme		
Fonctions	2.09	2.27	3.0
ABS_REAL	•	•	•
ABS_DINT	•	•	•
SQRT	•	•	•
LN	•	•	•
LOG	•	•	•
EXP	•	•	•
MODULUS(MOD)	•	•	•
EXPT	•	•	•
SIN	•	•	•
COS	•	•	•
TAN	•	•	•
ASIN	•	•	•
ACOS	•	•	•
ATAN	•	•	•
SEL_BOOL	•	•	•
SEL_DINT	•	•	•
SEL_REAL	•	•	•
SEL_TIME	•	•	•
SEL_DATE	•	•	•
SEL_TIME_OF_DAY	•	•	•
SEL_DATE_AND_TIME	•	•	•
SEL_STRING	•	•	•
MAX_DINT	•	•	•
MAX_REAL	•	•	•
MIN_DINT	•	•	•
MIN_REAL	•	•	•
EQUAL	•	•	•
LEN	•	•	•
LEFT	•	•	•
RIGHT	•	•	•
MID	•	•	•
CONCAT	•	•	•
INSERT	•	•	•
DELETE	•	•	•
REPLACE	•	•	•
FIND	•	•	•
JUSTIFY_LEFT	•	•	•
JUSTIFY_RIGHT	•	•	•
JUSTIFY_CENTRE	•	•	•

	Versions de microprogramme		
Fonctions	2.09	2.27	3.0
DINT_TO_REAL	•	•	•
REAL_TO_DINT	•	•	•
TRUNC	•	•	•
TIME_TO_REAL	•	•	•
REAL_TO_TIME	•	•	•
TIME_TO_UDINT	•	•	•
UDINT_TO_TIME	•	•	•
DATE_AND_TIME_TO_TOD	•	•	•
DATE_TO_TIME_TO_DT	•	•	•
CONCAT_DATE_TOD	•	•	•
DT_TO_UDINT		•	•
UDINT_TO_DT		•	•
STRING_TO_DINT	•	•	•
HEX_STRING_TO_DINT	•	•	•
BIN_STRING_TO_UDINT	•	•	•
OCT_STRING_TO_UDINT	•	•	•
STRING_TO_REAL	•	•	•
STRING_TO_TIME	•	•	•
HMS_STRING_TO_TIME	•	•	•
DHMS_STRING_TO_TIME	•	•	•
STRING_TO_DATE	•	•	•
EURO_STRING_TO_DATE	•	•	•
US_STRING_TO_DATE	•	•	•
STRING_TO_TIME_OF_DAY	•	•	•
DINT_TO_STRING	•	•	•
UDINT_TO_HEX_STRING	•	•	•
UDINT_TO_BIN_STRING	•	•	•
UDINT_TO_OCT_STRING	•	•	•
REAL_TO_STRING	•	•	•
TIME_TO_STRING	•	•	•
TIME_TO_HMS_STRING	•	•	•
TIME_TO_DHMS_STRING	•	•	•
DATE_TO_STRING	•	•	•
DATE_TO_EURO_STRING	•	•	•
DATE_TO_US_STRING	•	•	•
TIME_OF_DAY_TO_STRING	•	•	•
ASCII_TO_CHAR	•	•	•
CHAR_TO_ASCII	•	•	•
ADD_DATE_AND_TIME_T	•	•	•
SUB_DATE_AND_TIME_T	•	•	•
SUB_DATE_AND_TIME	•	•	•

	Versions de microprogramme		
Fonctions	2.09	2.27	3.0
ADD_TOD_TIME	•	•	•
SUB_TOD_TIME	•	•	•
SUB_TOD_TOD	•	•	•
ADD_TIME_TO_TIME		•	•
SUB_TIME_FROM_TIME		•	•
MUL_TIME_BY_REAL		•	•
EXT_BOOL_FROM_STR	•	•	•
REP_BOOL_IN_STR	•	•	•
EXT_SINT_FROM_STR	•	•	•
REP_SINT_IN_STR	•	•	•
EXT_INT_FROM_STR	•	•	•
REP_INT_IN_STR	•	•	•
EXT_DINT_FROM_STR	•	•	•
REP_DINT_IN_STR	•	•	•
EXT_USINT_FROM_STR	•	•	•
REP_USINT_IN_STR	•	•	•
EXT_UINT_FROM_STR	•	•	•
REP_UINT_INSTR	•	•	•
EXT_UDINT_FROM_STR	•	•	•
REP_UDINT_IN_STR	•	•	•
EXT_REAL_FROM_STR	•	•	•
REP_REAL_IN_STR	•	•	•
EXT_TIME_FROM_STR	•	•	•
REP_TIME_IN_STR	•	•	•
EXT_DT_FROM_STR	•	•	•
REP_DT_IN_STR	•	•	•
EXT_INT_FROM_STR_X	•	•	•
IEEE_STRING_TO_REAL	•	•	•
REAL_TO_IEEE_STRING	•	•	•
SET_BIT_IN_DINT			•
GET_BIT_FROM_DINT			•
AND_DINT			•
OR_DINT			•
XOR_DINT			•
NOT_DINT			•

Chapitre 1

INTRODUCTION

Version 1

Sommaire

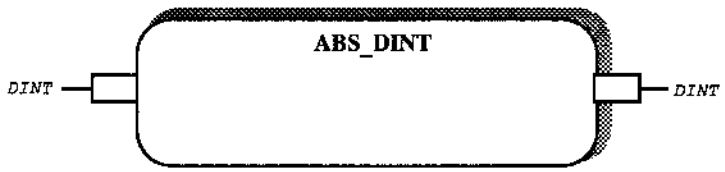
Présentation

Les fonctions Texte structuré (ST) sont exécutées comme faisant partie du corps du Texte structuré dans lequel elles sont appelées. Elles restituent toutes une valeur unique qui est habituellement affectée à un paramètre de bloc fonction ou utilisée dans une expression en ST. Il n'y a pas d'effets indirects, par exemple une modification d'autres paramètres, et la valeur restituée n'est pas stockée, sauf si elle est affectée explicitement à un paramètre donné.

Il est possible d'imbriquer les fonctions ST : ainsi, il est possible d'utiliser les fonctions de conversion de type pour apparier des types de données dans une expression puis de reconvertir le résultat global dans le type de données qui convient. Voir l'annexe E pour avoir des exemples.

Forme générale

Toutes les fonctions du PC3000 sont représentées dans ce texte sous la forme :



Les fonctions ont une seule sortie qui est représentée à droite du schéma.

Les fonctions peuvent avoir n'importe quel nombre d'entrées qui apparaissent à gauche du schéma. Lorsque plusieurs entrées sont utilisées, chacune porte un nom différent, par exemple *G, IN0, IN1*, etc. dans l'encadré de la fonction. Les types de données d'entrée et de sortie, *REAL, BOOL, STRING*, etc. sont énumérés à côté de l'entrée ou de la sortie à laquelle ils se rapportent. Le nom de la fonction est indiqué dans l'encadré de la manière représentée.

> **Remarque :** les fonctions d'entrée simple ont un paramètre d'entrée IN auquel doit être affectée une valeur ou une expression lors de l'appel des fonctions en ST. Toutefois, pour répondre à la norme IEC 1131-3, le nom du paramètre d'entrée simple n'apparaît pas sur les schémas fonctionnels.

Dans les descriptions des paramètres de fonctions, les types de données répondant à la norme IEC 1131-3 figurent entre parenthèses après la description du type de données, par exemple à virgule flottante (REAL).

Restrictions et détection des erreurs

Pour un certain nombre de fonctions, dont une partie des fonctions numériques, il existe des restrictions sur la valeur de la plage de paramètres utilisable. Dans un certain nombre de cas, des valeurs interdites produisent une ou plusieurs erreurs système.

Par exemple, la fonction racine carrée (SQRT) d'un nombre négatif entraîne une erreur système et produit un résultat "IEEE Not a Number" ou NAN. Cette erreur peut se propager par une expression et provoquer un certain nombre d'erreurs système par d'autres opérateurs ou fonctions. Dans certains cas, les fonctions produisent un résultat "IEEE infinity" qui, utilisé dans d'autres expressions, peut à son tour provoquer des erreurs système.

Toutes les erreurs système sont détaillées dans le "Manuel du système d'exploitation temps réel PC3000". Les codes d'erreur applicables aux fonctions ST sont énumérés dans l'annexe C. Il faut prendre des précautions, lors du développement d'un programme utilisateur, pour être sûr de n'utiliser que des valeurs de paramètres de fonctions correctes.

Un certain nombre de fonctions ne produisent des valeurs de sortie correctes que lorsque les paramètres d'entrée se situent dans une plage donnée, c'est-à-dire entre des valeurs maximales et minimales données.

Dans les descriptions de fonctions ci-après, les restrictions des plages de paramètres sont définies ; dans le cas contraire, on peut prendre la plage normale de paramètres.

Attention

1. Un certain nombre de fonctions, en particulier les fonctions de chaînes de caractères, possèdent des paramètres entiers sans signe (c'est-à-dire USINT= entiers courts sans signe et UDINT = entiers doubles sans signe) pour les longueurs, les positions, etc. Il faut veiller à ne pas affecter un paramètre entier, comme un entier double UDINT, avec une valeur négative (<0) à des paramètres qui sont des entiers sans signe. Il se peut que le paramètre entier sans signe suppose qu'il y a une valeur positive très grande.

2. Etant donné que les opérations sur les chaînes de caractères peuvent utiliser une zone importante de l'espace d'empilage du système PC3000, il faut veiller à ne pas imbriquer les fonctions de chaînes de caractères sur plus de 2 niveaux de fonctions. Si une application nécessite une séquence complexe de fonctions de chaînes de caractères, il est préférable d'utiliser des blocs fonctions Chaînes de variables utilisateurs pour stocker les valeurs intermédiaires.

Chapitre 2

FONCTIONS NUMERIQUES

Version 2

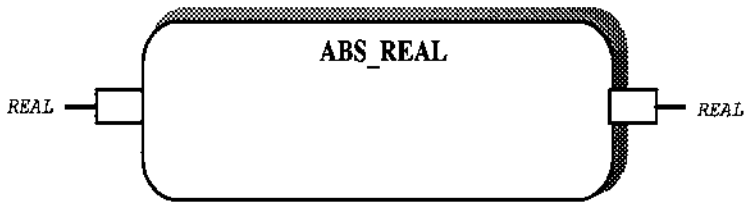
Sommaire

Présentation

Les fonctions numériques comprennent toutes les fonctions mathématiques courantes comme le logarithme, l'exposant, les fonctions trigonométriques sinus, cosinus, etc.

ABS_REAL

Valeur absolue d'une valeur à virgule flottante (REAL), c'est-à-dire que les valeurs négatives sont traitées comme étant positives.



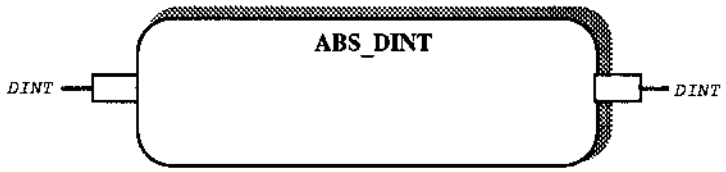
Exemple ST:

```
real1.Val:=  ABS_REAL  (IN:=  -100.3);
```

real1.Val est réglée à 100.3

ABS_DINT

Valeur absolue d'une valeur entière (DINT), c'est-à-dire que les valeurs négatives sont traitées comme étant positives.



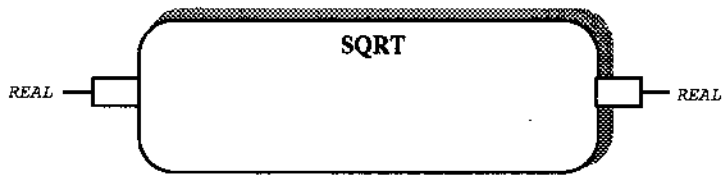
Exemple ST:

```
int1.Val:=  ABS_DINT(IN:=  -100);
```

int1.Val est réglée à 100

SQRT

Racine carrée d'une valeur à virgule flottante (REAL), seules les valeurs positives sont valables.



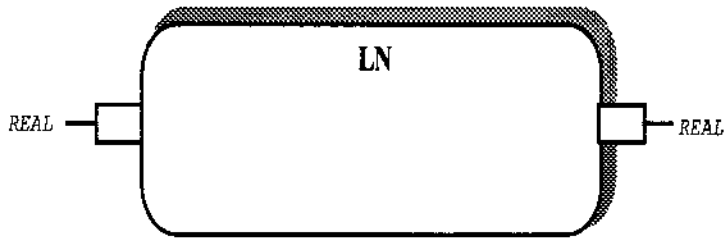
Exemple ST:

```
real1.Val:=  SQRT(IN:=  100.0);
```

real1.Val est réglée à 10.0

LN

Logarithme népérien d'une valeur à virgule flottante, seules les valeurs supérieures à 0 sont valables.



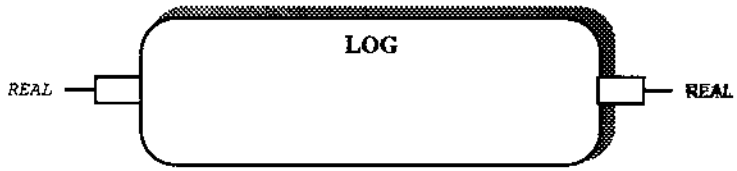
Exemple ST:

```
real1.Val:=  LN(IN:=  2.45);
```

real1.Val est réglée à 0.89609

LOG

Logarithme en base 10 d'une valeur à virgule flottante (REAL) ; seules les valeurs supérieures à 0 sont valables.



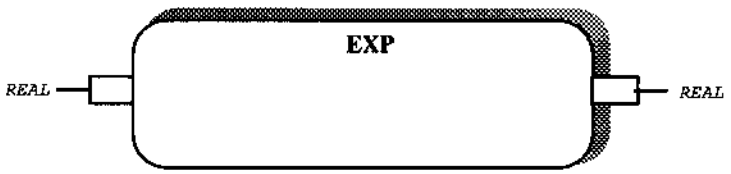
Exemple ST:

```
real1.Val:=  LOG(IN:=  100.0);
```

real1.Val est réglée à 2.0

EXP

Exposant naturel d'une valeur à virgule flottante (REAL), c'est-à-dire e^x ; il est possible d'utiliser à la fois des valeurs positives et des valeurs négatives.



Exemple ST:

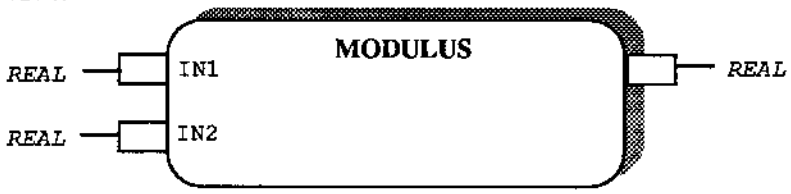
```
real1.Val:=  EXP(IN:=  2.0);
```

real1.Val est réglée à 7.3890

MODULUS

Dans les versions antérieures à 2.27, cette fonction était appelée MOD.

Contrôle Modulo d'une valeur (REAL) à virgule flottante où la valeur du dividende IN1 est divisée par le diviseur IN2 et le reste est retourné. Le diviseur IN2 ne doit pas être 0.



Exemple ST:

```
real1.Val:= MODULUS(IN1:= 13.5, IN2:= 11.0);
```

real1.Val est réglée à 2.5

Exemple ST:

```
real1.Val:= MODULUS(IN1:= 22.0, IN2:= 11.0);
```

real1.Val est réglée à 0

Exemple ST:

```
real1.Val:= MODULUS(IN1:= -3.0, IN2:= 10.0);
```

real1.Val est réglée à -3.0

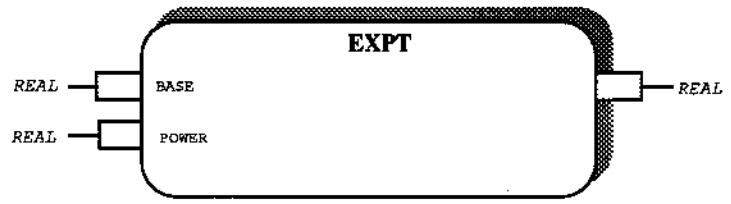
Exemple ST:

```
real1.Val:= MODULUS(IN1:= 11, IN2:= -2.0);
```

real1.Val est réglée à 1.0

EXPT

L'élévation à une puissance élève la valeur de la BASE à une PUISSANCE donnée, c'est-à-dire $BASE^{EXPOSANT}$, ce qui équivaut à $e^{PUISSANCE*ln(BASE)}$. La valeur de la BASE doit toujours être positive.



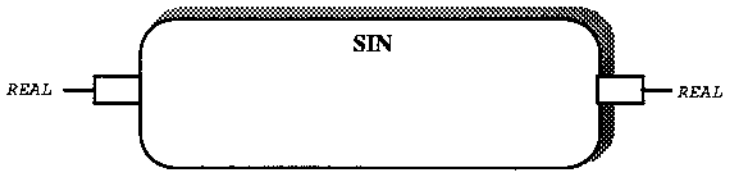
Exemple ST:

```
real1.Val:= EXPT(BASE:= 2.0, EXPOSANT: = 3.0);
```

real1.Val est réglée à 8.0

SIN

Sinus d'une valeur à virgule flottante (REEL) exprimé en radians.



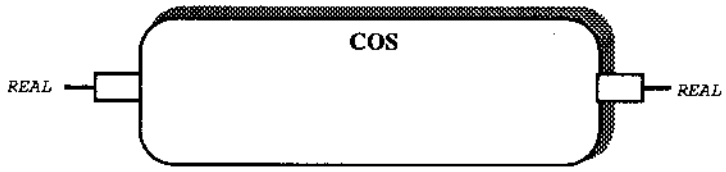
Exemple ST:

```
real1.Val:= SIN(IN:= 0.556);
```

real1.Val est réglée à 0.527

COS

Cosinus d'une valeur à virgule flottante (REAL) exprimé en radians.



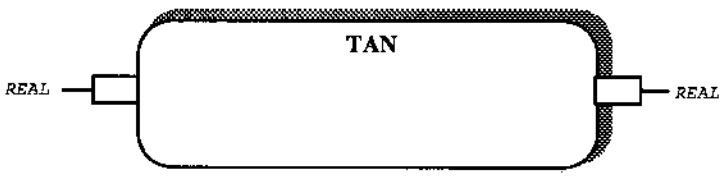
Exemple ST:

```
real1.Val:=  COS(IN:=  0.556);
```

real1.Val est réglée à 0.849

TAN

Tangente d'une valeur à virgule flottante (REAL) exprimée en radians.



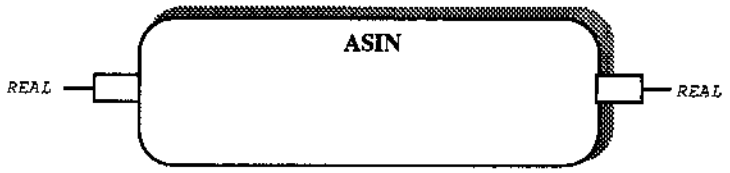
Exemple ST:

```
real1.Val:=  TAN(IN:=  0.71);
```

real1.Val est réglée à 0.859

ASIN

Sinus de l'arc principal, la valeur d'entrée à virgule flottante (REAL) doit être comprise entre -1,0 et +1,0. La sortie est comprise entre - π/2 et π /2.



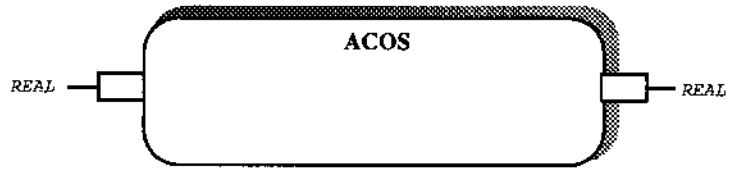
Exemple ST:

```
real1.Val:=  ASIN(IN:=  0.3);
```

real1.Val est réglée à 0.3047

ACOS

Cosinus de l'arc principal, la valeur d'entrée à virgule flottante (REAL) doit être comprise entre -1,0 et +1,0. La sortie est comprise entre 0 et π.



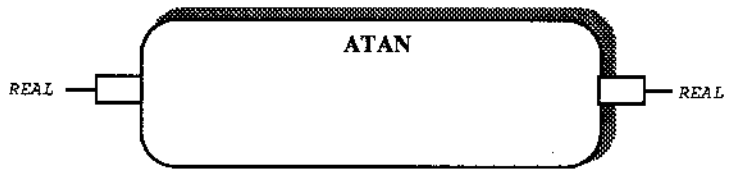
Exemple ST:

```
real1.Val:=  ACOS(IN:=  0.3);
```

real1.Val est réglée à 1.266

ATAN

Tangente de l'arc principal d'une valeur d'entrée à virgule flottante (REAL) ; il n'y a aucune limite pour l'entrée mais la sortie doit être comprise entre - π/2 et π/2.



Exemple ST:

```
real1.Val:=  ATAN(IN:=  20.5);
```

real1.Val est réglée à 1.522

Chapitre 3

FONCTIONS DE SELECTION

Version 1

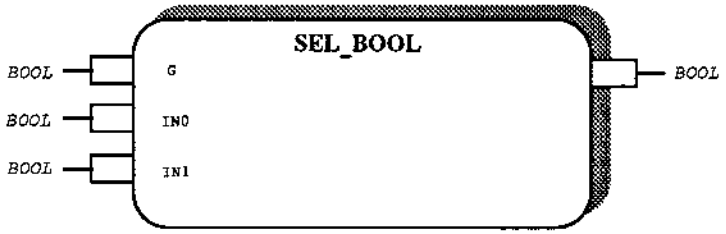
Sommaire

Présentation

Les fonctions de sélection permettent de sélectionner des valeurs en fonction de l'état d'un paramètre ou d'une expression booléen(ne) ou d'autres critères comme le maximum de deux valeurs. Les fonctions de sélection sont utilisables pour des types de données différents, par exemple SEL_REAL sélectionne une valeur à virgule flottante (REAL) parmi deux selon qu'une expression booléenne est vraie ou fausse.

SEL_BOOL

Sélectionne une entrée booléenne (BOOL) parmi deux IN0 et IN1 en fonction de la valeur booléenne de G. Le résultat est IN0 si la valeur de G est 0 (c'est-à-dire FAUSSE) ou IN1 si la valeur de G est 1 (c'est-à-dire VRAIE).



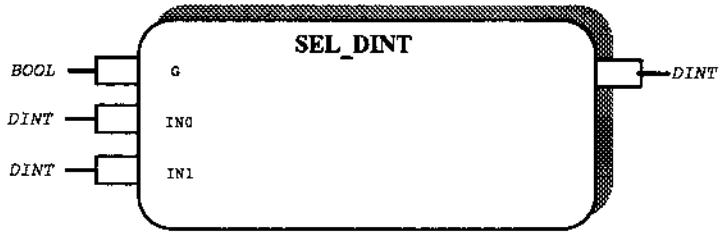
Exemple ST:

```
coolpump.Val:= SEL_BOOL(G:= zone1.Process_Val>100.5,
                        IN0:= 1,
                        IN1:= 0) ;
```

coolpump.Val est réglée à 0 (c'est-à-dire FAUSSE) si zone1.Process_Val est supérieure à 100.5.

SEL_DINT

Sélectionne une entrée entière (DINT) parmi deux IN0 et IN1 en fonction de la valeur booléenne de G. Le résultat est IN0 si la valeur de G est 0 (c'est-à-dire FAUSSE) ou IN1 si la valeur de G est 1 (c'est-à-dire VRAIE)



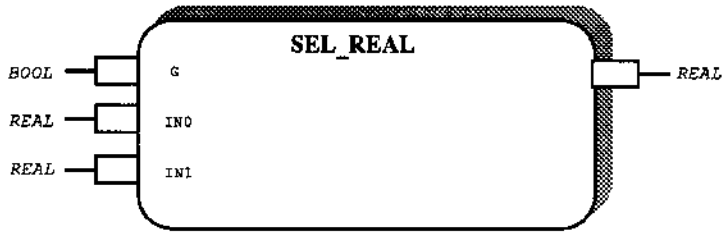
Exemple ST:

```
labelID.Val:=  SEL_DINT(G:=  batchNo.Val  =  1999,
                        IN0:=  batchNo.Val,
                        IN1:=  0);
```

labelID.Val est réglée à 0 lorsque batchNo.Val est égal à 1999 ou à batchNo.Val dans les autres cas.

SEL_REAL

Sélectionne une entrée à virgule flottante (REAL) parmi deux IN0 et IN1 en fonction de la valeur booléenne de G. Le résultat est IN0 si la valeur de G est 0 (c'est-à-dire FAUSSE) ou IN1 si la valeur de G est 1 (c'est-à-dire VRAIE)



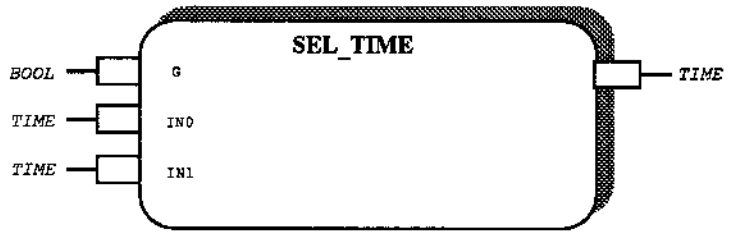
Exemple ST:

```
speed.Val:= SEL_REAL(G:= sensor.Val,
                     IN0 := 10.23
                     IN1 := 23.11 );
```

speed.Val est réglée à 23.11 si sensor.Val est égale à 1 (c'est-à-dire VRAIE) ou à 10.23 dans les autres cas.

SEL_TIME

Sélectionne une entrée de durée (TIME) parmi deux IN0 et IN1 en fonction de la valeur booléenne de G. Le résultat est IN0 si la valeur de G est 0 (c'est-à-dire FAUSSE) ou IN1 si la valeur de G est 1 (c'est-à-dire VRAIE).



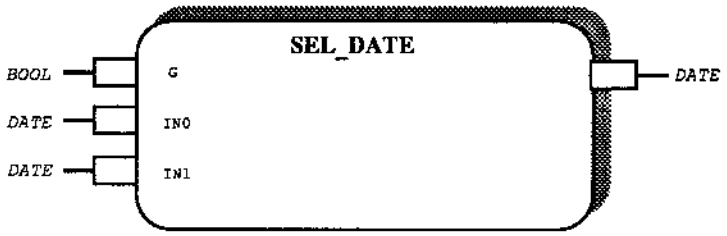
Exemple ST:

```
heatTime.Val:=SEL_TIME(G:=  procType.Val  =  3,
                       IN0:=  time2.Val,
                       IN1:=  T#10s_500ms);
```

heatTime.Val est réglée à T#10s_500ms si procType value est égale à 3 ou à la valeur de time2.Val dans les autres cas.

SEL_DATE

Sélectionne une entrée de date (DATE) parmi deux IN0 et IN1 en fonction de la valeur booléenne de G. Le résultat est IN0 si la valeur de G est 0 (c'est-à-dire FAUSSE) ou IN1 si la valeur de G est 1 (c'est-à-dire VRAIE).



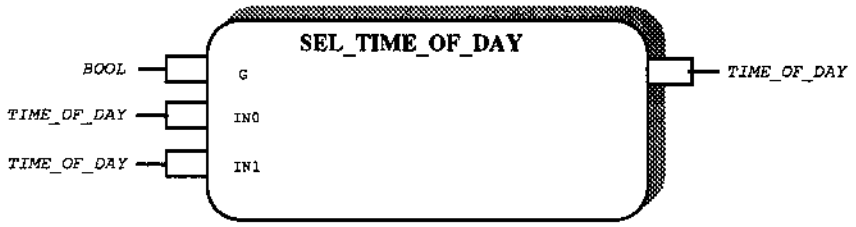
Exemple ST:

```
endDate.Val:= SEL_DATE(G:= jobType.Val = recipe.Val,
                       IN0:= normDate.Val,
                       IN1:= specDate.Val);
```

endDate.Val est réglée à la valeur de specDate si jobType est égale à la valeur de la recette ou endDate.Val est réglée à la valeur de normDate.Val dans les autres cas.

SEL_TIME_OF_DAY

Sélectionne une entrée heure du jour (TIME_OF_DAY) parmi deux IN0 et IN1 en fonction de la valeur booléenne de G. Le résultat est IN0 si la valeur de G est 0 (c'est-à-dire FAUSSE) ou IN1 si la valeur de G est 1 (c'est-à-dire VRAIE).



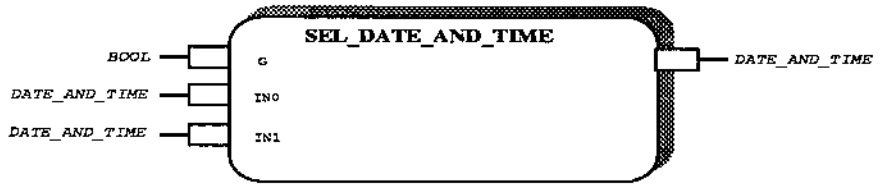
Exemple ST:

```
startime.Val:=
     SEL_TIME_OF_DAY(G:=  RT_Clock.day  =  1(*MON*),
                     IN0:=  TOD#08:00:00,
                     IN1:=  TOD#06:00:00);
```

Si RT_Clock.day est égale à 1 (c'est-à-dire lundi), startime.Val est réglée à 06:00:00 ; elle est réglée à 08:00:00 dans les autres cas.

SEL_DATE_AND_TIME

Sélectionne une valeur date et heure (DATE_AND_TIME) parmi deux IN0 et IN1 en fonction de la valeur booléenne de G. Le résultat est IN0 si la valeur de G est 0 (c'est-à-dire FAUSSE) ou IN1 si la valeur de G est 1 (c'est-à-dire VRAIE).



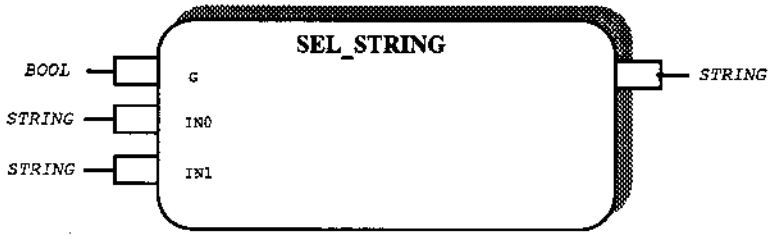
Exemple ST:

```
logDate.Val:=
     SEL_DATE_AND_TIME(G:=logFlag.Val,
                       IN0:=  nullDate.Val,
                       IN1:=  RT_Clock.DateAndTime)  ;
```

Si logFlag.Val est VRAIE, logDate.Val est réglée à RT_Clock.DateAndTime ; elle est réglée à nullDate.Val dans les autres cas.

SEL_STRING

Sélectionne une entrée de chaîne de caractères (STRING) parmi deux IN0 et IN1 en fonction de la valeur booléenne de G. Le résultat est IN0 si la valeur de G est 0 (c'est-à-dire FAUSSE) ou IN1 si la valeur de G est 1 (c'est-à-dire VRAIE).



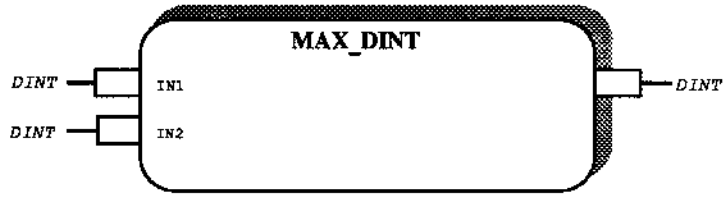
Exemple ST:

```
message.Val:=  SEL_STRING(G:=  ALarm.VAl,
                          IN0:=  'No  Alarm',
                          IN1:=  'ALARM!');
```

Si Alarm.Val est VRAIE, message.Val est réglée à 'ALARM!' ; elle est réglée à 'No Alarm' dans les autres cas.

MAX_DINT

Sélectionne la valeur maximale de deux entrées entières (DINT).



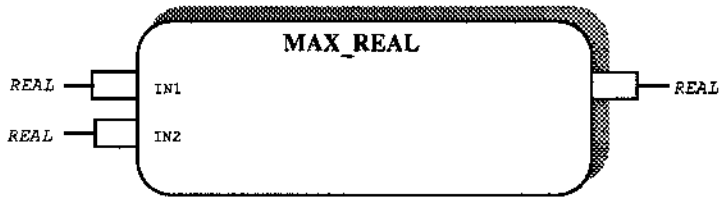
Exemple ST:

```
rate.Val:= MAX_DINT (IN1:= pulseCt.Val, IN2:= 30);
```

rate.Val est réglée à pulseCt.Val lorsque pulseCt est supérieure à 30 ; dans les autres cas, elle est réglée à 30.

MAX_REAL

Sélectionne la valeur maximale de deux entrées à virgule flottante (REAL).



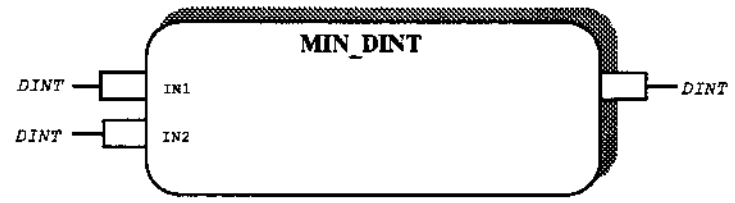
Exemple ST:

```
tempMax.Val:= MAX_REAL(IN1:= zone1.Process_Val,
                       IN2:= MAX_REAL(
                        IN1:= zone2.Process_Val,
                        IN2:= zone3.Process_Val));
```

tempMax.Val est réglée à la valeur maximale des valeurs de process zone1, zone2 et zone3.

MIN_DINT

Sélectionne la valeur minimale de deux entrées entières (DINT).



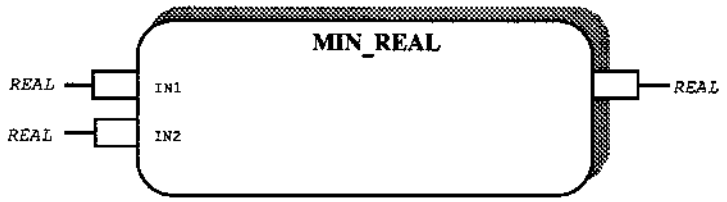
Exemple ST:

```
cntMin.Val:=  MIN_DINT(IN1:=  count1.Val,
                       IN2:=  count2.Val);
```

cntMin.Val est réglée au minimum de count1.Val et count2.Val.

MIN_REAL

Sélectionne la valeur minimale de deux entrées à virgule flottante (REAL).



Exemple ST:

```
     loopSetpoint:=  MAX_REAL(IN1:=  low.Val,
IN2:=  MIN_REAL
(IN1:=  working.Val,
IN2:=  high.val));
```

loop.Setpoint est réglée à working.Val, limitée entre un minimum fixé par low.Val et un maximum fixé par high.Val.

Chapitre 4

FONCTIONS CHAINES

Version 1

Sommaire

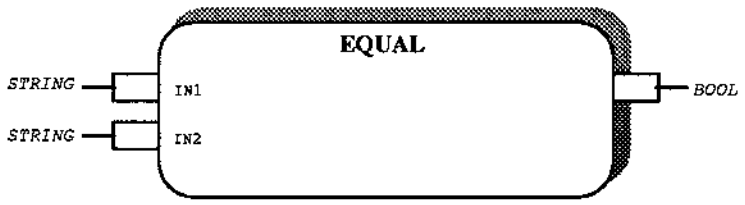
Présentation

Un ensemble très complet de fonctions de chaînes de caractères permet de construire et de manipuler des chaînes de caractères. Il est par exemple possible de construire un message à partir d'un certain nombre de valeurs de paramètres de blocs fonctions possédant des données de types différents.

Normalement, ces fonctions sont nécessaires pour créer des messages de panneaux, des messages aux imprimantes pour produire des rapports, créer et décomposer les messages de demande et de réponse des communications. Dans les fonctions de chaînes ci-dessous, la position des caractères dans une chaîne est considérée comme étant numérotée 1, 2,...L, L étant la longueur totale de la chaîne.

EQUAL

Vérifie l'égalité entre deux chaînes de caractères (STRINGs).



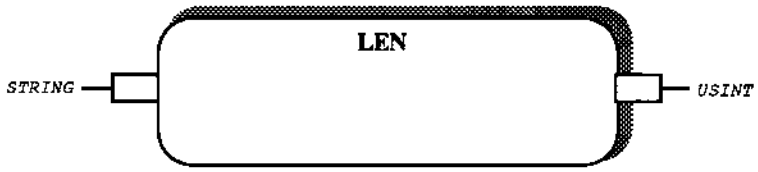
Exemple ST:

```
IF EQUAL (IN1:= ans.Val, IN2:= 'YES') THEN
          response.Val:= 10;
ELSE
          response.Val:=0;
END_IF ;
```

Si la chaîne contenue dans ans.Val est identique à 'YES', response.Val est réglée à 10. Toutefois, pour les autres valeurs d'ans.Val, comme ' YES', 'YEs' ou 'YES ', response.Val est réglée à zéro.

LEN

Restitue la longueur d'une chaîne de caractères (STRING) sous la forme d'un entier court sans signe (USINT).



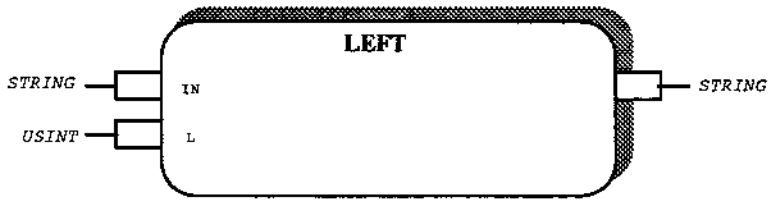
Exemple ST:

```
A.Val:=  LEN('A  STRING');
```

A.Val est réglée à 8.

LEFT

Extrait un nombre donné de caractères L, à partir de l'extrémité gauche d'une chaîne de caractères (STRING) IN, c'est-à-dire à partir du début de la chaîne. Toutefois, si le paramètre IN a une longueur inférieure à L caractères, une chaîne de longueur nulle sera restituée, c'est-à-dire ''.



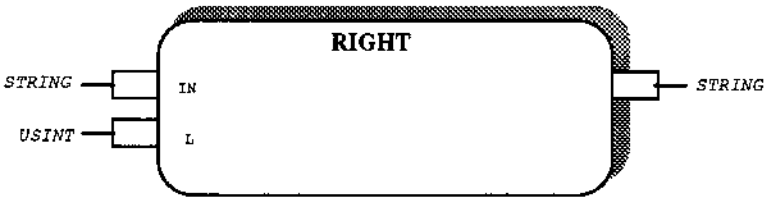
Exemple ST:

```
product.Val:= LEFT(IN:= 'A34F Product', L:= 4);
```

product.Val est réglée à 'A34F'.

RIGHT

Extrait un nombre donné de caractères L, à partir de l'extrémité droite d'une chaîne de caractères (STRING) IN, c'est-à-dire jusqu'à la fin de la chaîne. Toutefois, si le paramètre IN a une longueur inférieure à L caractères, une chaîne de longueur nulle sera restituée, c'est-à-dire ''.



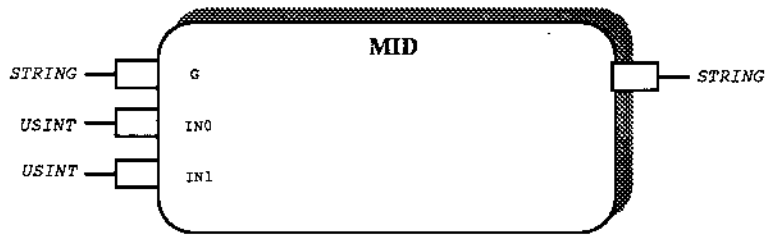
Exemple ST:

```
recipe.Val:= RIGHT(IN:= 'Recipe = Blue', L:= 4);
```

recipe.Val est réglée à 'Blue'.

MID

Extrait une chaîne de caractères (STRING) d'une longueur donnée L à partir d'une position donnée P, dans une chaîne IN. Toutefois, si le paramètre IN a une longueur inférieure à (L+P)-1 caractères ou si P est égal à 0, une chaîne de longueur nulle sera restituée, c'est-à-dire ''.



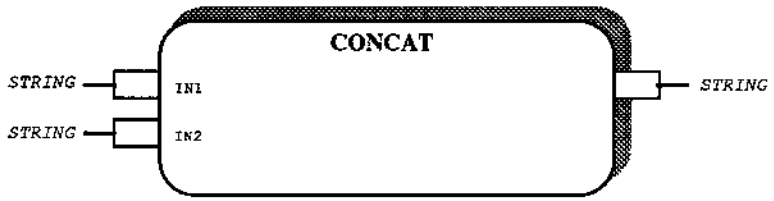
Exemple ST:

```
log.Val:= MID(IN:= 'Valve 12_56 shut',
              L:= 5, P:= 7);
```

log.Val est réglée à '12_56'.

CONCAT

Unit deux chaînes de caractères (STRING) IN1 et IN2, la chaîne IN1 étant située au début de la chaîne restituée.



Exemple ST:

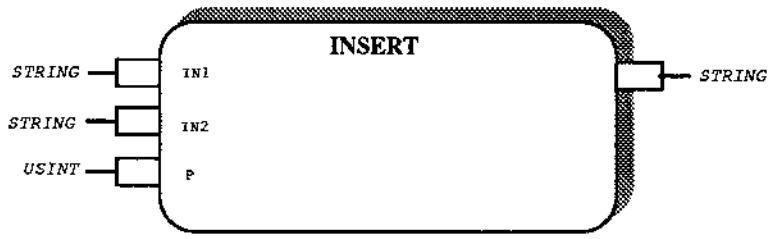
```
report.Val:= CONCAT(IN1:= 'Alarm Status ',
                    IN2:= alarm.Val);
```

Si, pour le paramètre pris comme exemple, alarm.Val est 'OK', report.Val de la chaîne est réglée à 'Alarm Status OK'

INSERT

Insère une chaîne de caractères (STRING) IN2 dans une autre chaîne IN1 en commençant à une position donnée P dans la chaîne IN1. Toutefois, si le paramètre IN1 a une longueur inférieure à P caractères, la fonction restitue une chaîne de longueur nulle, c'est-à-dire ''. Le contenu précédent de la chaîne IN1 est écrasé par la chaîne IN2 insérée.

Si P est égale à zéro, la chaîne IN2 est insérée devant la chaîne IN1.



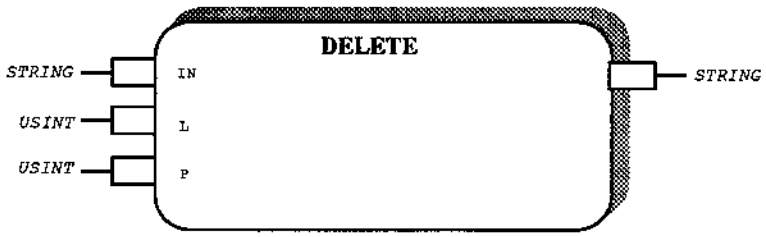
Exemple ST:

```
log.Val:=  INSERT(IN:=  'log:  :'
                  IN2:=  '03',  P:=  5);
```

La chaîne log.Val est réglée à 'log:03:'

DELETE

Supprime une partie d'une chaîne de caractères (STRING) d'une longueur de L caractères d'une chaîne IN en commençant à la position P. Toutefois, si le paramètre IN a une longueur inférieure à (L+P)-1 caractères ou si P est égal à 0, la fonction restitue une chaîne de longueur nulle, c'est-à-dire ''.



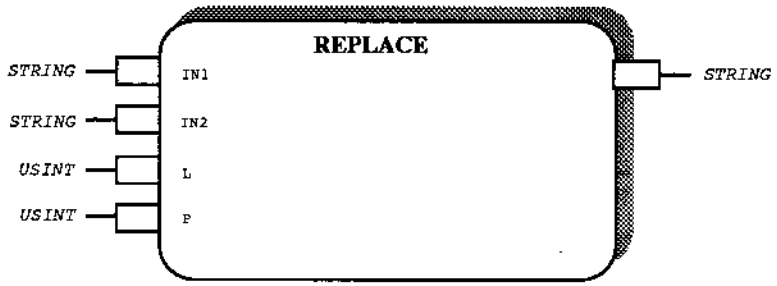
Exemple ST:

```
new.Val:= DELETE(IN:= 'Batch AB Ready'
                 L:= 3, P:= 7);
```

new.Val est réglée à 'Batch Ready'.

REPLACE

Remplace L caractères d'une chaîne de caractères (STRING) IN1 par une autre chaîne IN2 en commençant par la position P dans la chaîne IN1. Toutefois, si le paramètre IN1 a une longueur inférieure à (L+P)-1 caractères ou si P est égal à 0, la fonction restitue une chaîne de longueur nulle, c'est-à-dire ''.



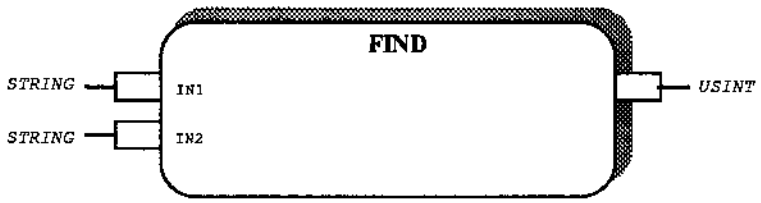
Exemple ST:

```
response.Val:= REPLACE(IN1:= 'Valve 12B Open',
                       IN2:= '5C', L:= 3, P:= 7);
```

response.Val est réglée à 'Valve 5C Open'

FIND

Restitue sous la forme d'un entier sans signe (USINT) la position du début de la première occurrence d'une chaîne IN2 dans une autre chaîne IN1. Si aucune occurrence n'est trouvée, restitue 0. Si la chaîne IN2 est nulle, c'est-à-dire '', restitue 1.



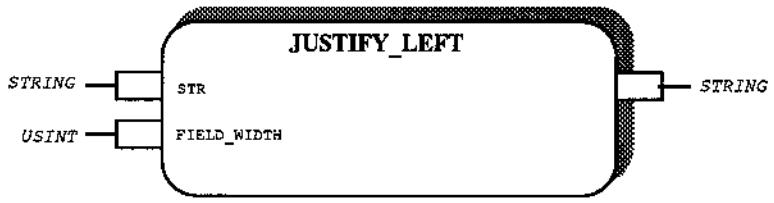
Exemple ST:

```
A.Val:= FIND(IN1:= 'ABCBC', IN2:= 'BC');
```

A.Val est réglée à 2.

JUSTIFY_LEFT

Cette fonction justifie à gauche, c'est-à-dire positionne une chaîne de caractères (STRING) dans un champ spécifié à l'endroit le plus à gauche possible. La chaîne restituée est complétée par des espaces pour occuper toute la largeur du champ. Toutefois, si la longueur de la chaîne STR est supérieure à FIELD_WIDTH, la fonction restitue une chaîne de longueur nulle, c'est-à-dire ''.



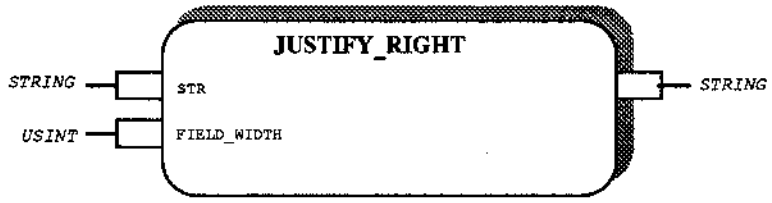
Exemple ST:

```
report.Val:=  JUSTIFY_LEFT(STR:=  'Report:'
                           FIELD_WIDTH:=  12);
```

report.Val est réglée à 'Report:.

JUSTIFY_RIGHT

Cette fonction justifie à droite, c'est-à-dire positionne une chaîne de caractères (STRING) dans un champ spécifié à l'endroit le plus à droite possible. La chaîne restituée est précédée par des espaces pour occuper toute la largeur du champ. Si la longueur de la chaîne STR est supérieure à FIELD_WIDTH, la fonction restitue une chaîne de longueur nulle, c'est-à-dire ''.



Exemple ST:

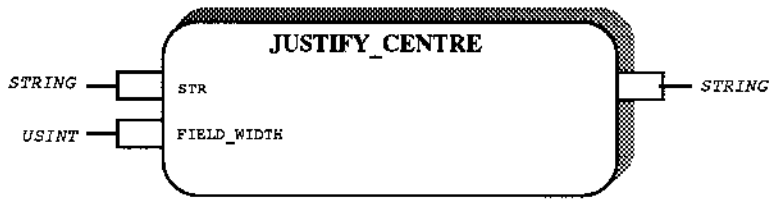
```
report.Val:=  JUSTIFY_RIGHT(STR:=  'Report:',
                            FIELD_WIDTH:=  12);
```

report.Val est réglée à 'Report'.

JUSTIFY_CENTRE

Cette fonction justifie au centre, c'est-à-dire positionne une chaîne de caractères (STRING) dans un champ spécifié à l'endroit central avec des espaces égaux ajoutés aux extrémités gauche et droite de la chaîne. La chaîne restituée est précédée et suivie par des espaces pour occuper toute la largeur du champ. Toutefois, si la longueur de la chaîne STR est supérieure à FIELD_WIDTH, la fonction restitue une chaîne de longueur nulle, c'est-à-dire ''.

S'il n'est pas possible de centrer exactement la chaîne, elle estdécalée sur la gauche.



Exemple ST:

```
banner.Val:=  JUSTIFY_CENTRE(STR:=  'ZONE1',
                             FIELD_WIDTH:=  9);
```

banner.Val est réglée à ' ZONE1 '.

Chapitre 5

FONCTIONS DE CONVERSION DE TYPE

Version 2

Sommaire

Présentation

Ces fonctions permettent de convertir un type de données en un autre type. Il existe une large gamme de fonctions de conversion, par exemple TIME_TO_REAL peut servir à convertir une durée (TIME) en valeur à virgule flottante (REAL) sous forme d'un nombre de secondes.

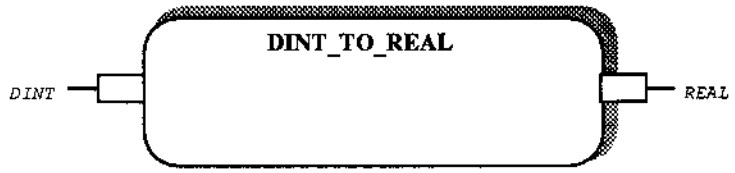
Avec la plupart des fonctions et opérations en Texte structuré, il est indispensable d'utiliser des paramètres possédant le bon type de données. La station de programmation PC3000 émet normalement un message d'erreur en cas de création d'une instruction en Texte structuré comportant un type de données incorrect. Toutefois, il n'y a actuellement aucune vérification relative aux différents types de données entières comme DINT et USINT, voir, les remarques de mise en garde de la page 1-2.

Dans certains cas, il est illogique d'essayer d'utiliser un type de données erroné. Exemple : `a.Val :=b.Val * c.val` n'aurait aucun sens si a.Val, b.Val etc.Val étaient tous des données de type "chaîne de caractères" (STRING). Toutefois, il existe de nombreux cas où il est nécessaire de prendre un paramètre d'un type de données et de l'utiliser dans une expression comportant d'autres types de données. Par exemple, il est souvent nécessaire d'effectuer des calculs avec des paramètres à virgule flottante (REAL) et des paramètres entiers.

Les fonctions de conversion suivantes sont utilisables lorsque la conversion d'un type de données en un autre type a des raisons d'être. L'annexe D fournit des indications supplémentaires.

DINT_TO_REAL

Convertit un nombre entier avec signe (DINT) en un nombre à virgule flottante (REAL).



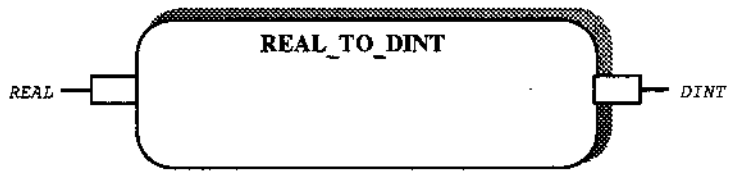
Exemple ST:

```
a.Val:= DINT_TO_REAL(IN:= 12)
```

Fixe a.Val à 12.000.

REAL_TO_DINT

Convertit un nombre à virgule flottante (REAL) en un nombre entier avec signe (DINT). La valeur obtenue est arrondie à l'entier le plus proche.



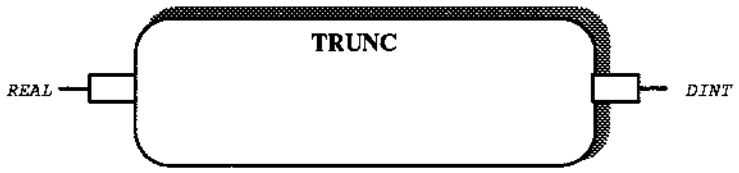
Exemple ST:

```
a.Val:= REAL_TO_DINT (IN:=12.678);
```

Fixe a.Val à 13.

TRUNC

Convertit la partie entière d'un nombre à virgule flottante (REAL) en un nombre entier avec signe (DINT) en tronquant la partie décimale de la valeur.



Exemple ST:

```
a.Val:= TRUNC(IN:= 12.678)
```

Fixe a.Val à 12.

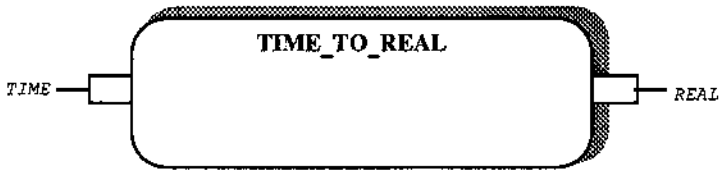
Exemple ST:

```
a.Val:=  TRUNC(IN:=  -6.753)
```

Fixe a.Val à -6.

TIME_TO_REAL

Convertit une durée (TIME) en un nombre de secondes avec virgule flottante (REAL).



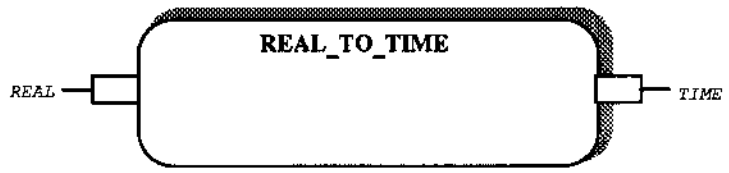
Exemple ST:

```
a.Val:=  TIME_TO_REAL(IN:=  T#10s_500ms);
```

Fixe a.Val à 10.5.

REAL_TO_TIME

Convertit un nombre de secondes avec virgule flottante (REAL) en une durée (TIME). L'entrée IN doit être un nombre positif inférieur à (2^{32}/1000), c'est-à-dire inférieur à 4294967.295 pour que la conversion s'effectue en une durée valable.



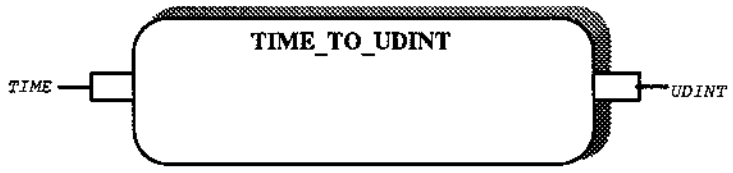
Exemple ST:

```
a.Val:=  REAL_TO_TIME(IN  :=  10.500);
```

Fixe a.Val sur T#10s_500ms.

TIME_TO_UDINT

Convertit une durée (TIME) en nombre entier de millisecondes (UDINT).



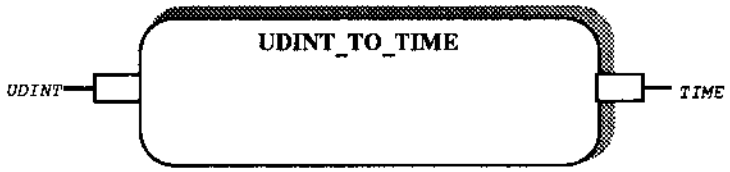
Exemple ST:

```
a.Val:=  TIME_TO_UDINT(IN:=  T#10s_500ms);
```

Fixe a.Val sur 10500.

UDINT_TO_TIME

Convertit un nombre entier (UDINT) de millisecondes en une durée (TIME). L'entrée IN doit être un nombre positif pour donner une durée valable.



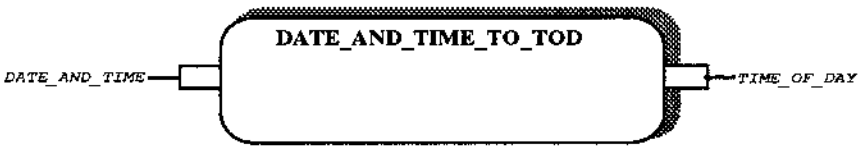
Exemple ST:

```
a.Val:=  UDINT_TO_TIME(IN:=  10500)
```

Fixe a.Val sur T#10s_500ms.

DATE_AND_TIME_TO_TOD

Convertit une date et heure (DATE_AND_TIME) en heure du jour (TIME_OF_DAY) en supprimant la composante date (DATE).



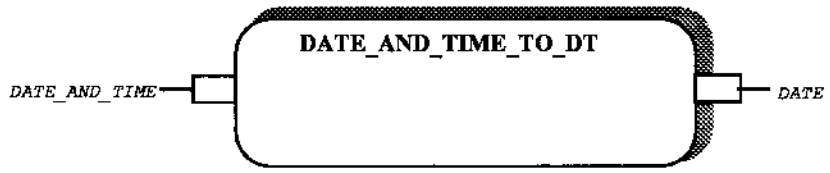
Exemple ST:

```
a.Val:=  DATE_AND_TIME_TO_TOD(
                          IN:=  DT#02-Sep-1991-14:30:00);
```

Fixe a.Val sur TOD#14:30:00.

DATE_AND_TIME_TO_DT

Convertit une date et heure (DATE_AND_TIME) en date (DATE) en supprimant la composante heure du jour (TIME_OF_DAY).



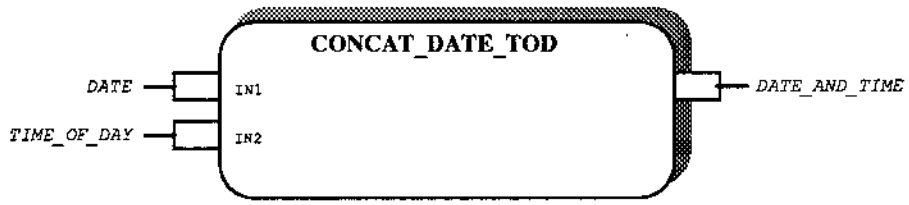
Exemple ST:

```
a.Val:=  DATE_AND_TIME_TO_DT  (
                              IN:=DT#02-Sep-1991-14:30:00);
```

Fixe a.Val sur D#02-Sep-1991.

CONCAT_DATE_TOD

Combine une date (DATE) et une heure du jour (TIME_OF_DAY) pour produire une date et heure (DATE_AND_TIME).



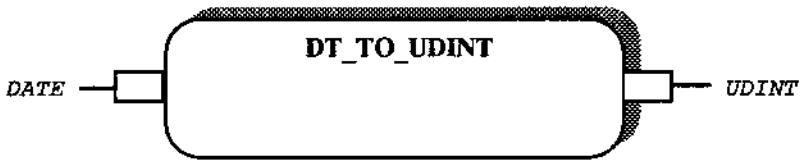
Exemple ST:

```
a.Val:=  CONCAT_DATE_TOD(IN1:=  D#02-Sep-1991,
                                IN2:=  TOD#14:30:00);
```

Fixe a.Val sur DT#02-Sep-1991-14:30:00.

DT_TO_UDINT (versions 2.27 et ultérieures)

Convertit une date (DATE) en un nombre entier sans signe (UDINT).

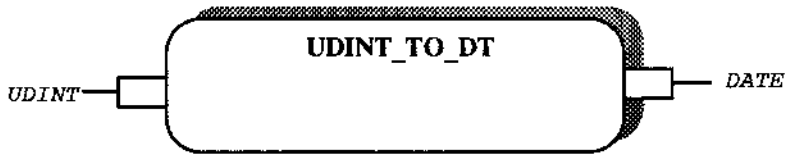


Exemple ST:

```
a.Val:=  DT_TO_UDINT  (IN:=DT#02-Sep-1993)
```

UDINT_TO_DT (versions 2.27 et ultérieures)

Convertit un nombre entier sans signe (UDINT) en (DATE).



Exemple ST:

```
date.Val:=  UDINT_TO_DT  (IN:=10349)
```

Chapitre 6

FONCTIONS DE CONVERSION DE CHAINES

Version 1

Sommaire

Sommaire (suite)

Présentation

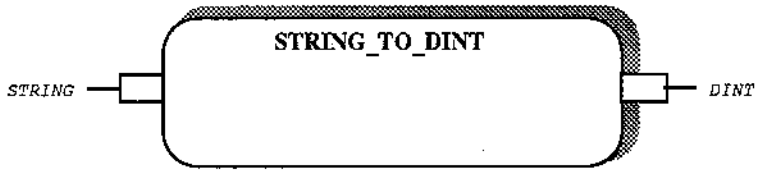
Ces fonctions permettent de convertir des données contenues dans des chaînes de caractères en un type de données particulier ou de convertir une valeur d'un type de données particulier en une chaîne. Elles sont normalement utilisées avec des applications comprenant des communications, comme lorsqu'une valeur a été reçue dans une chaîne sous la forme de texte ASCII et nécessite une conversion en un type de données IEC DIS 1131 ou lorsqu'il est nécessaire de transmettre la valeur d'un paramètre à un terminal sous la forme de texte ASCII. Par exemple, il est possible d'utiliser STRING_TO_REAL pour convertir une chaîne de caractères (STRING) contenant '1.23' en valeur à virgule flottante (REAL) ou d'utiliser TIME_TO_STRING pour convertir une durée en chaîne de caractères comme '18s100ms'.

Avec les fonctions de conversion des chaînes de caractères, une chaîne de longueur nulle, c'est-à-dire '', est restituée si le début de la chaîne d'entrée ne contient pas de caractères valables.

Le type de caractères qui définit des chaînes valables varie selon les différentes fonctions de conversion des chaînes de caractères et est spécifié dans la description des fonctions. Les fonctions de conversion des chaînes ne tiennent pas compte des espaces de tête et s'arrêtent au premier caractère qui n'est pas valable pour le type d'entrée considéré.

STRING_TO_DINT

Convertit une chaîne de caractères (SRING) en un nombre entier (DINT) qui contient les chiffres 0 à 9. Les signes "+" ou "-" placés devant les chiffres sont interprétés comme le signe de la sortie.



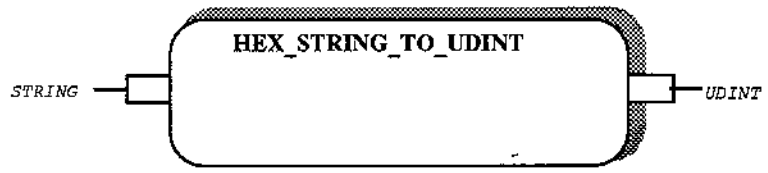
Exemple ST:

```
a.Val:=  STRING_TO_DINT(IN:=  '234');
```

Fixe a.Val sur 234.

HEX_STRING_TO_UDINT

Convertit une chaîne de caractères (STRING) contenant un codage hexadécimal ASCII avec les caractères 0-9, A, B, C, D, E, F en un nombre entier sans signe (UDINT). L'entrée ne doit pas être supérieure à '7FFFFFFF'. Les espaces de tête ne sont pas pris en compte et la conversion s'arrête au premier chiffre qui n'est pas hexadécimal.



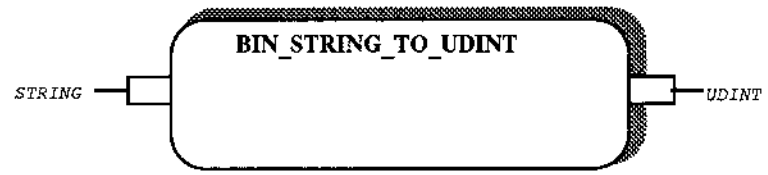
Exemple ST:

```
a.Val:= HEX_STRING_TO_UDINT (IN:= ' 2A');
```

Fixe a.Val sur 42.

BIN_STRING_TO_UDINT

Convertit une chaîne de caractères(STRING) contenant les chiffres "1" et "0" en un nombre entier (UDINT), l'entrée pouvant contenir jusqu'à 31 '1' et '0'. Les espaces de tête ne sont pas pris en compte et la conversion s'arrête au premier caractère qui n'est ni '1' ni '0'.



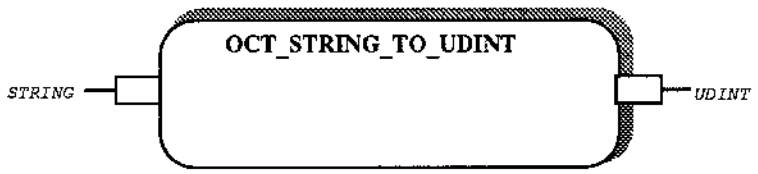
Exemple ST:

```
a.Val:= BIN_STRING_TO_UDINT(IN:= ' 10100101');
```

Fixe a.val sur 165.

OCT_STRING_TO_UDINT

Convertit une chaîne de caractères (STRING), contenant les chiffres '0' et '1' à '7' et qui représente une valeur octale, en un nombre entier (UDINT). L'entrée ne doit pas représenter une valeur octale supérieure à '17777777777'. Les espaces de tête dans la chaîne d'entrée ne sont pas pris en compte. La conversion s'arrête au premier caractère qui n'est pas compris entre '0 à '7'.



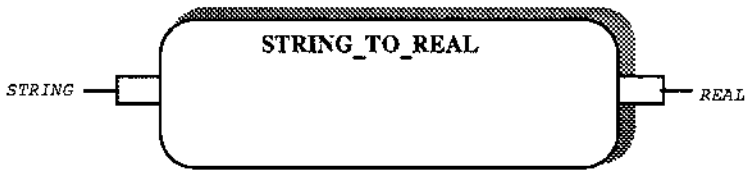
Exemple ST:

```
a.Val:= OCT_STRING_TO_UDINT (IN:= ' 070');
```

Fixe a.Val sur 56.

STRING_TO_REAL

Convertit une chaîne de caractères (STRING) en un nombre à virgule flottante (REAL). La chaîne d'entrée peut comporter des espaces de tête, un signe + ou - ou des espaces entre le signe et le premier chiffre. Un zéro de tête avant la virgule décimale est facultatif. Après le signe facultatif, l'entrée doit uniquement contenir les chiffres '0' à '9' et une seule virgule décimale. La conversion s'arrête au premier caractère non valable. Le format exponentiel n'est pas pris en charge.



Exemple ST:

```
a.Val:= STRING_TO_REAL(IN:= ' - .23');
```

Fixe a.Val sur -0,23.

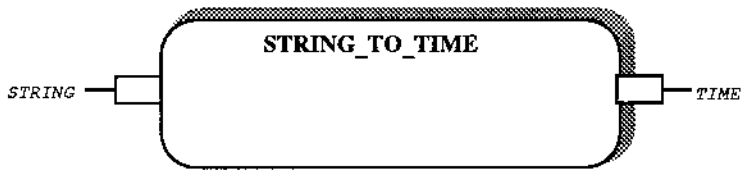
Exemple ST:

```
a.Val:= STRING_TO_REAL(IN:= '123.82FGZ);
```

Fixe a.Val sur 123,82.

STRING_TO_TIME

Convertit une chaîne de caractères (STRING) en une durée (TIME). La chaîne doit avoir la structure IEC, par exemple 5d14h12m18s100ms. Les champs nuls peuvent ne pas être indiqués, par exemple 5d18s. Le champ le moins significatif peut avoir une composante décimale.



Exemple ST:

```
a.Val:= STRING_TO_TIME(IN:= '5d14h');
```

Fixe a.Val sur T#5d_14h.

Exemple ST:

```
a.Val:=  STRING_TO_TIME(IN:=  '10.5h');
```

Fixe a.Val sur T#10h_30m.

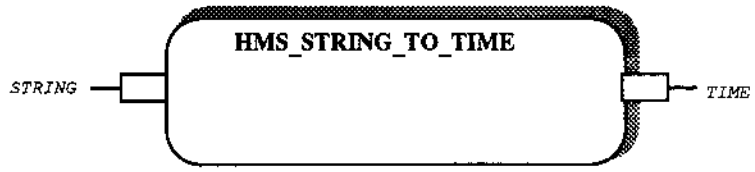
Exemple ST:

```
a.Val:=  STRING_TO_TIME(IN:=  '12m500ms');
```

Fixe a.Val sur T#12m0s500ms.

HMS_STRING_TO_TIME

Convertit une chaîne de caractères (STRING) en une durée (TIME). La chaîne doit avoir la structure heures, minutes, secondes, c'est-à-dire HH:MM:SS.



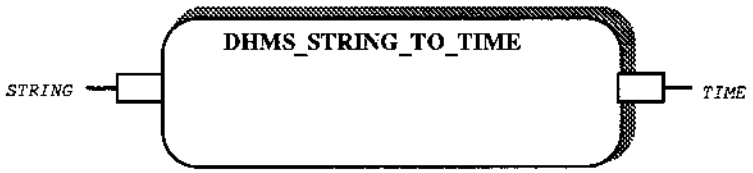
Exemple ST:

```
a.Val:= HMS_STRING_TO_TIME(IN:= '10:20:10');
```

Fixe a.Val sur T#10h_20m_10s.

DHMS_STRING_TO_TIME

Convertit une chaîne de caractères (STRING) en une durée (TIME). La chaîne doit avoir la structure jour, heure, minute, seconde, c'est-à-dire DD:HH:MM:SS.



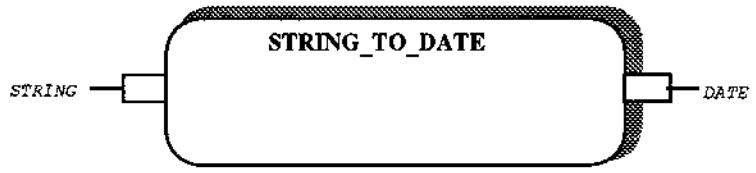
Exemple ST:

```
a.Val:=  DHMS_STRING_TO_TIME(IN:=  '20:10:20:10');
```

Fixe a.Val sur T#20d_10h_20m_10s.

STRING_TO_DATE

Convertit une chaîne de caractères (STRING) en une date (DATE). La chaîne doit avoir la structure IEC, c'est-à-dire année, mois, jour (YYYY-MM-DD).



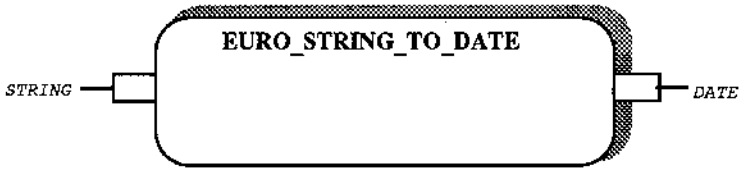
Exemple ST:

```
a.Val:=  STRING_TO_DATE(IN:='1991-09-02');
```

Fixe a.Val sr D#02-Sep-1991.

EURO_STRING_TO_DATE

Convertit une chaîne de caractères avec la date de structure européenne (STRING) en une date (DATE). La chaîne doit avoir la structure jour, mois, année, c'est-à-dire DD-MM-YY.



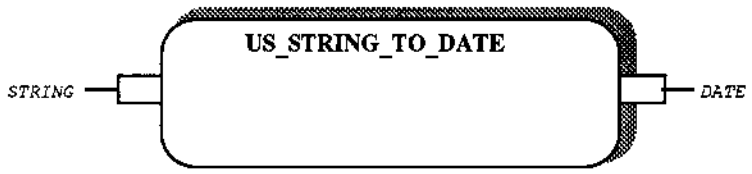
Exemple ST:

```
a.Val:=  EURO_STRING_TO_DATE(IN:=  '02-09-91');
```

Fixe a.Val sur D#02-Sep-1991.

US_STRING_TO_DATE

Convertit une CHAINE en présentation américaine en une DATE. La chaîne doit avoir la structure mois, jour, année, c'est-à-dire MM-DD-YY.



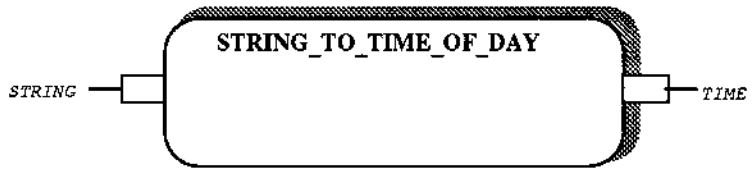
Exemple ST:

```
a.Val:=  US_STRING_TO_DATE(IN:=  '09-02-91');
```

Fixe a.Val sur D#02-Sep-1991.

STRING_TO_TIME_OF_DAY

Convertit une chaîne de caractères (STRING) en une heure du jour (TIME_OF_DAY). La chaîne doit avoir la structure IEC heure, minute, seconde, c'est-à-dire HH:MM:SS.



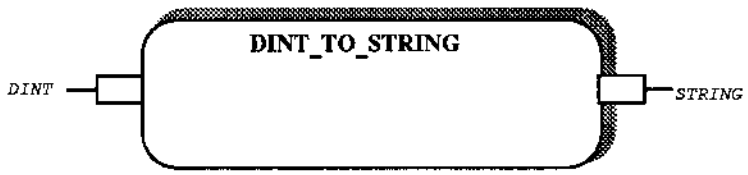
Exemple ST:

```
a.Val:=  STRING_TO_TIME_OF_DAY(IN:=  '12:30:10');
```

Fixe a.Val sur TOD#12:30:10.

DINT_TO_STRING

Convertit un entier double (DINT) en une chaîne de caractères décimaux avec signe (STRING).



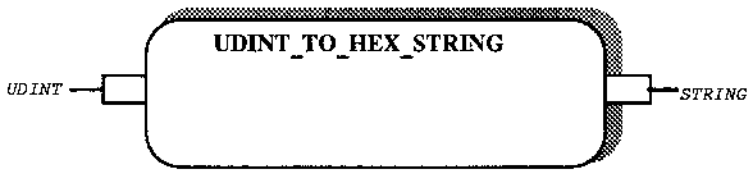
Exemple ST:

```
a.Val:=  DINT_TO_STRING(IN:=  -33);
```

Fixe a.val sur '-33'.

UDINT_TO_HEX_STRING

Convertit un nombre entier double sans signe (UDINT) en chaîne de caractères hexadécimaux (STRING). La valeur d'entrée ne doit pas être négative.



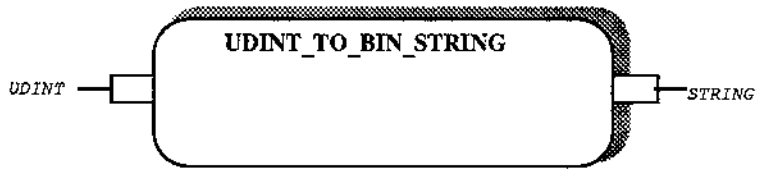
Exemple ST:

```
a.Val:=  UDINT_TO_HEX_STRING(IN:=  42);
```

Fixe a.Val sur '2A'.

UDINT_TO_BIN_STRING

Convertit un entier double sans signe (UDINT) en chaîne de caractères binaires (STRING). Seul le nombre de chiffres nécessaire pour représenter la valeur est créé ; aucun zéro de tête n'est ajouté. La valeur d'entrée ne doit pas être négative.



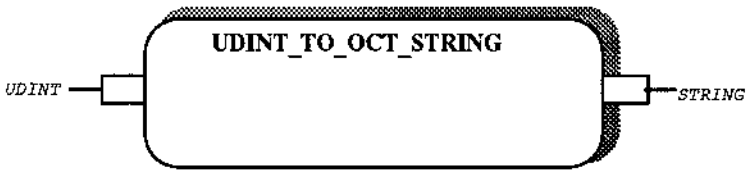
Exemple ST:

```
a.Val:=  UDINT_TO_BIN_STRING(IN:=  42);
```

Fixe a.Val sur '101010'

UDINT_TO_OCT_STRING

Convertit un entier double sans signe (UDINT) en chaîne de caractères octaux (STRING). Seul le nombre de chiffres nécessaire pour représenter la valeur est créé ; aucun zéro de tête n'est ajouté. La valeur d'entrée ne doit pas être négative.



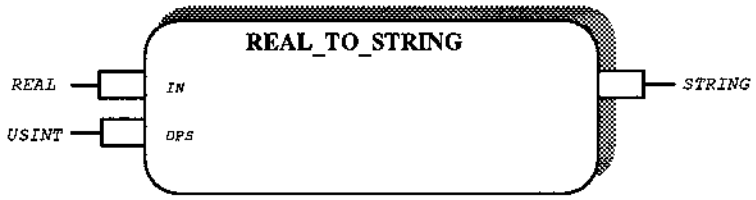
Exemple ST:

```
a.Val:=  UDINT_TO_OCT_STRING(IN:=  42);
```

Fixe a.Val sur '52'.

REAL_TO_STRING

Convertit un nombre à virgule flottante (REAL) en une chaîne de caractères égale à (STRING). Le paramètre DPS définit le nombre de décimales. Si DPS est égal à 0, il n'y a aucune décimale mais le dernier caractère est une virgule décimale. Si DPS est égal à -1, il n'y a aucune décimale et aucune virgule décimale.



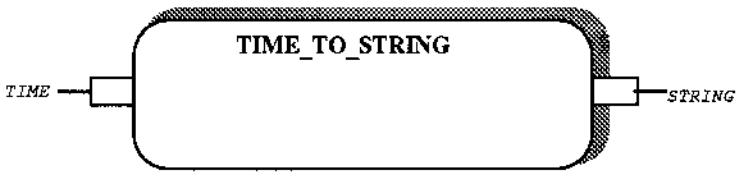
Exemple ST:

```
a.Val:= REAL_TO_STRING(IN:= 42.343, DPS := 2);
```

Fixe a.Val à '42.34'.

TIME_TO_STRING

Convertit une durée (TIME) en une chaîne de caractères (STRING). La chaîne a la structure IEC, par exemple 5d14h12m18s100ms.



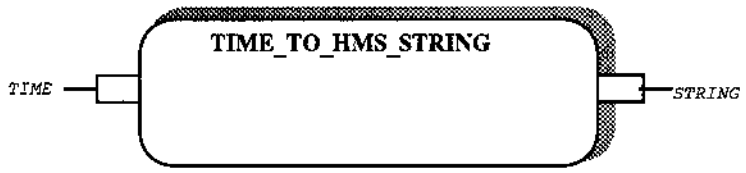
Exemple ST:

```
a.Val:= TIME_TO_SRING(IN:= T#12h_30m_20s);
```

Fixe a.Val sur '12h30m20s'.

TIME_TO_HMS_STRING

Convertit une durée (TIME) en une chaîne de caractères (STRING). La chaîne doit avoir la structure heure, minute, seconde, c'est-à-dire HH:MM:SS.



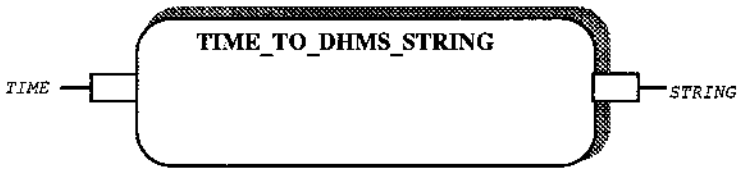
Exemple ST:

```
a.Val:=  TIME_TO_HMS_STRING(IN:=  T#12h_30m_20s);
```

Fixe a.Val sur '12:30:20'.

TIME_TO_DHMS_STRING

Convertit une durée (TIME) en une chaîne de caractères (STRING). La chaîne doit avoir la structure jour, heure, minute, seconde, c'est-à-dire DD:HH:MM:SS.



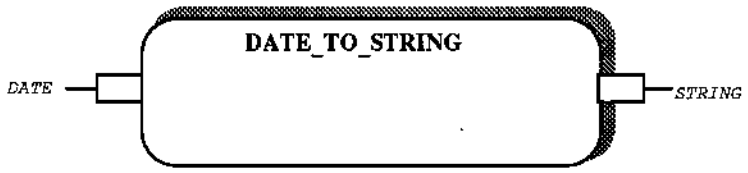
Exemple ST:

```
a.Val:=  TIME_TO_DHMS_STRING(IN:=  T#2d_12h_30m_20s);
```

Fixe a.Val sur '02:12:30:20'.

DATE_TO_STRING

Convertit une date (DATE) en une chaîne de caractères (STRING). La chaîne a la structure IEC année, mois, jour, c'est-à-dire YYYY-MM-DD.



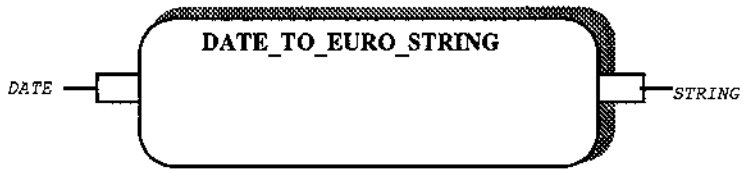
Exemple ST:

```
a.Val:=  DATE_TO_STRING(IN:=  D#02-Sep-1991)
```

Fixe a.Val sur '1991-09-02'.

DATE_TO_EURO_STRING

Convertit une date (DATE) en une chaîne de caractères de structure européenne (STRING). La chaîne a la structure jour, mois, année, c'est-à-dire DD-MM-YY.



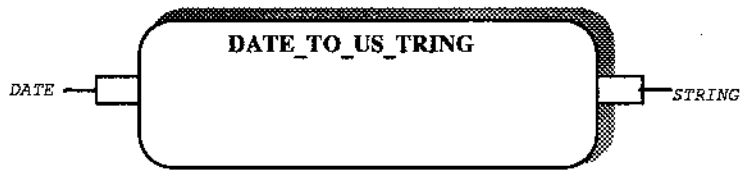
Exemple ST:

```
a.Val:=  DATE_TO_EURO_STRING(IN:=  D#02-Sep-1991);
```

Fixe a.val sur '02-09-91'.

DATE_TO_US_STRING

Convertit une date (DATE) en une chaîne de caractères de structure américaine (STRING). La chaîne a la structure mois, jour, année, c'est-à-dire MM-DD-YY.



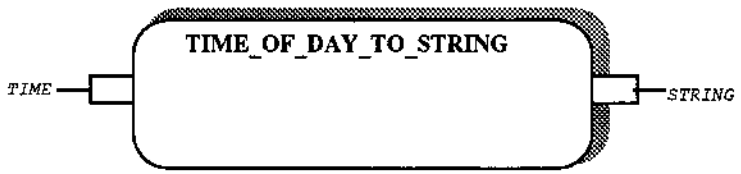
Exemple ST:

```
a.Val:=  DATE_TO_US_STRING(IN:=  D#02-Sep-1991)
```

Fixe a.Val sur '09-02-91'.

TIME_OF_DAY_TO_STRING

Convertit une heure du jour (TIME_OF_DAY) en une chaîne de caractères (STRING). La chaîne a la structure IEC heure, minute, seconde, c'est-à-dire HH:MM:SS.



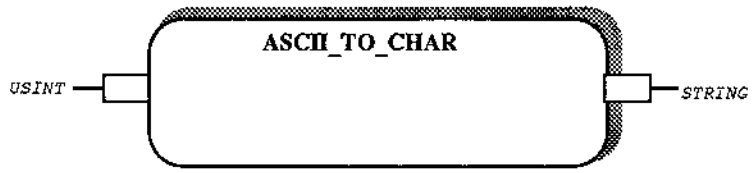
Exemple ST:

```
a.Val:=  TIME_OF_DAY_TO_STRING(IN:=  TOD#12:30:10);
```

Fixa a.Val sur '12:30:10'.

ASCII_TO_CHAR

Convertit un nombre entier (USINT) compris entre 0 et 255 en une chaîne de caractères ASCII. Si l'entier est en dehors de cette plage, la valeur restituée est 0.



Exemple ST:

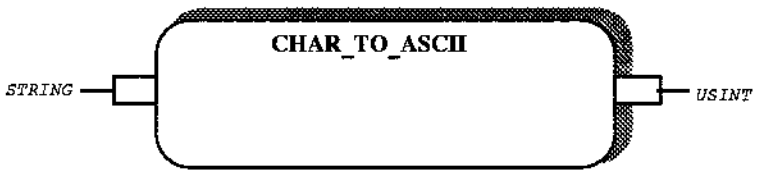
```
a.Val:=  ASCII_TO_CHAR(IN:=  66);
```

Fixe a.Val sur 'B'.

CHAR_TO_ASCII

Convertit un caractère ASCII (STRING) en un entier (USINT) compris entre 0 et 255.

Une chaîne vide donne la valeur 0.



Exemple ST:

```
a.Val:=  CHAR_TO_ASCII(IN:=  'B');
```

Fixe a.val sur 66.

Chapitre 7

FONCTIONS DE CALCULS ARITHMETIQUES SUR LE TEMPS

Version 2

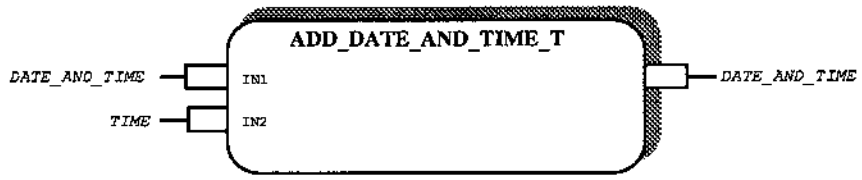
Sommaire

Présentation

La manipulation des calculs de dates et d'heures est facilitée par tout un ensemble de fonctions temporelles. Les fonctions offertes comprennent la plupart des opérations valables d'addition et de soustraction entre les différents types de données dates et heures. Par exemple, ADD_DATE_AND_TIME_T permet d'ajouter une période exprimée sous la forme d'une durée (TIME) à une date et heure (DATE_AND_TIME) pour créer une date et heure future (DATE_AND_TIME).

ADD_DATE_AND_TIME_T

Ajoute une durée (TIME) (IN2) à une date et heure (DATE_AND_TIME) (IN1) pour donner une date et heure (DATE_AND_TIME).



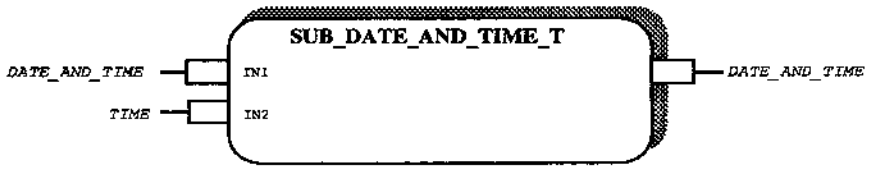
Exemple ST:

```
a.Val:=  ADD_DATE_AND_TIME_T(
           IN1:=  DT#02-Sep-1991-12:30:10,
           IN2:=  T#01d_01h_01m_01s);
```

Règle a.Val à DT#03-Sep-1991-13:31:11.

SUB_DATE_AND_TIME_T

Soustrait une durée (TIME) (IN2) d'une date et heure (DATE_AND_TIME) (IN1) pour donner une date et heure (DATE_AND_TIME).



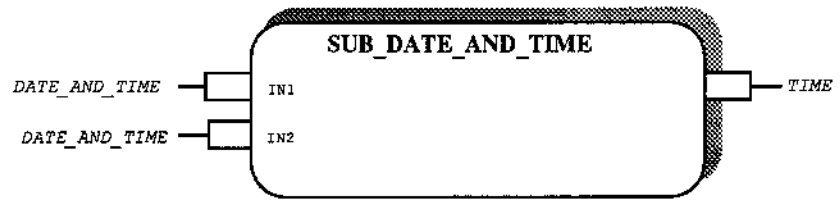
Exemple ST:

```
a.Val:= SUB_DATE_AND_TIME_T(
            IN1:=DT#02-Sep-1991-12:30:10,
            IN2:=T#01d_01h_01m_01s);
```

Règle a.Val à DT#01-Sep-1991-11:29:09.

SUB_DATE_AND_TIME

Soustrait une date et heure (DATE_AND_TIME) (IN2) d'une autre date et heure (DATE_AND_TIME) (IN1) pour donner une durée (TIME).



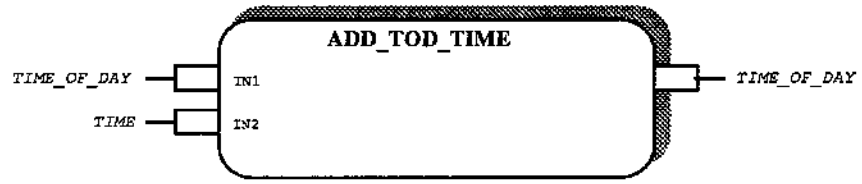
Exemple ST:

```
a.Val:= SUB_DATE_AND_TIME_T(
                IN1:= DT#02-Sep-1991-12:30:10,
                IN2:= DT#01-Sep-1991-11:29:09);
```

Règle a.Val à T#01d_01h_01m_01s.

ADD_TOD_TIME

Ajoute une durée (TIME) (IN2) à une heure du jour (TIME_0F_DAY) (IN1) pour donner une heure du jour (TIME_OF_DAY).



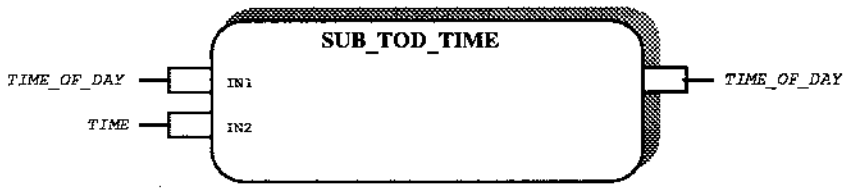
Exemple ST:

```
a.Val:=  ADD_TOD_TIME  (IN1:=  TOD#12:30:10,
                        IN2:=  T#20m_10s);
```

Règle a.Val à TOD#12:50:20.

SUB_TOD_TIME

Soustrait une durée (TIME) (IN2) d'une heure du jour (TIME_OF_DAY) (IN1) pour donner une heure du jour (TIME_OF_DAY).



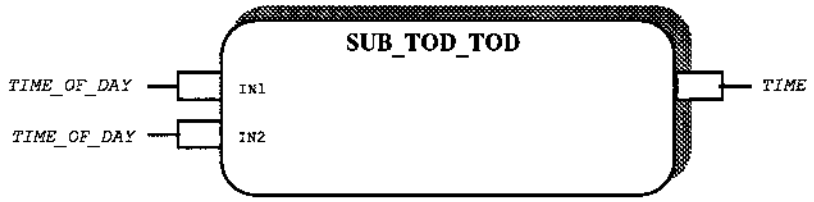
Exemple ST:

```
a.Val:=  SUB_TOD_TIME  (IN1:=  TOD#12:30:10,
                        IN2:=  T#20m_10s);
```

Règle a.Val à TOD#12:10:00.

SUB_TOD_TOD

Soustrait une heure du jour (TIME_OF_DAY) (IN2) d'une autre heure du jour (TIME_OF_DAY) (IN1) pour donner une durée (TIME). Il faut noter que, si IN2 est postérieure à IN1 dans la journée, la durée (TIME) sera restituée sous la forme T#0s..



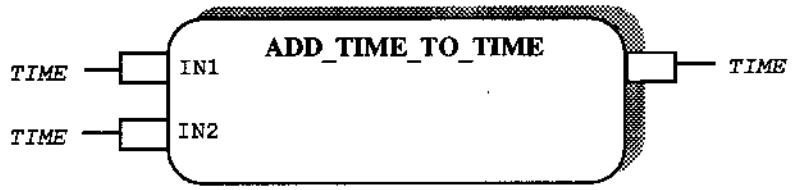
Exemple ST:

```
a.Val:=  SUB_TOD_TOD  (IN1:=  TOD#12:30:10,
                      IN2:=  TOD#10:20:08);
```

Règle time.Val à T#2h_10m_2s.

ADD_TIME_TO_TIME (versions 2.27 et postérieures)

Ajoute une durée (TIME) à une seconde durée (TIME). Le résultat est aussi une durée (TIME).



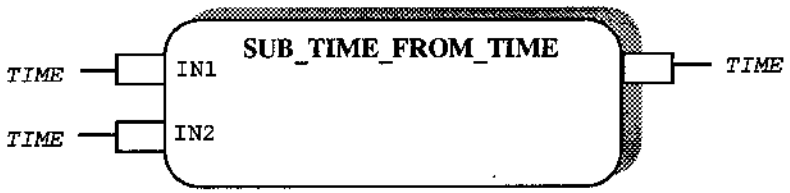
Exemple ST:

```
time.Val:= ADD_TIME_TO_TIME (IN1:=T#12m_14s,
                             IN2:=T#4h_18m_59s);
```

Règle time.Val à T#4h_31m_13s

SUB_TIME_FROM_TIME (versions 2.27 et postérieures)

Soustrait une durée (TIME) (IN2) d'une deuxième durée (TIME)(IN1).
Le résultat est aussi une durée (TIME). Si IN2 est supérieure à IN1, la fonction restitue une valeur de T#0mS.



Exemple ST:

```
time.Val:=  SUB_TIME_FROM_TIME  (IN1:=T#3d_5h_43m_12s,
                                 IN2:=T#4h_16m_4s);
```

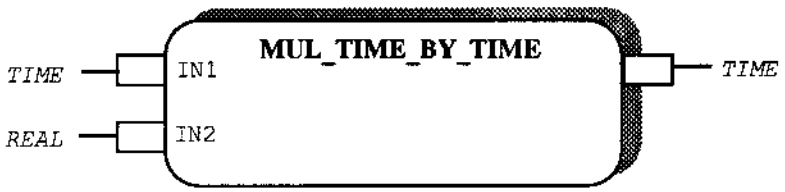
Règle time.Val à T#3d_1h_27m_8s

```
time.Val:=  SUB_TIME_FROM_TIME  (IN1:=T#1h_59m_6s,
                                 IN2:=T#1d_3h-4m_36s);
```

Règle time.Val à T#0mS

MUL_TIME_BY_REAL (versions 2.27 et postérieures)

Multiplie une durée (TIME) par une valeur numérique (REAL). Le résultat est une durée (TIME).



Exemple ST:

```
time.Val:=  MUL_TIME_BY_REAL  (IN1:=T#1d_3h_36m_12s,
                               IN2:=2.0);
```

Règle time.Val à T#2d_7h_12m_24s.

Chapitre 8

FONCTIONS COMPACTES

Version 1

Sommaire

Sommaire (suite)

Présentation

Une série de fonctions compactes permet de compacter ou de décompacter un ensemble de valeurs d'un type de données particulier dans une chaîne. Ces fonctions peuvent servir à un certain nombre d'applications comme le compactage d'une série de valeurs en une chaîne de caractères pour la transmission par les communications ou l'inclusion d'un ensemble de valeurs dans un système de recettes. Par exemple, REP_SINT_IN_STR remplace un entier court dans une chaîne à un emplacement fixé.

Les fonctions compactes permettent de traiter les chaînes de caractères comme les variables utilisateur Long_string ou les variables déportées Remote_Str comme des ensembles de valeurs de types de données spécifiés. Il existe une paire de fonctions pour chaque type de données élémentaire : une fonction remplace une valeur dans la chaîne (REP_***_IN_STR) et une fonction extrait une valeur d'une chaîne (EXT_***_FROM_STR) où *** est le nom du type de données.

La fonction 'remplace' sert aussi à insérer une valeur initiale dans la chaîne de caractères. En règle générale, l'accès aux chaînes devrait uniquement se faire à l'aide de ces fonctions. Il faut normalement stocker un seul type de données dans une chaîne. Le paramètre INDEX sert à sélectionner l'élément de l''ensemble' à lire ou écrire. Le premier élément a un INDEX égal à zéro. Les éléments sont stockés de la gauche vers la droite dans la chaîne et condensés en octets adjacents ; les valeurs ne sont séparées par aucun octet vide.

Si l'on essaie de lire un élément situé après la fin de la chaîne, la valeur par défaut de ce type de données (0 normalement) est restituée. Si l'on essaie d'écrire un élément situé après la fin de la chaîne, la chaîne sera prolongée de manière à inclure cet élément. Lors du prolongement des chaînes, les nouveaux éléments sont configurés pour la valeur par défaut de ce type de données. Par exemple, lors de la création d'une variable utilisateur Long_String, elle devient par défaut une chaîne de longueur nulle, c'est-à-dire ''. Si une valeur est ajoutée à la chaîne à l'aide d'INDEX3, la chaîne sera prolongée pour contenir 4 éléments (0, 1, 2 et 3) ; les valeurs des éléments 0, 1 et 2 seront initialisées pour prendre des valeurs nulles, c'est-à-dire 0 pour les éléments numériques.

Exemple ST:

```
str1.Val:=  '';  (*  Initialize  String*)
str1.Val:=  REP_INT_IN_STR(STR:=  str1.Val,
                           INDEX:=  3,
                           VAL:=  3000);
int0.Val:=  EXT_INT_FROM_STR(STR:=  str1.Val,
                           INDEX:=  0);
```

```
int1.Val:=  EXT_INT_FROM_STR(STR:=  str1.Val,
                             INDEX:=  1);
int3.Val  :=  EXT_INT_FROM_STR(STR:=  str1.Val,
                             INDEX:=  3);
```

Règle int0.Val à 0, int1.Val à 0 et int3.Val à 3000.

> N.B. : les chaînes manipulées à l'aide des fonctions COMPACTES ne doivent pas être ensuite manipulées à l'aide des fonctions STRING (chaîne) car des valeurs internes risqueraient d'être perdues en raison d'un mauvais alignement dans la zone de stockage de la chaîne.

Si la chaîne de destination d'un REP_***_IN_STR ne peut pas être prolongée pour contenir le résultat parce qu'elle dépasse la longueur maximale de la chaîne, l'ensemble de la chaîne sera positionné sur une chaîne nulle, c'est-à-dire ".

Ces fonctions sont normalement utilisées avec des chaînes qui ont une longueur maximale de 255 octets comme la variable utilisateur Long_String mais peuvent être utilisées avec des chaînes plus courtes. Dans ce cas, le nombre de valeurs qui peuvent être condensées sera réduit et il faut faire attention à ne pas extraire ou remplacer des valeurs après la fin de la chaîne.

Se reporter aux fonctions REP_BOOL_IN_STR et EXT_BOOL_FROM_STR pour avoir des exemples de ST. Toutes les fonctions suivent le même modèle, bien qu'elles extraient ou remplacent des types de données différents et que le nombre de valeurs condensable en une chaîne varie.

Inversion d'octets

Si la transmission de chaînes compactes à d'autres équipements ou la réception de chaînes compactes provenant d'autres équipements s'impose, par exemple par l'intermédiaire de communications série, il peut être nécessaire d'utiliser la variante Inversion d'octets de ces fonctions. Elles ont la forme

REP_***_IN_STR_X et EXT_***_FROM_STR_X.

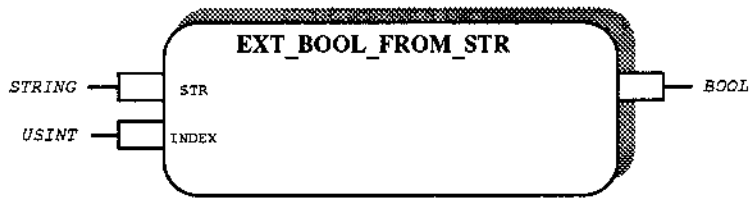
L'inversion d'octets peut être nécessaire pour les types de données en cas de transfert des chaînes compactes entre des systèmes qui utilisent une implantation mémoire différente du PC3000.

Par exemple, si le PC3000 reçoit des chaînes contenant des blocs d'entiers 32 bits (entier double DINT) qui ont été créés sur certains automates programmables, la fonction EXT_DINT_FROM_STR_X peut être nécessaire.

REP_DINT_IN_STR_X peut être nécessaire pour créer une chaîne contenant un bloc d'entiers à transmettre à certains automates programmables.

EXT_BOOL_FROM_STR

Extrait une valeur booléenne (BOOL) d'une chaîne de caractères. Huit valeurs booléennes sont stockées dans chaque caractère, ce qui permet d'en stocker un maximum de 2040 dans une chaîne de 255 caractères (STRING). INDEX est compris entre 0 et 2039.



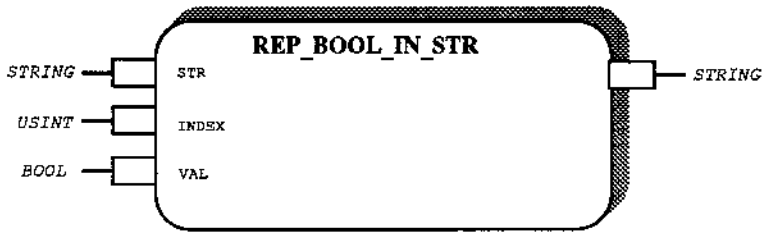
Exemple ST:

```
bool.Val:=  EXT_BOOL_FROM_STR(STR  :=  str1Val,
                                INDEX:=  300);
```

bool.Val est réglée à l'état du 301ème bit condensé dans la chaîne str1.Val.

REP_BOOL_IN_STR

Remplace une valeur booléenne dans une chaîne. Huit valeurs booléennes (BOOL) sont stockées dans chaque caractère, ce qui permet d'en stocker un maximum de 2040 dans une chaîne de 255 caractères (STRING). INDEX est compris entre 0 et 2039.



Exemple ST:

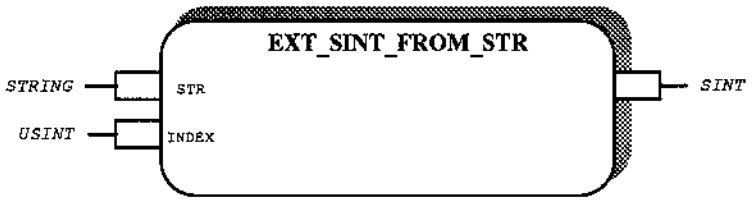
```
strl.Val:= REP_BOOL_IN_STR  (STR  :=  srl.Val,
                             INDEX  :=  300,
                             VAL  :=  1  (*  TRUE  *)  );
```

Le 301ème bit condensé dans la chaîne str1.Val est réglé à 1, c'est-à-dire VRAI.

EXT_SINT_FROM_STR

Extrait un entier court avec signe (SINT) d'une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme d'un seul caractère de la chaîne, ce qui permet de stocker un maximum de 255 entiers courts dans une chaîne de 255 caractères. INDEX est compris entre 0 et 254.

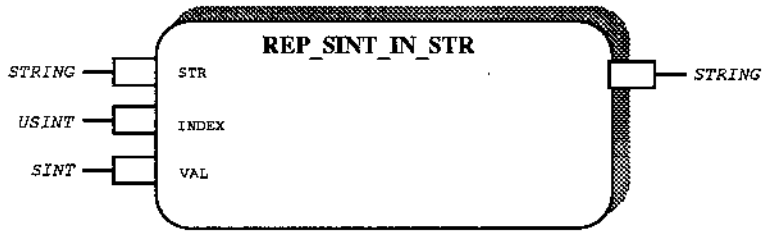
Restitue une valeur comprise entre -128 et 127.



REP_SINT_IN_STR

Remplace un entier court avec signe (SINT) dans une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme d'un seul caractère de la chaîne, ce qui permet de stocker un maximum de 255 entiers courts dans une chaîne de 255 caractères. INDEX est compris entre 0 et 254.

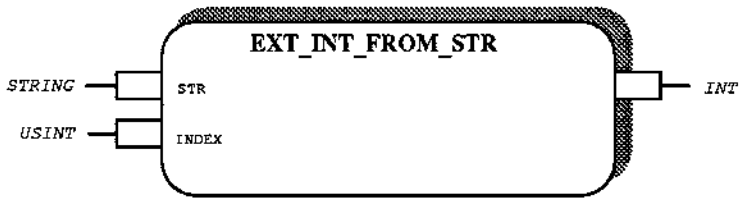
VAL est comprise entre -128 et 127.



EXT_INT_FROM_STR

Exrait un entier avec signe (INT) d'une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de deux caractères de la chaîne (STRING), ce qui permet de stocker un maximum de 127 entiers avec signe dans une chaîne de 255 caractères. INDEX est compris entre 0 et 126.

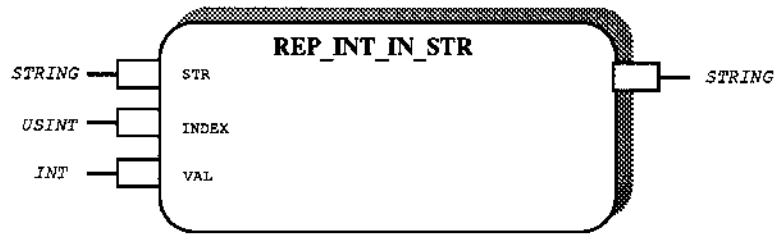
Restitue une valeur comprise entre -32768 et 32767.



REP_INT_IN_STR

Remplace un entier avec signe (INT) dans une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de deux caractères de la chaîne, ce qui permet d'en stocker un maximum de 127 dans une chaîne de 255 caractères. INDEX est compris entre 0 et 126.

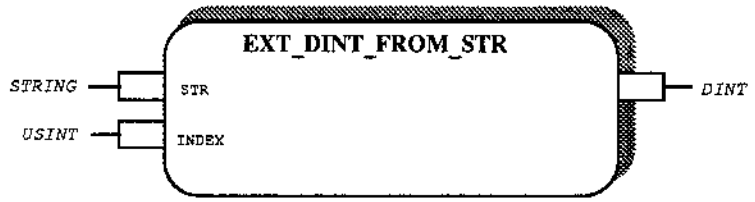
VAL est comprise entre -32768 et 32767.



EXT_DINT_FROM_STR

Extrait un entier double avec signe (DINT) d'une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de quatre caractères de la chaîne, ce qui permet d'en stocker un maximum de 63 dans une chaîne de 255 caractères. INDEX est compris entre 0 et 62.

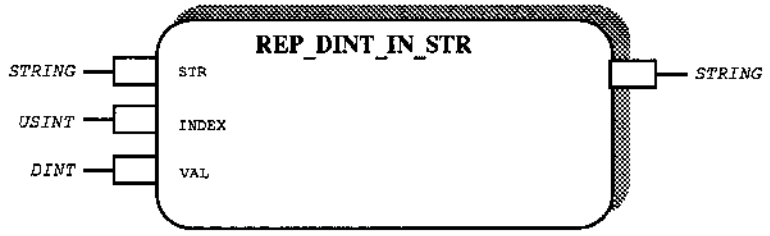
Restitue une valeur comprise entre -2147483648 et 2147483647.



REP_DINT_IN_STR

Remplace un entier double avec signe (DINT) dans une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de quatre caractères de la chaîne, ce qui permet d'en stocker un maximum de 63 dans une chaîne de 255 caractères. INDEX est compris entre 0 et 62.

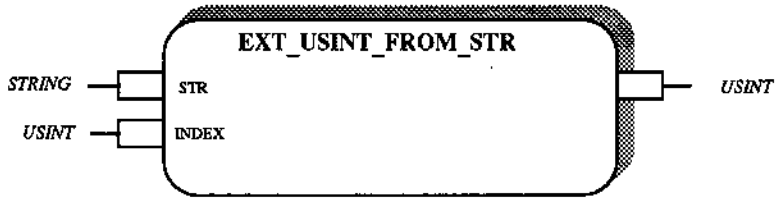
VAL est comprise entre -2147483648 et 2147483647.



EXT_USINT_FROM_STR

Extrait un entier court sans signe (USINT) d'une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme d'un seul caractère de la chaîne, ce qui permet d'en stocker un maximum de 255 dans une chaîne de 255 caractères. INDEX est compris entre 0 et 254.

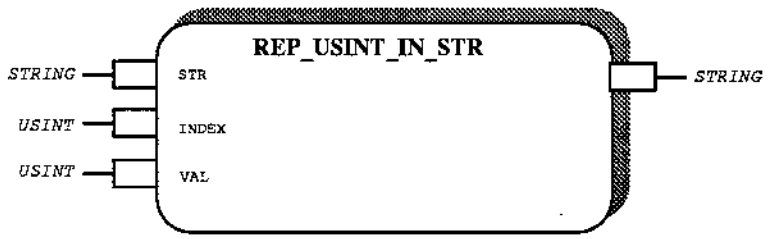
Restitue une valeur comprise entre 0 et 255.



REP_USINT_IN_STR

Remplace un entier court sans signe (USINT) dans une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme d'un seul caractère de la chaîne, ce qui permet d'en stocker un maximum de 255 dans une chaîne de 255 caractères. INDEX est compris entre 0 et 254.

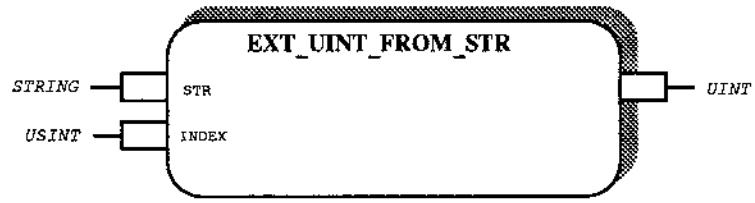
VAL est comprise entre 0 et 255.



EXT_UINT_FROM_STR

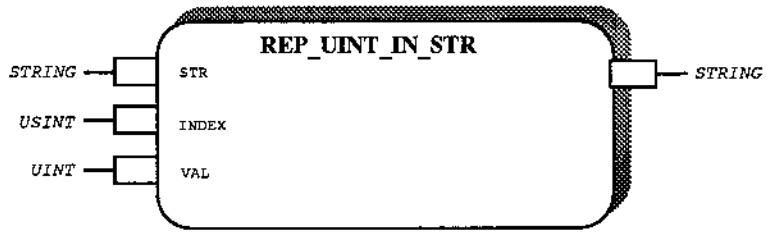
Extrait un entier sans signe (UINT) d'une chaîne de caractères. Chaque valeur est stockée sous la forme de deux caractères de la chaîne, ce qui permet d'en stocker un maximum de 127 dans une chaîne de 255 caractères. INDEX est compris entre 0 et 126.

Restitue une valeur comprise entre 0 et 65535.



REP_UINT_IN_STR

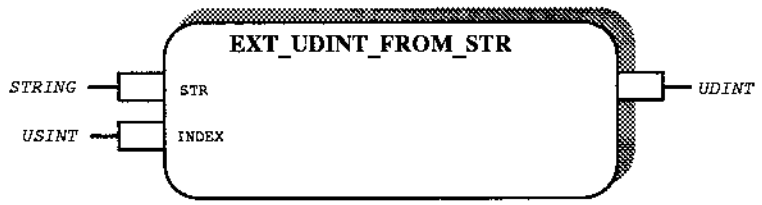
Remplace un entier sans signe (UINT) dans une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de deux caractères de la chaîne, ce qui permet d'en stocker un maximum de 127 dans une chaîne de 255 caractères.



EXT_UDINT_FROM_STR

Extrait un entier double sans signe (UDINT) d'une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de quatre caractères de la chaîne, ce qui permet d'en stocker un maximum de 63 dans une chaîne de 255 caractères. INDEX est compris entre 0 et 62.

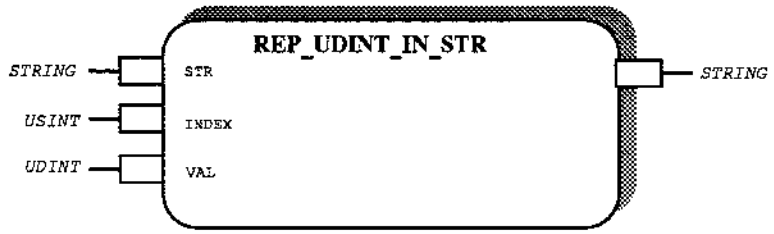
Restitue une valeur comprise entre 0 et 4294967295.



REP_UDINT_IN_STR

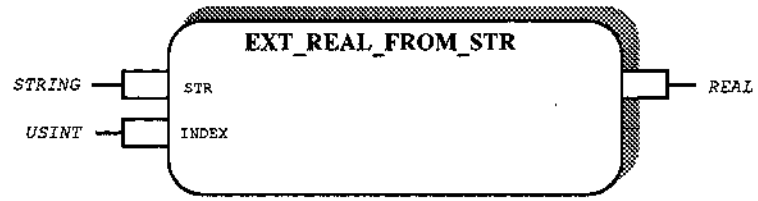
Remplace un entier double sans signe (UDINT) dans une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de quatre caractères de la chaîne, ce qui permet d'en stocker un maximum de 63 dans une chaîne de 255 caractères. INDEX est compris entre 0 et 62.

VAL est comprise entre 0 et 4294967294.



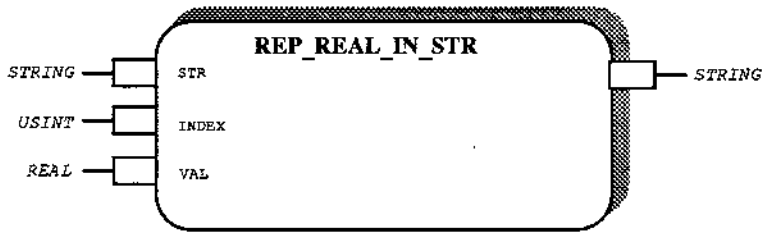
EXT_REAL_FROM_STR

Extrait une valeur à virgule flottante (REAL) d'une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de quatre caractères de la chaîne, ce qui permet d'en stocker un maximum de 63 dans une chaîne de 255 caractères. La valeur est stockée avec la structure IEEE Single Precision Binary Real. INDEX est compris entre 0 et 62.



REP_REAL_IN_STR

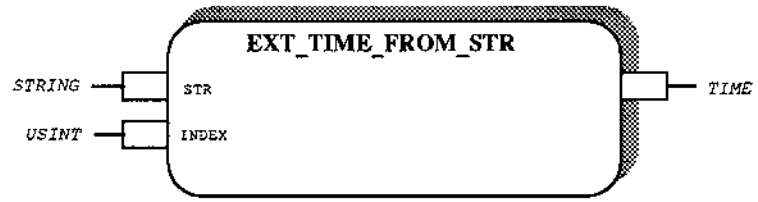
Remplace une valeur à virgule flottante (REAL) dans une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de quatre caractères de la chaîne, ce qui permet d'en stocker un maximum de 63 dans une chaîne de 255 caractères. La valeur est stockée avec la structure IEEE Single Precision Binary Real. INDEX est compris entre 0 et 62.



EXT_TIME_FROM_STR

Extrait une durée (TIME) d'une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de quatre caractères de la chaîne, ce qui permet d'en stocker un maximum de 63 dans une chaîne de 255 caractères. La valeur est stockée sous la forme d'un nombre entier de millisecondes. INDEX est compris entre 0 et 62.

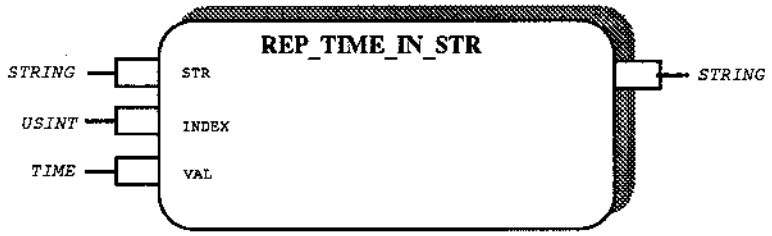
Restitue une valeur comprise entre 0ms et 49d17h2m47s295ms.



REP_TIME_IN_STR

Remplace une durée (TIME) dans une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de quatre caractères de la chaîne, ce qui permet d'en stocker un maximum de 63 dans une chaîne de 255 caractères. INDEX est compris entre 0 et 62.

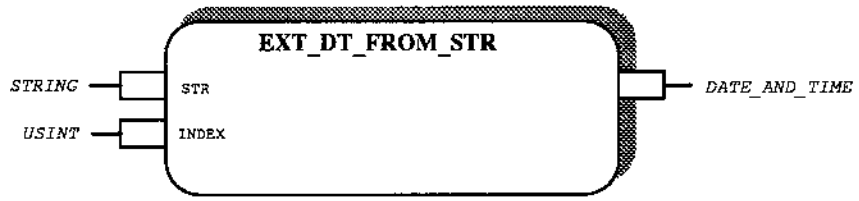
VAL est comprise entre 0ms et 49d17h2m47s295ms.



EXT_DT_FROM_STR

Extrait une date et heure d'une chaîne de caractères. Chaque valeur est stockée sous la forme de quatre caractères de la chaîne, ce qui permet d'en stocker un maximum de 63 dans une chaîne de 255 caractères.

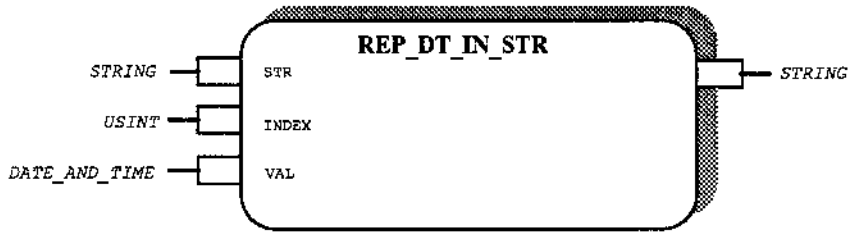
La valeur est stockée sous la forme d'un nombre entier de secondes depuis 1970-01-01-00:00:00. INDEX est compris entre 0 et 62.



REP_DT_IN_STR

Remplace une date et heure (DATE_AND_TIME) dans une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de quatre caractères de la chaîne, ce qui permet d'en stocker un maximum de 63 dans une chaîne de 255 caractères.

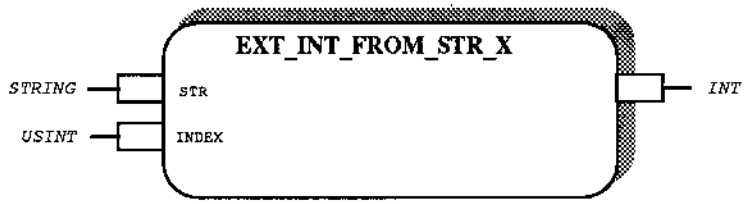
La valeur est stockée sous la forme d'un nombre entier de secondes depuis 1970-01-01-00:00:00. INDEX est compris entre 0 et 62.



EXT_INT_FROM_STR_X

Extrait un entier avec signe (INT), dans l'ordre inverse des octets, d'une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de deux caractères de la chaîne, ce qui permet d'en stocker un maximum de 127 dans une chaîne de 255 caractères. INDEX est compris entre 0 et 126.

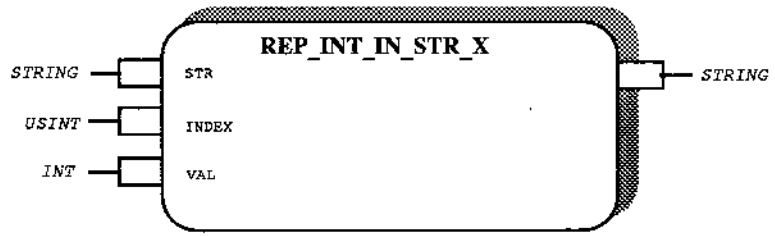
Restitue une valeur comprise entre -32768 et 32767.



REP_INT_IN_STR_X

Remplace un entier avec signe (INT), dans l'ordre inverse des octets, dans une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de deux caractères de la chaîne, ce qui permet d'en stocker un maximum de 127 dans une chaîne de 255 caractères. INDEX est compris entre 0 et 126.

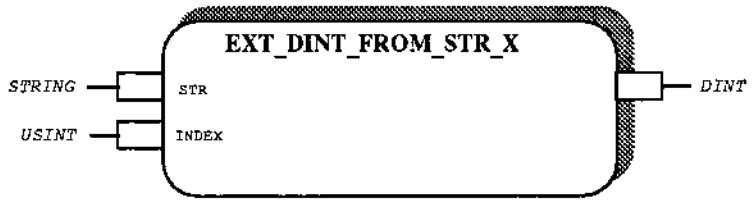
VAL est comprise entre -32768 et 32767.



EXT_DINT_FROM_STR_X

Extrait un entier double avec signe (DINT), dans l'ordre inverse des octets, d'une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de quatre caractères de la chaîne, ce qui permet d'en stocker un maximum de 63 dans une chaîne de 255 caractères. INDEX est compris entre 0 et 62.

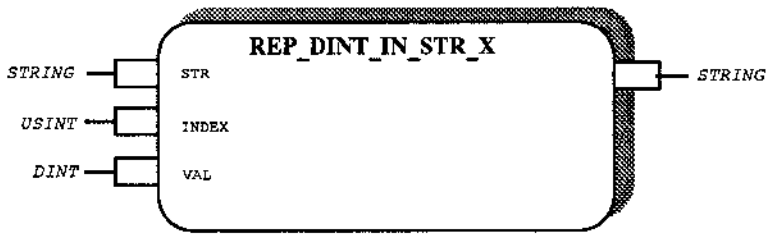
Restitue une valeur comprise entre -2147483648 et 2147483647.



REP_DINT_IN_STR_X

Remplace un entier double avec signe (DINT), dans l'ordre inverse des octets, dans une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de quatre caractères de la chaîne, ce qui permet d'en stocker un maximum de 63 dans une chaîne de 255 caractères. INDEX est compris entre 0 et 62.

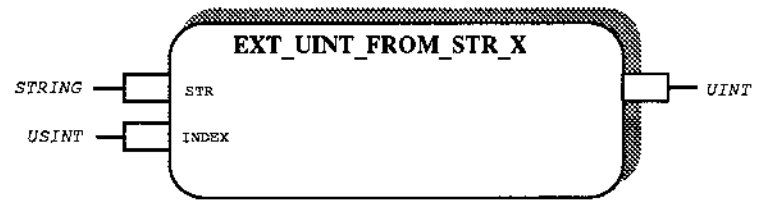
VAL est comprise entre -2147483648 et 2147483647.



EXT_UINT_FROM_STR_X

Extrait un entier sans signe (DINT), dans l'ordre inverse des octets, d'une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de deux caractères de la chaîne, ce qui permet d'en stocker un maximum de 127 dans une chaîne de 255 caractères. INDEX est compris entre 0 et 126.

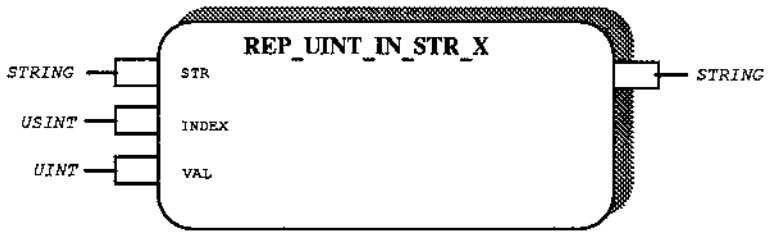
Restitue une valeur comprise entre 0 et 65535.



REP_UINT_IN_STR_X

Remplace un entier sans signe (UINT), dans l'ordre inverse des octets, dans une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de deux caractères de la chaîne, ce qui permet d'en stocker un maximum de 127 dans une chaîne de 255 caractères. INDEX est compris entre 0 et 126.

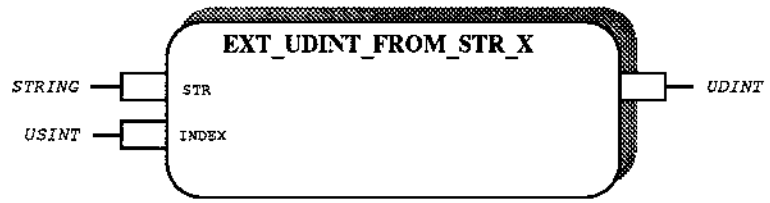
VAL est comprise entre 0 et 65535.



EXT_UDINT_FROM_STR_X

Extrait un entier double sans signe (UDINT), dans l'ordre inverse des octets, d'une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de quatre caractères de la chaîne, ce qui permet d'en stocker un maximum de 63 dans une chaîne de 255 caractères. INDEX est compris entre 0 et 62.

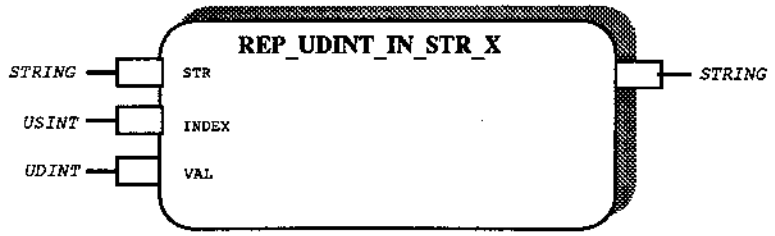
Restitue une valeur comprise entre 0 et 4294967294.



REP_UDINT_IN_STR

Remplace un entier double sans signe (USINT), dans l'ordre inverse des octets, dans une chaîne de caractères (STRING). Chaque valeur est stockée sous la forme de quatre caractères de la chaîne, ce qui permet d'en stocker un maximum de 63 dans une chaîne de 255 caractères. INDEX est compris entre 0 et 62.

Restitue une valeur comprise entre 0 et 4294967294.



Chapitre 9

FONCTIONS SUPPLEMENTAIRES DE COMMUNICATION

Version 1

Sommaire

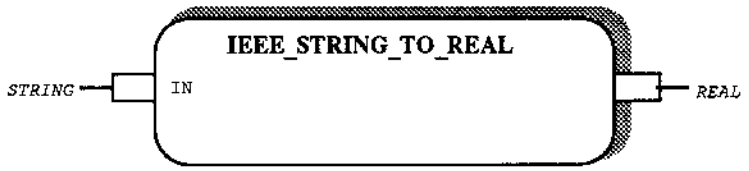
Présentation

Deux fonctions permettent de comprimer et de décomprimer des nombres IEEE stockés dans des chaînes à l'aide de la structure de codage IEEE EI Bisync. Ces fonctions assurent le codage ou le décodage des valeurs de paramètres à transmettre avec le protocole EIBisync.

Elles sont uniquement nécessaires pour les applications spécialisées dans lesquelles il faut construire ou décoder des messages composés. Dans la majeure partie des applications, le codage et le décodage des valeurs de paramètres sont traités automatiquement par le bloc fonction du module EI Bisync.

IEE_STRING_TO_REAL

Convertit une variable à virgule flottante IEEE (REAL), condensée en une chaîne de caractères (STRING) avec la structure de codage EI Bisync, en une variable à virgule flottante (REAL). Consulter le document Présentation des communications PC3000 et le chapitre consacré aux communications "Esclave EI Bisync" dans le "Manuel des blocs fonctions" PC 3000 pour avoir des détails sur le codage des valeurs IEEE.



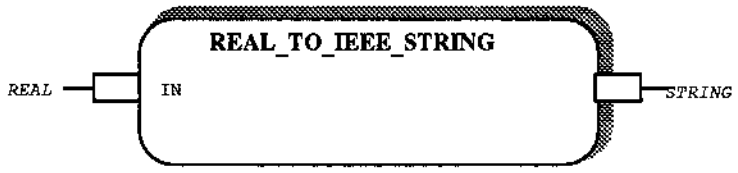
Exemple ST:

```
real1.Val:=  IEEE_STRING_TO_REAL  (IN  :=  @op');
```

Règle real1.Val à -0,5.

REAL_TO_IEEE_STRING

Convertit une variable à virgule flottante (REAL) en variable chaîne de caractères (STRING) à l'aide du codage EI Bisync IEEE.



Exemple ST:

```
str1.Val:=  IEEE_REAL_TO_STRING(IN:=  -0.5);
```

Règle str1.Val à '@op'.

Chapitre 10

Fonctions de manipulation de bits

Version 1

Sommaire

Présentation

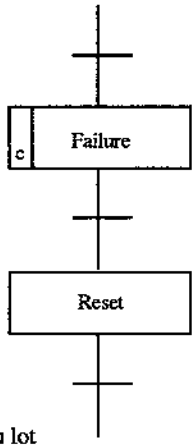
Les nombres entiers sont souvent interprétés comme des configurations binaires. C'est notamment le cas lorsque le traitement des communications série doit se faire à un niveau élémentaire ou lorsque les états d'alarme sont assemblés en codes d'erreur.

Pour simplifier l'utilisation des nombres entiers, les fonctions BIT_MANIPUL permettent de régler les bits des nombres entiers, d'extraire des bits individuel des nombres entiers et de combiner et comparer des configurations binaires de nombres entiers par des opérations AND, OR, XOR et NOT au niveau du bit.

Par exemple, au cours d'un traitement séquentiel, nous voulons surveiller toutes les machines pour détecter un éventuel défaut dans un lot donné et remettre cet enregistrement à l'état initial à la fin du lot. Si l'état des différentes machines arrive au PC3000 sous forme d'entrées numériques et si l'état des machines de l'usine est représenté par un seul nombre entier surveillé par un ordinateur de contrôle, le programme séquentiel contiendra peut-être ce qui suit :

Programme séquentiel :

...du démarrage des étapes du lot



...jusqu'à l'achèvement des étapes du lot

Liste ST :

```
(* CONTINUOUS *)
STEP  failure :
 (* update failure record if any machine has failed *)
 (* machine 4 *)
 IF mc4_fail.Process_Val = 1(*On*) THEN
  failures.Val := SET_BIT_IN_DINT( VALUE :=  failures.Val
                  , BIT_NO :=  4 , BIT :=  1 ) ;
```

```
 END_IF;
 (* machine 5 *)
 IF mc5_fail.Process_Val = 1(*On*) THEN
  failures.Val := SET_BIT_IN_DINT( VALUE :=  failures.Val
                  , BIT_NO :=  5 , BIT :=  1 ) ;
 END_IF;
END_STEP

TRANSITION
  FROM failure
  TO   reset
:=  batchend.Process_Val = 1(*end*) ;
END_TRANSITION

(* SINGLE SHOT *)
STEP  reset :
 (* reset reason code at end of batch *)
 failures.Val := 0 ;
END_STEP
```

A noter que ce programme donne un effet différent à l'utilisation du bloc fonction BoolToDint. Avec lui, les défauts sont "verrouillés" à la variable dint "failures", tandis que le bloc fonction BoolToDint ne produirait que la mise à jour continue de la situation de défaut.

De manière semblable, la valeur d'un bit d'un nombre entier peut être saisie à un point donné du cycle de traitement. Ce nombre entier peut provenir d'un flux continu de nombres entiers produit par un simple appareil communicant. Au moment du cycle où la valeur doit être saisie, une étape contenant la ligne suivante de ST est exécutée :

mc_state.Val := GET_BIT_FROM_DINT (VALUE := commsval.Val ,
BIT_NO:=4);

Des opérations plus complexes peuvent être exécutées en utilisant les fonctions de manipulation de bit AND, OR, XOR et NOT.

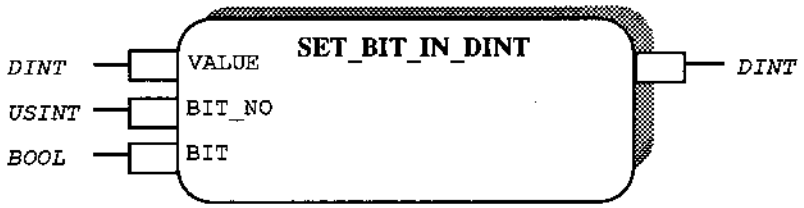
NOMBRES ENTIERS DOUBLES (DINTS)

Contrairement à nombre d'automates programmables (PLC), les mathématiques du LCM utilisent des nombres entiers à 32 bits à signe en mode complément de deux. Les numéros de bit vont de zéro (le moins significatif) à 31 (le plus significatif). Les bits entre zéro et 30 ont un poids binaire égal à leur numéro et sont positifs. Ainsi, le nombre décimal 123 représente les numéros de bit 0, 1, 3, 4, 5 et 6 réglés tandis que le nombre décimal 1,073,741,824 représente le numéro de bit 30 réglé. Pourtant, le bit trente et un porte un signe négatif ainsi que le poids binaire de trente et un. Ainsi, si seul le bit trente et un est réglé, le nombre entier a une valeur décimale de -2,147,483,648. Si tous sont réglés, le nombre entier a une valeur décimale de -1.

Il est important que les personnes habituées à travailler avec des nombres entiers de seize bits sans signe comprennent cette différence, notamment lorsqu'elles font des comparaisons sur la base de la valeur décimale de configurations binaires.

SET_BIT_IN_DINT (version 3.0 et supérieure)

Régler un bit dans un double nombre entier avec signe (DINT). Le DINT sur lequel l'opération doit être effectuée, le bit à régler et l'état auquel doit être mis ce bit sont donnés comme des arguments à la fonction. Cette fonction peut servir à modifier un dint ou à transférer une valeur modifiée sur un dint différent.



Exemple de modification d'un dint:

```
dint.Val := SET_BIT_IN_DINT ( VALUE := dint.Val,
                              BIT_NO := 3, BIT := 1 ) ;
```

le troisième bit de "dint" est mis sur "activé" 1.

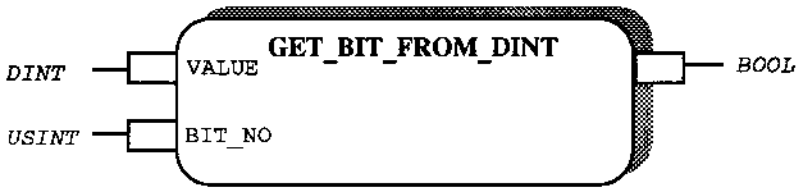
Exemple de passage d'une valeur modifiée sur un autre dint :

```
other.Val := SET_BIT_IN_DINT ( VALUE := thisone.Val,
                               BIT_NO := 5, BIT := 1 ) ;
```

la valeur de "thisone" est mise à "other" avec le cinquième bit réglé, quel que soit son état dans "thisone".

GET_BIT_FROM_DINT (version 3.0 et sup rieure)

Extrait un bit d'un double nombre entier avec signe (DINT). Le DINT sur lequel l'opération doit être effectuée et le numéro du bit à extraire sont donnés comme des arguments à la fonction. Le résultat est une valeur booléenne (BOOL).



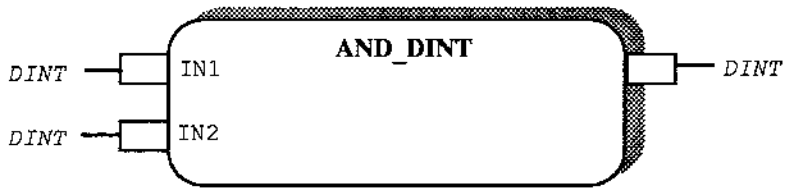
Exemple ST:

```
bool.Val := GET_BIT_FROM_DINT ( VALUE := dint.Val,
                                BIT_NO := 3 ) ;
```

si le troisième bit du dint est mis à 1 (la valeur 8 fait partie de la composition binaire du dint), la bool.Val est vraie (1).

AND_DINT (version 3.0 et sup rieure)

Permet de faire une intersection logique entre deux doubles nombres entiers (DINTs), bit par bit. Chaque bit qui est vrai dans les deux nombres entiers sera réglé dans la résultante de cette fonction et chaque bit qui n'est vrai que dans l'un ou l'autre des nombres entiers d'entrée n'est pas réglé dans la résultante.



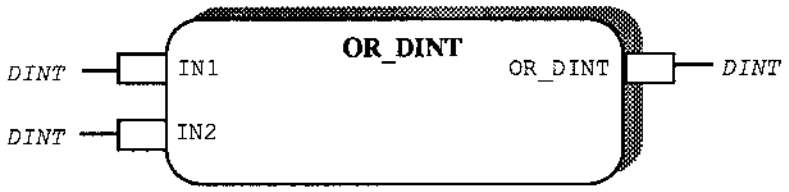
Exemple ST:

```
dintres.Val := AND_DINT( IN1 := dint1.Val ,
                         IN2 := dint2.Val );
```

si dint1 est 100 (bits 2, 5 et 6 mis à un) et dint 2 est 90 (bits 1, 3, 4 et 6 mis à 1), les dintres sont 64 (bit 6 mis à 1, binaire 1000000).

OR_DINT

Permet de faire une réunion logique, non exclusive, entre deux doubles nombres entiers (DINTs), bit par bit. Chaque bit vrai dans l'un ou dans les deux nombres entiers est réglé dans la résultante de cette fonction et tout bit qui n'est pas réglé dans l'un ou l'autre des nombres entiers entrés n'est pas réglé dans la résultante.



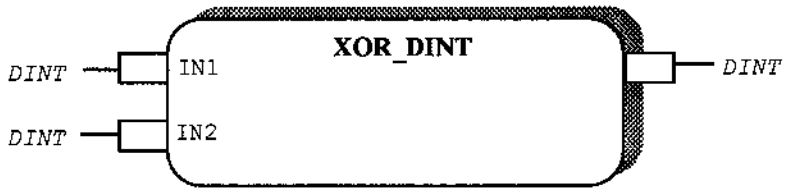
Exemple ST:

```
dintres.Val := OR_DINT( IN1 := dint1.Val ,
                        IN2 := dint2.Val );
```

si dint1 est 100 (bits 2, 5 et 6 réglés) et dint2 est 90 (bits 1, 3, 4 et 6 réglés), alors les dintres sont 126 (bits 1 à 6 réglés, binaire 1111110).

XOR_DINT

Permet de faire une réunion logique, exclusive, entre deux doubles nombres entiers (DINTs), bit par bit. Chaque bit vrai dans l'un ou l'autre mais pas dans les deux nombres entiers entrés est réglé dans la résultante de cette fonction et tout bit qui n'est pas réglé dans l'un ou l'autre des nombres entiers entrés n'est pas réglé dans la résultante.



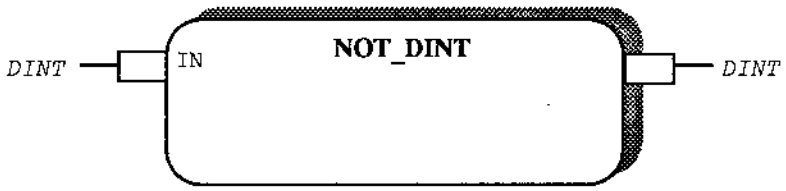
Exemple ST:

```
dintres.Val := XOR_DINT( IN1 := dint1.Val ,
                         IN2 := dint2.Val );
```

si dint1 est 100 (bits 2, 5 et 6 réglés) et dint2 est 90 (bits 1, 3, 4 et 6 réglés), le dintres est 62 (bits 1 à 5 réglés, binaire 111110).

NOT_DINT

Permet d'attribuer une fonction logique "NOT" à un nombre seul entier double (DINT). Tout bit qui est vrai dans le nombre entier entré n'est pas réglé dans la résultante de cette fonction et tout bit qui n'est pas réglé dans le nombre entier entré est réglé dans la résultante.



Exemple ST:

```
dintres.Val := NOT_DINT( IN := dint.Val ) ;
```

si dint est 100 (bits 2, 5 et 6 réglés), les dintres sont -101 (bits 0, 1, 3, 4 et 7 à 31 réglés, binaire 11111111111111111111111110011011).

ANNEXE A VITESSES D'EXECUTION DES FONCTIONS

La vitesse d'exécution des fonctions varie en fonction des opérandes des fonctions, de la longueur des chaînes, etc. Ces valeurs doivent être considérées comme étant approximatives. Prises avec les données de l'annexe B, elles peuvent servir à estimer la vitesse d'exécution d'un bloc donné de Texte structuré.

FONCTION ET VITESSE D'EXECUTION (µS)		FONCTION ET VITESSE D'EXECUTION (µS)	
NUMERIQUE		AX_REAL	38
ABS_REAL	34	MIN_DINT	5,1
ABS_DINT	3,7	MIN REAL	38
SQRT	84	**STRING**	
LN	115	EQUAL	100
LOG	119	LEN	3,4
EXP	100	LEFT	61
MOD	46	RIGHT	67
EXPT	136	MID	57
SIN	96	CONCAT	103
COS	97	INSERT	18
TAN	99	DELETE	21
ASIN	93	REPLACE	115
ACOS	93	FIND	236
ATAN	95	JUSTIFY_LEFT	126
SELECTION		JUSTIFY_RIGHT	123
SEL_BOOL	5,6	JUSTIFY_CENTRE	179
SEL_DINT	4,6	**CONVERSION DE TYPE**	
SEL_REAL	5,9	DINT_TO_REAL	46
SEL_TIME	4,7	REAL_TO_DINT	313
SEL_DATE	4,7	TRUNC	59
SEL_DATE_OF_DAY	4,7	TIME_TO_REAL	111
SEL_DATE_AND_TIME	4,7	REAL_TO_TIME	121
SEL_STRING	200[1]	TIME_TO_UDINT	2,4
MAX_DINT	5,1	UDINT_TO_TIME	2,6
DATE_AND_TIME_TO_TOD	79	**ARITHMETIQUE SUR LE TEMPS**	
DATE_AND_TIME_TO_DT	102	ADD_DATE_AND_TIME_T	82
CONCAT_TO_DINT	3,8	SUB_DATE_AND_TIME_T	82

FONCTION ET VITESSE D'EXECUTION (µS)		FONCTION ET VITESSE D'EXECUTION (µS)	
CONVERSION DE CHAINE		SUB_DATE_AND_TIME_	35
STRING_TO_DINT	294	ADD_TOD_TIME	112
HEX_STRING_TO_UDINT	155	SUB_TOD_TIME	188
BIN_STRING_TO_UDINT	159	SUB_TOD_TOD	46
OCT_STRING_TO_UDINT	149	**COMPACT**	
STRING_TO_REAL	1376	EXT_BOOL_FROM_STR	216
STRING_TO_TIME	936	REP_BOOL_IN_STR	132
HMS_STRING_TO_TIME	251	EXT_SINT_FROM_STR	87
DHMS_STRING_TO_TIME	323	REP_SINT_IN_STR	115
STRING_TO_DATE	447	EXT_INT_FROM_STR	93
EURO_STRING_TO_DATE	322	REP_INT_IN_STR	129
US_STRING_TO_DATE	322	EXT_DINT_FROM_STR	99
STR_TO_TIME_OF_DAY	230	REP_DINT_IN_STR	152
DINT_TO_STRING	379	EXT_USINT_FROM_STR	84
UDINT_TO_HEX_STRING	143	REP_USINT_IN_STR	115
UDINT_TO_BIN_STRING	184	EXT_UINT_FROM STR	91
UDINT_TO_OCT_STRING	161	REP_UINT_IN_STR	129
REAL_TO_STRING	1600	EXT_UDINT_FROM_STR	99
TIME_TO_STRING	1401	REP_UDINT_IN_STR	152
TIME_TO_HMS_STRING	481	EXT_REAL_FROM_STR	118
TIME_TO_DHMS_STRING	670	REP_REAL_IN_STR	179
DATE_TO_STRING	680	EXT_TIME_FROM_STR	99
DATE_TO_EURO_STRING	602	REP_TIME_IN_STR	152
DATE_TO_US_STRING	602	EXT_DT_FROM_STR	110
TIME_OF_DAY_TO_STR	443	REP_DT_IN_STR	152
ASCII_TO_CHAR	14	EXT_INT_FROM_STR	93
CHAR_TO_ASCII	6,3	REP_INT_INSTR_X_	129
EXT_DINT_FROM_STR_X	99	**SUPPLEMENT COMMUNICATIONS**	
REP_DINT_IN_STR_X	152	IEEE_STRING_TO_REAL	124
EXT_UINT_FROM_STR_X	91	REAL_TO_IEEE_STRING	90
REP_UINT_IN_STR_X	129		
EXT_UDINT_FROM_STR_X	100		
REP_UDINT_IN_STR_X	152		

Remarque 1 : valeur type, dépend de la longueur de la chaîne de caractères.

ANNEXE B VITESSES D'EXECUTION DES OPERATEURS

Le tableau 1 ci-dessous montre la vitesse d'exécution des opérateurs :

Tableau 1 Vitesses d'exécution

Opérateur	Opérande réel µS	Opérande entier µs	Opérande booléen µs
+	66	2,6	S/O
x	66	23	S/O
-	66	2,6	S/O
/	72	43	S/O
>	31	6,5	S/O
<	31	5,7	S/O
=	31	5,8	S/O
<>	31	6,8	S/O
>=	31	6,8	S/O
<=	31	5,8	S/O
- (unaire)	8	3,7	S/O
mod	S/O	46	S/O
et	S/O	S/O	6
ou	S/O	S/O	6
ou exclusif	S/O	S/O	5
non	S/O	S/O	5

Remarque : évaluation d'une expression du genre :

```
reall.Val := real12.Val + real13.Val ;
```

Des temps système supplémentaires sont associés à la conversion des opérandes réels de la structure "précision simple" à la structure "précision double" avant l'évaluation. La valeur "précision double" résultante, reall.Val, est ensuite reconvertie en structure à virgule flottante "précision simple".

Cette conversion est effectuée parce que, en interne, toutes les instructions en ST sont converties en "C" avant d'être compilées. Le compilateur utilisé dans la station de programmation ne traite pas la structure"précision simple". Tous les calculs sont effectués à l'aide de la structure "précision double".

Les temps système supplémentaires sont les suivants :

Opération :	ajout
structure simple en structure double	17µs
structure double en structure simple	30µs

Dans l'exemple indiqué, la durée totale est la suivante :

```
real1.Val := real12.Val + real13.Val ;
```

30µs	17µs	66µs	17µs
(double en simple)	(simple en double)	(+)	(simple en double)

Total : 30 +17+66+17 = 130µs

Il faut noter que toutes les conversions s'effectuent automatiquement. Ces informations sont fournies pour permettre au développeur de programmes de calculer la vitesse d'exécution dans une partie donnée de texte structuré.

Tableau 2 Ordre de priorité des opérateurs

No.	Opération	Symbole	Priorité
1	Mise entre parenthèses Fonction Evaluation	(expression) identificateur (liste d'arguments) par exemple LN(A), MAX(X,Y) etc.	MAXIMALE
2	Elévation à une puissance	**Remarque : A**B=EXP(B*LN(A))	
3	Négation Complément	- NON	
4	Multiplication Division Modulo	* / MOD	
5	Addition Soustraction Comparaison	+ - <, >, <=, >=	
6	Egalité Inégalité	= <>	
7	ET booléen	&, ET	
8	OU exclusif booléen	XOU	
9	OU booléen	OU	MINIMALE

ANNEXE C CODES D'ERREUR

Le PC3000 offre une consignation des erreurs système pour suivre les événements en temps réel comme les pannes de courant, le redémarrage du système, les défauts de communications et d'autres informations de diagnostic. Tous les codes d'erreur portent l'heure à laquelle ils se sont produits.

Les codes d'erreur associés aux opérateurs mathématiques sont également produits et sont saisis dans le journal de consignation en temps réel. Les erreurs mathématiques peuvent être la division par zéro ou le dépassement de capacité. Ce type d'erreur peut se produire dans l'exécution des blocs fonctions, le câblage par soft ou le programme séquentiel du programme utilisateur.

Dans tous les cas, l'entrée dans le journal de consignation est représentée sous la forme

<5xx> <code d'opération mathématique> <informations de diagnostic>

Les codes d'opérations mathématiques sont regroupés séparément dans un tableau et indiquent le type d'opération à l'origine de l'erreur.

Les informations de diagnostic sont destinées à une utilisation en interne par Eurotherm Automation.

En cas d'erreurs, il faut chercher dans le programme utilisateur les conditions qui donnent des valeurs interdites ou des opérations impossibles comme la division par zéro.

Tableau 1 Description des codes d'erreur

Code d'erreur	Description de l'erreur	Champ1	Champ 2
500	**Erreur** Une opération mathématique comportant une valeur à virgule flottante (REAL) a donné un résultat infini. **Cause** Une opération mathématique comme la division par zéro a été effectuée et a donné un résultat infini. Une fois que l'on a obtenu une infinité, elle peut se propager à de nombreuses opérations mathématiques et provoquer ainsi des erreurs multiples.	Cf. tableau 2	

Code d'erreur	Description de l'erreur	Champ1	Champ 2
501	**Erreur** Une opération mathématique REELLE a été effectuée avec un opérande interdit. **Cause** Une opération mathématique a été effectuée sur une valeur interdite comme ASIN (IN :=2). Cette opération entraîne la production d'une valeur interdite comme IEEE "ceci n'est pas un nombre" ou NAN (identique à une infinité). Cette erreur peut se propager par de nombreuses opérations mathématiques et provoquer la signalisation de nombreuses erreurs.	Cf. tableau 2	Informations de diagnostic
502	**Erreur** Une opération mathématique sur un REEL a provoqué un dépassement de capacité. **Cause** Une opération mathématique qui donne une valeur trop importante pour être représentée a été effectuée. Cette opération donnera une infinité IEEE et risque de se propager par de nombreuses opérations mathématiques et de provoquer la signalisation de nombreuses erreurs.	Cf. tableau 2	Informations de diagnostic
550	**Erreur** Division d'un ENTIER par zéro. **Cause** Il y a eu une tentative de division d'un entier par zéro.	0	Informations de diagnostic

Tableau 2 Codes d'opérations mathématiques

Code de l'opération	Description de l'opération
1	conversion double-entier
2	conversion précision double-précision simple
3	conversion simple-entier
11	addition simple précision-entier
12	soustraction simple précision
13	multiplication simple précision
14	division simple précision
21	addition double précision
22	soustraction double précision
23	multiplication double précision
24	division double précision
25	carré double précision
26	1n double précision
27	log double précision
28	exp [1] double précision
29	sin double précision
30	cos double précision
31	tan double précision
32	asin double précision
33	acos double précision
34	atan double précision
35	mod double précision
36	exp [2] double précision

Remarque 1. Exposant naturel

Remarque 2. Elévation à une puissance

Remarque 3. Dans le tableau 2, précision "simple" et précision "double" désignent les mots 32 et 64 bits servant à mémoriser les valeurs à virgule flottante IEEE. Le PC3000 utilise ces deux types de format IEEE des nombres dans différentes parties du programme utilisateur.

ANNEXE D REGLES APPLICABLES AUX DONNEES DE TYPES DIFFERENTS

Le PC3000 prend en charge un sous-ensemble des types de données définis par IEC DIS 1131 dans le cadre de la définition du langage Texte structuré (ST). Ces types de données s'appliquent aux paramètres des blocs fonctions et aux variables créées par l'utilisateur à l'aide de la classe de bloc fonction USER_VAR.

Le mot-clé est le terme utilisé pour décrire un groupe de caractères qui ont une signification spéciale en Texte structuré. Les types de données acceptés sont les suivants :

Mot-clé	Type de données	Bits	Plage
IO_ADDRESS	Adresse de canal	32	1:01à 12:14 (remarque 2)
BOOL	Booléen	1	0 ou 1
REAL	Nombre réel	32	$\pm 10^{\pm 38}$
SINT	Entier court	8	-128 à 127
USINT	Entier court sans signe	8	0 à 255
INT	Entier	16	-32768 à 32767
UINT	Entier sans signe	16	0 à 65535
DINT	Entier double	32	-2147483648 à 2147483647
UDINT	Entier double sans signe	32	0 à 429449967295
TIME	Durée	32	Jusqu'à 49 jours
TIME_OF_DAY DATE	Heure du jour Date	32	00:00:00 à 23:59:59 01-Jan-1970 au 01-Jan-2136
DATE AND TIME	Date ET heure du jour	32	01-Jan-1970-00:00:00 au 01-Jan-2136-23:59:59
STRING	Chaîne de caractères de longueur variable		(Remarque 3)
ENUM	Elément énuméré	32	0 à 2^{32} (remarque 4)

Remarque 1. Bien que les données de type SINT, USINT, INT et UINT soient toutes répertoriées, elles sont traitées comme étant 32 bits dans le PC3000, ce qui garantit la compatibilité avec les autres produits.

Remarque 2. IO_ADDRESS est un type de données spécial servant à identifier l'adresse d'une canal matériel d'entrée ou de sortie. Il prend la forme :

```
<RACK>:<EMPLACEMENT>:<CANAL>
(1-8)  (1-12)  (1 au nombre maximal de canaux):
```

MODULE Les blocs fonctions possèdent une IO_ADDRESS représentée sous la forme d'un type de données Chaîne de caractères (String).

Remarque 3 : Longueurs de chaînes : une variable utilisateur **USER_VAR** 'String' (chaîne de caractères) est limitée à 80 caractères. Une variable utilisateur USER_VAR 'Long String' (chaîne de caractères longue) peut comporter un maximum de 255 caractères. Ces deux variables sont caractérisées sous le type de données STRING (chaîne de caractères).

Remarque 4 : les types de données énumérées apparaissent sur les listes où l'on peut sélectionner un nombre fixe d'options. On peut aussi choisir de signaler un certain nombre d'états à l'aide de ce type de données, avec la forme :

Nom 1 (0)

Nom 2 (1)

etc.

L'addition, la soustraction, etc. des paramètres énumérés sont interdites.

Expressions

Une expression est un élément qui, lorsqu'il est évalué, produit une valeur correspondant à un des types de données répertoriés.

Les expressions comportent des opérateurs (+, -, *", etc.) et des opérandes qui peuvent être des constantes texte (2,5, 3, T#2s, etc.), des variables (int1.Val, real11.val, etc.) ou un paramètre Bloc fonction, une fonction ou une autre expression.

Dans la norme IEC 1131, tous les types de données font l'objet d'un contrôle strict du type dans toutes les expressions. Cela signifie que toute expression doit contenir des types de données "appariés" ou qu'il faut convertir les types de données à l'aide des fonctions de conversion de type.

Le PC3000 déroge à ces règles de la manière suivante :

A. `real1.Val := real2.Val opérateur Int.1.Val;`

   Lorsque `l'opérateur est *, +, - ou /` produit une valeur à virgule flottante (REAL)

B. `Int1.Val:=realA.Val opérateur Int1.Val;`

   Lorsque `l'opérateur est *, +, - ou /` produit une valeur entière tronquée.

Il est toutefois bon de s'assurer que toutes les données d'une expression ont un type générique identique.

Cela signifie que l'exemple A donnerait :

C.
```
real1.Val:=  real2.Val  opérateur  DINT_TO_REAL
      (IN:=  Int1.Val);
```
produit une valeur à virgule flottante (REAL).

Expressions comportant des données de type temporel

L'utilisation des opérations arithmétiques normales du genre ci-après n'est pas prise en charge :

```
time1.Val  :=  time1.Val  +  time2.Val
```

Il faut utiliser les fonctions de conversion du temps dans les expressions pour lesquelles des opérations mathématiques s'imposent entre les types temporels. Exemple :

```
time1.Val  :=   UDINT_TO_TIME(IN:=
                TIME_TO_UDINT(IN:=  time1.Val)  +
                TIME_TO_UDINT(IN:=  time2.Val);
```

Expressions

Les règles sont identiques à celles illustrées pour l'addition et la soustraction. Il faut convertir le temps dans une structure entière avant l'évaluation dans une expression comportant plusieurs paramètres temporels.

ANNEXE E IMBRICATION DES FONCTIONS ST DU PC3000

Il est possible d'imbriquer les fonctions pour créer des stratégies de régulation complexes comme le basculement automatique d'un thermocouple "de secours" pour assurer une protection contre une panne de détecteur.

En outre, il est possible d'utiliser des fonctions de conversion de type de manière répétée pour apparier les types de données dans une expression.

Il est possible d'imbriquer les fonctions associées à la manipulation des chaînes de caractères pour créer des chaînes de message concaténées destinées à une utilisation avec un panneau par exemple. Il est toutefois conseillé de limiter l'imbrication des fonctions de chaînes à deux niveaux car les fonctions de chaînes utilisent des parties importantes de l'espace d'empilage du PC3000. Il est possible d'obtenir une profondeur d'imbrication supérieure en utilisant le bloc fonction chaîne USER_VAR pour stocker les valeurs intermédiaires.

Les exemples ci-après illustrent l'imbrication :

Exemple 1.

Basculement automatique de thermocouple

```
loop1.Process_Val  :=SEL_REAL
            (G:=level1.Status=1 (*GO*),
            IN0:=SEL_REAL
                  (G:=level2.Status  =  1  (*GO*),
                  IN0:=loop1.Process_Val,
                  IN1:=level2.Process_Val),
            IN1:=level1.Process_Val);
```

L'entrée du bloc fonction PID, loop1, provient d'une entrée analogique parmi deux (celle qui a l'état GO). Si les deux capteurs sont sur NOGO, l'entrée PID est forcée pour conserver sa valeur antérieure.

Exemple 2.

Conversion de type

```
time1.Val:=      UDINT_TO_TIME(IN:=
                 TIME_TO_UDINT(IN:=  time1.Val)  +
                 TIME_TO_UDINT(IN:=  time2.Val);
```

Les fonctions de conversion de type servent à convertir les durées (TIME) en entiers (UDINT) avant l'addition. Le résultat est ensuite converti en une durée (TIME).

Exemple 3.

Fonctions chaîne

```
Display.Val:=   CONCAT(IN1:='Temperature of sample 1 is',
                       IN2:=REAL_TO_STRING
                       (IN1:=INPUT1.Process_Value));
```

Display.Val est une variable chaîne servant à contenir la chaîne concaténée. On peut l'utiliser pour créer un affichage de tableau opérateur.

EUROTHERM AUTOMATION SERVICE REGIONAL

SIÈGE SOCIAL ET USINE
6 chemin des Joncs
BP 55
69572 Dardilly Cedex

Tél. : 04 78 66 45 00
Fax : 04 78 35 24 90

AGENCES

Aix-en-Provence
Tél.: 04 42 39 70 31

Colmar
Tél.: 03 89 23 52 20

Lille
Tél.: 03 20 96 96 39

Lyon
Tél.: 04 78 66 45 10
04 78 66 45 12

Nantes
Tél.: 02 40 30 31 33

Paris
Tél.: 01 69 18 50 60

Toulouse
Tél.: 05 61 71 99 33

BUREAUX

Bordeaux
Clermont-Ferrand
Dijon
Grenoble
Metz
Normandie
Orléans

L'évolution de nos produits peut amener le présent document à être modifié sans préavis.

Imprimé en France 09/96 Manuel PC3000 Fonctions - HA 174 518 - Indice 1