EUROTHERM
CONTROLS

PC3000 FUNCTIONS REFERENCE

Handbook

Printed in England 0198

HA022916 Issue 3.1

CONTENTS

PREFACE

This reference manual defines the functions available on the PC3000 Programming Station that can be used within Structured Text (ST) statements for defining 'soft wiring' between Function Blocks and within Step and Transition definitions. The reader should be familiar with the PC3000 Programming Station and the Structured Text language. The PC3000 User Guide provides an introduction to the language and programming concepts.

Information contained in this document provides definitions and examples for all PC3000 functions and can be used for reference when developing PC3000 applications.

For further information refer to the PC3000 Reference documents:

PC3000 Hardware Reference - provides detailed information on all the PC3000 hardware modules including calibration, wiring and physical configuration details. Part No. HA022919

PC3000 Real Time Operating System Reference - describes the operation and the functionality of the PC3000 system software that manages the loading, initialisation and execution of a User Program.

PC3000 Function Block Reference - describes the numerous function blocks available to be incorporated into your control program for PID control, Ramps, Counters, Filters, Timers etc. Part No.HA022917

FIRMWARE COMPATIBILITY

Functions	Firmware versions 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	●	●	●

Functions	Firmware versions 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	●	●	●

Functions	Firmware versions 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			●

Chapter 1

INTRODUCTION

Edition 1

Contents

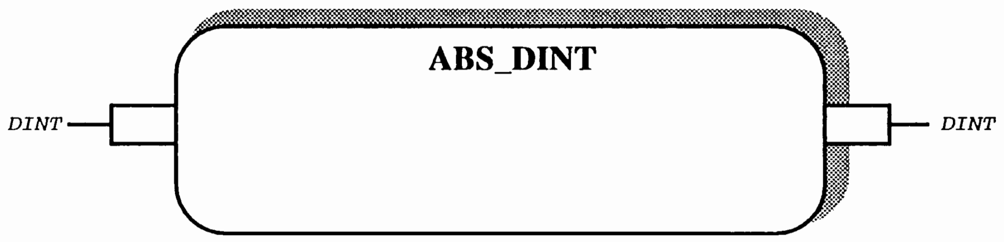
Overview

Structured Text (ST) functions are executed as part of the body of Structured Text in which they are called. They all return a single value which is usually assigned to a Function Block parameter or used within an ST expression. There are no side-effects, e.g. other parameters are not modified and the value returned is not stored unless explicitly assigned to a specific parameter.

The ST functions may be nested, e.g. type conversion functions may be used to match data types within an expression and the overall result then converted back to appropriate data type. See Appendix E for examples.

General form

All PC3000 functions are represented within this text in the form:



Functions have a single output which is shown on the right-hand side of the diagram.

Functions may have any number of inputs which appear on the left-hand side of the diagram. When more than one input is used they are each individually named, e.g. `G, IN0, IN1` etc. inside the function box. Input and output data types, i.e.. *REAL, BOOL, STRING* etc. are listed adjacent to the input or output to which they refer. The function name is stated inside the box as shown.

> **Note:** Single input functions have an input parameter IN which should be assigned a value or expression when the functions are called in ST. However, to comply with the IEC 1131-3 standard, the name of the single input parameter is not shown on function diagrams.

In the function parameter descriptions the formal IEC 1131-3 standard data types are shown in parenthesis after the data type description, e.g. floating point (REAL).

Limitations and error detection

A number of functions, including many of the numeric functions, have restrictions on the range of parameter values that can be used. In a number of cases illegal values will generate one or more system errors.

For example, the SQRT square root function of a negative number will generate a system error and will result in the SQRT function producing a "IEEE Not a Number" or NAN as the result. This may propagate through an expression and generate a number of system errors by other operators or functions. In some cases functions will generate an "IEEE infinity" as a result which when used in other expressions may also generate system errors.

All system errors are detailed in the 'PC3000 Real Time Operating System Reference'. The error codes that apply to ST functions are listed in Appendix C. Care should be taken when developing a user program to ensure that only legal function parameter values are used.

A number of functions only yield valid output values when the input parameters have values that lie within a specified range i.e. within particular maximum and minimum values.

In the following function descriptions, where parameter range limitations exist they are defined; otherwise the normal parameter range can be assumed.

Caution

1. A number of functions, particularly the string functions, have unsigned integer parameters (i.e. USINT unsigned short integer, and UDINT unsigned double integer) for lengths, positions etc. Care should be taken to ensure that an integer parameter, such as double integer DINT, with a negative value (i.e.< 0), is not assigned to parameters which are unsigned integers. There is a possibility that the unsigned integer parameter will assume a very large positive value.

2.Because string operations can use a large area of the PC3000 system stack space, care should be taken not to nest string functions more than 2 functions deep. If an application requires a complex sequence of string functions, it is preferable to use User Variable String function blocks to hold intermediate values.

Chapter 2

NUMERICAL FUNCTIONS

Edition 3

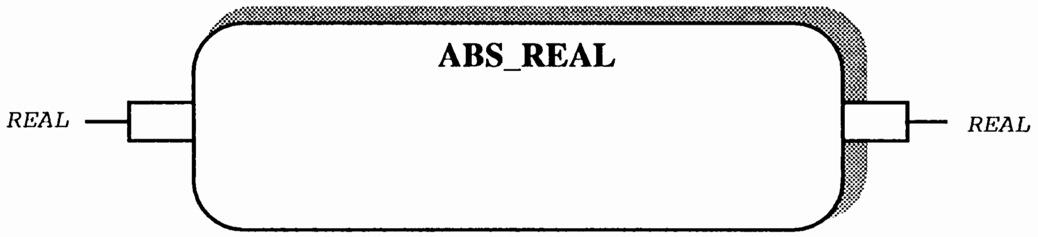
Contents

Overview

The Numerical functions include all common mathematical functions such as logarithm, exponential, trigonometric functions, sine, cosine etc.

ABS_REAL

The absolute value of a floating point (REAL) input, i.e. negative values are treated as being positive.



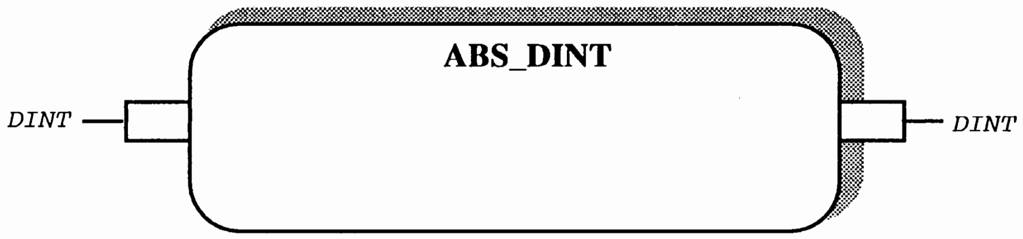
Example ST :

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

real1.Val is set to 100.3

ABS_DINT

The absolute value of an integer (DINT) input, i.e. negative values are treated as being positive.



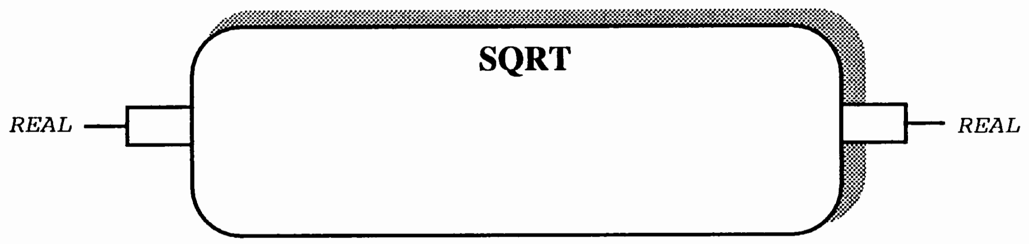
Example ST :

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

int1.Val is set to 100

SQRT

The square root of a floating point (REAL) value, only positive values are valid.



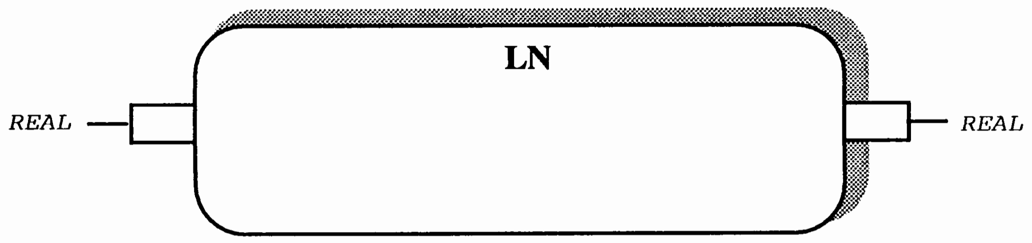
Example ST :

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

real1.Val is set to 10.0

LN

The natural logarithm of a floating point (REAL) value, only values greater than 0 are valid.



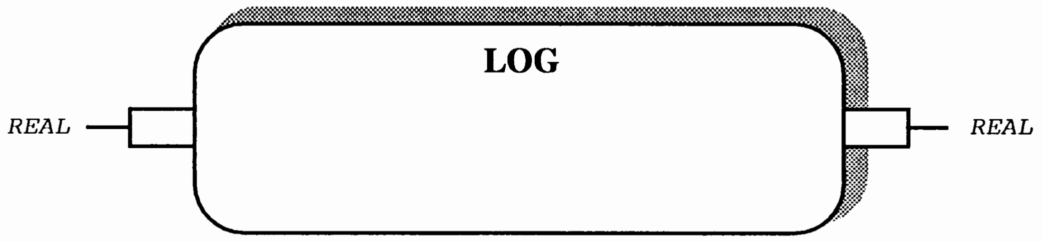
Example ST :

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

real1.Val is set to 0.89609

LOG

The logarithm to the Base 10 of a floating point (REAL) value; only values greater than 0 are valid.



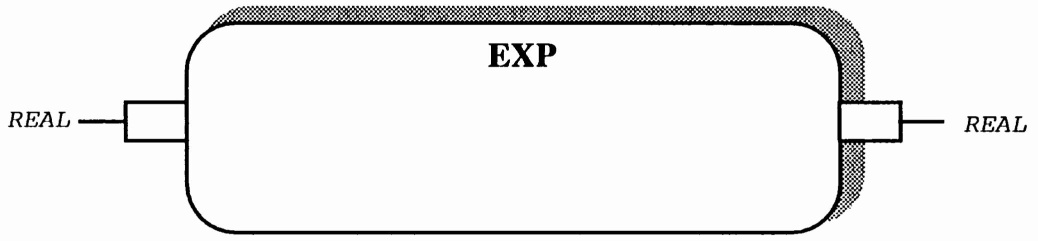
Example ST :

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

real1.Val is set to 2.0

EXP

Natural Exponential of a floating point (REAL) value i.e. e^x ; both positive and negative values can be used.



Example ST :

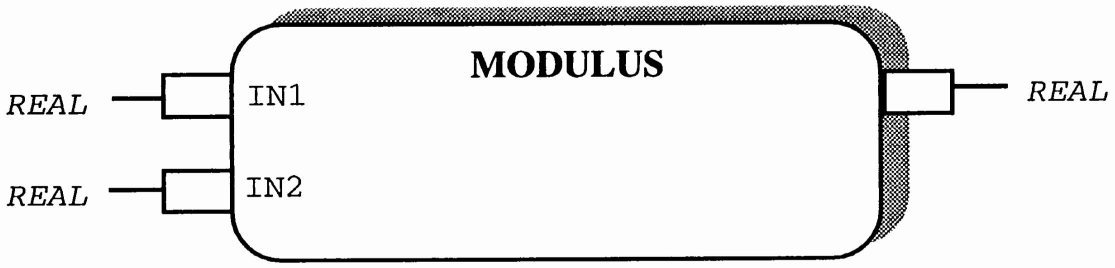
```
real1.Val:=  EXP(IN:=  2.0);
```

real1.Val is set to 7.3890

MODULUS

Prior to version 2.27 this function was called MOD.

Modulo of a floating point (REAL) value where the dividend value IN1 is divided by divisor IN2 and the remainder is returned. The divisor IN2 should not be 0.



Example ST:

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

real1.Val is set to 2.5

Example ST:

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

real1.Val is set to 0

Example ST:

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

real1.Val is set to -3.0

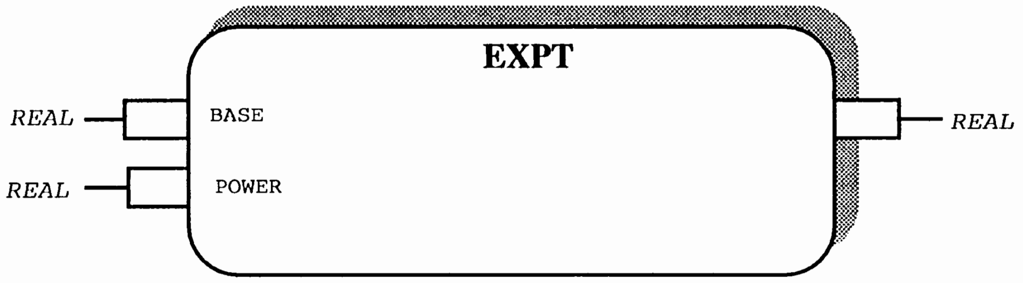
Example ST:

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

real1.Val is set to 1.0

EXPT

Exponentiation, raises the value of BASE to a given POWER, i.e. $BASE^{POWER}$. This is equivalent to $e^{POWER * \ln(BASE)}$. The BASE value should always be positive.



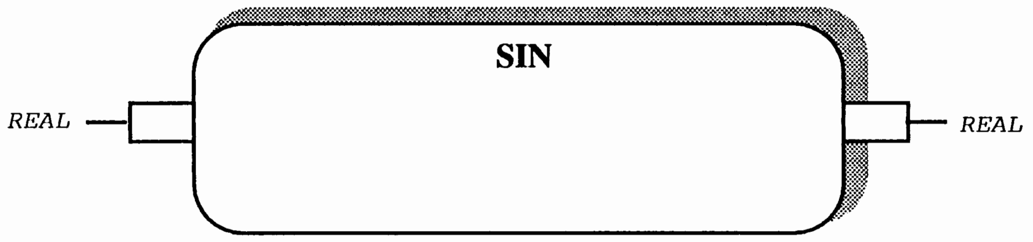
Example ST:

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

real1.Val is set to 8.0

SIN

Sine of a floating point (REAL) value input in radians.



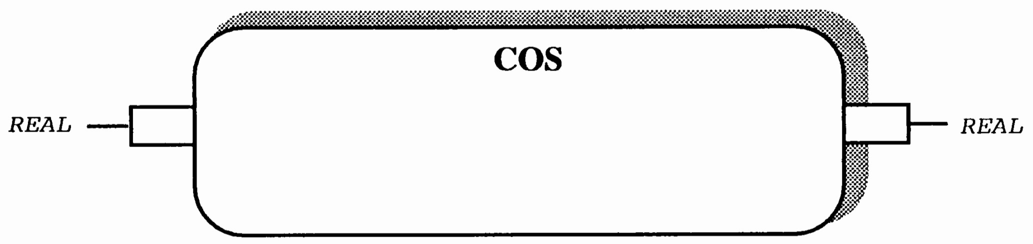
Example ST :

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

real1.Val is set to 0.527

COS

Cosine of a floating point (REAL) value in radians.



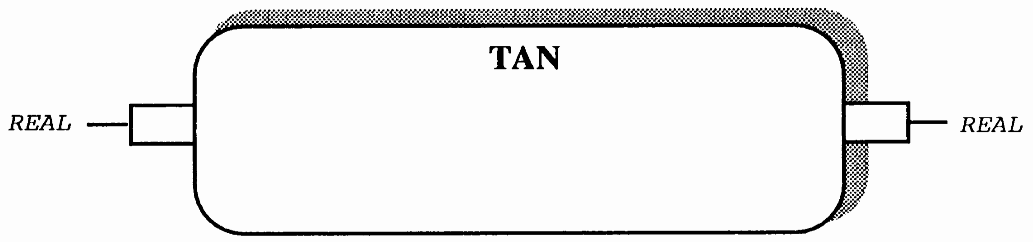
Example ST :

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

real1.Val is set to 0.849

TAN

Tangent of a floating point (REAL) value in radians.



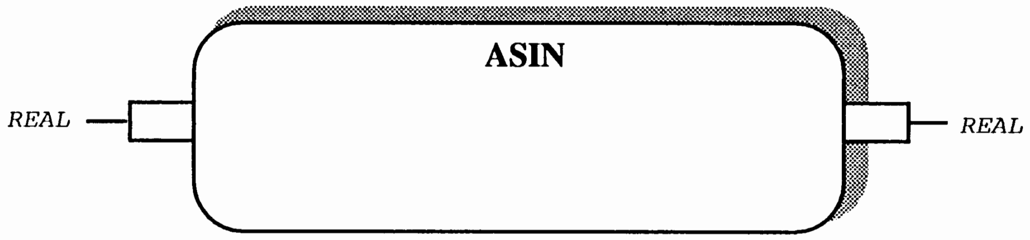
Example ST :

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

real1.Val is set to 0.859

ASIN

Principal Arc Sine, the floating point (REAL) input value should be within the range >= -1.0, <= 1.0. The output is in the range >=- π/2, <=π/2



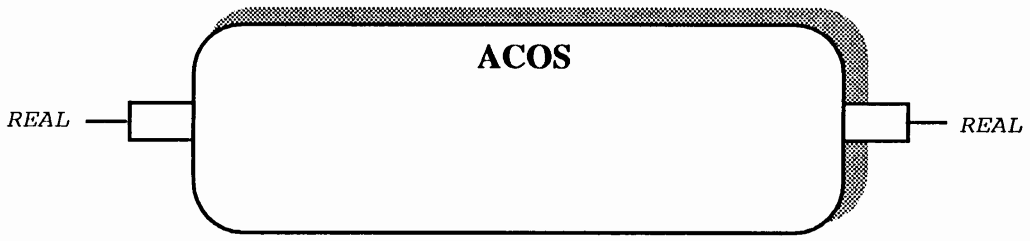
Example ST :

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

real1.Val is set to 0.3047

ACOS

Principal Arc Cosine, the floating point (REAL) input value should be within the range >= -1.0, <= 1.0. The output is in the range >=0, <= π



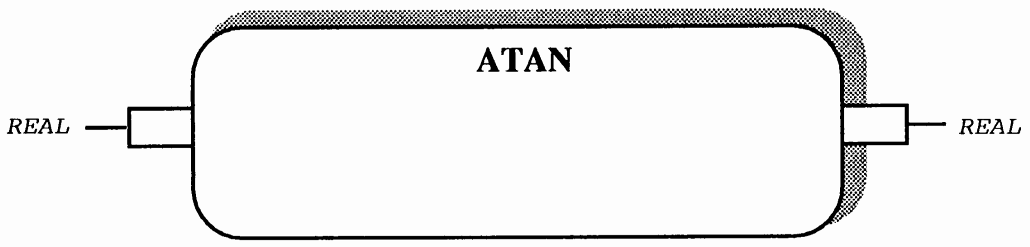
Example ST :

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

real1.Val is set to 1.266

ATAN

Principal Arc Tangent of a floating point (REAL) input value; no input limitations but output is within the range >= -π/2, <= π/2



Example ST :

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

real1.Val is set to 1.522

Chapter 3

SELECTION FUNCTIONS

Edition 1

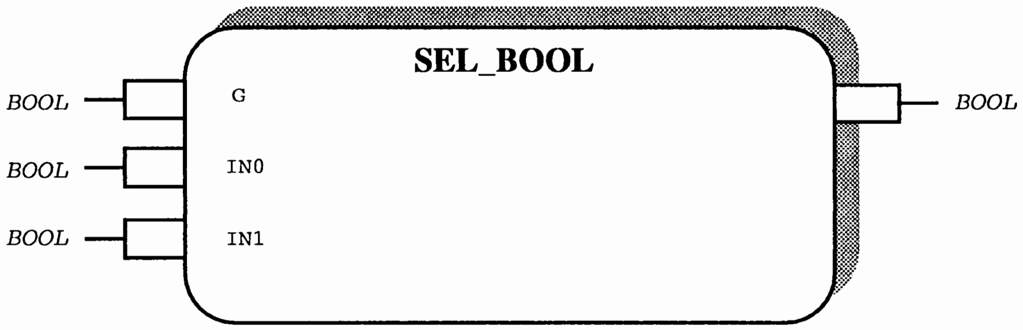
Contents

Overview

Selection Functions provide the selection of values depending on the state of either a boolean parameter or expression, or other criteria such as the maximum of two values. Selection functions are provided for a variety of different data types, e.g. SEL_REAL selects either of two floating point (REAL) values depending on a boolean expression being true or false.

SEL_BOOL

Selects between two boolean (BOOL) inputs IN0 and IN1 depending on the boolean value of G. The result is IN0 if the value of G is 0 (i.e. FALSE), otherwise it is the value IN1 for G is 1 (i.e. TRUE).



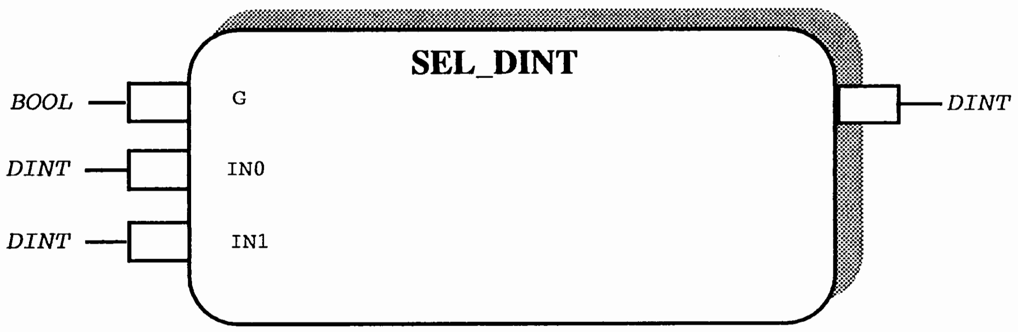
Example ST:

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

coolpump.Val is set to 0 (i.e.FALSE) if zone1.Process_Val is greater than 100.5.

SEL_DINT

Selects between two integer (DINT) inputs IN0 and IN1 depending on the boolean value of G. The result is IN0 if the value of G is 0 (i.e FALSE), otherwise it is the value IN1 for G is 1 (i.e. TRUE).



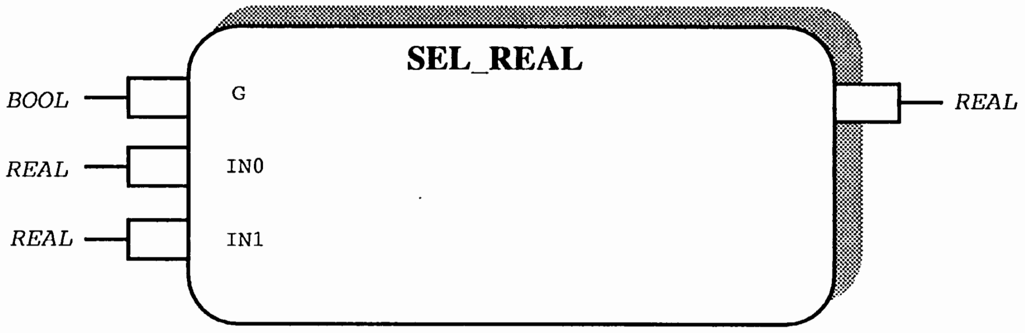
Example ST :

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

labelID.Val is set to 0 when batchNo.Val is 1999, otherwise it is set to batchNo.Val.

SEL_REAL

Selects between two floating point (REAL) inputs IN0 and IN1 depending on the boolean value of G. The result is IN0 if the value of G is 0 (i.e FALSE), otherwise it is the value IN1 for G is 1 (i.e. TRUE).



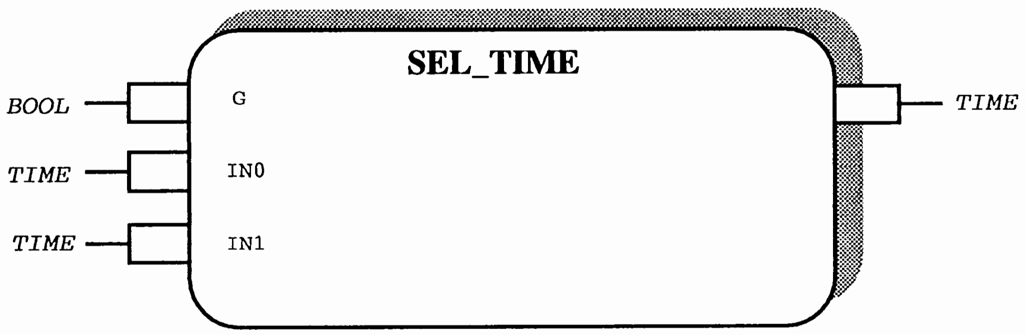
Example ST:

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

speed.Val is set to 23.11 if sensor.Val is 1 (i.e. TRUE), otherwise it is set to 10.23.

SEL_TIME

Selects between two duration (TIME) inputs IN0 and IN1 depending on the boolean value of G. The result is IN0 if the value of G is 0 (i.e FALSE), otherwise it is the value IN1 for G is 1 (i.e. TRUE).



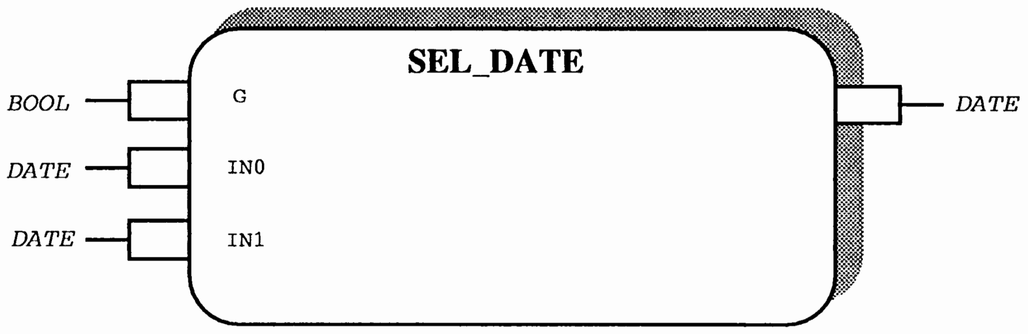
Example ST:

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

heatTime.Val is set to T#10s_500ms if procType value is 3, otherwise it is set to the value of time2.Val.

SEL_DATE

Selects between two date (DATE) inputs IN0 and IN1 depending on the boolean value of G. The result is IN0 if the value of G is 0 (i.e FALSE), otherwise it is the value IN1 for G is 1 (i.e. TRUE).



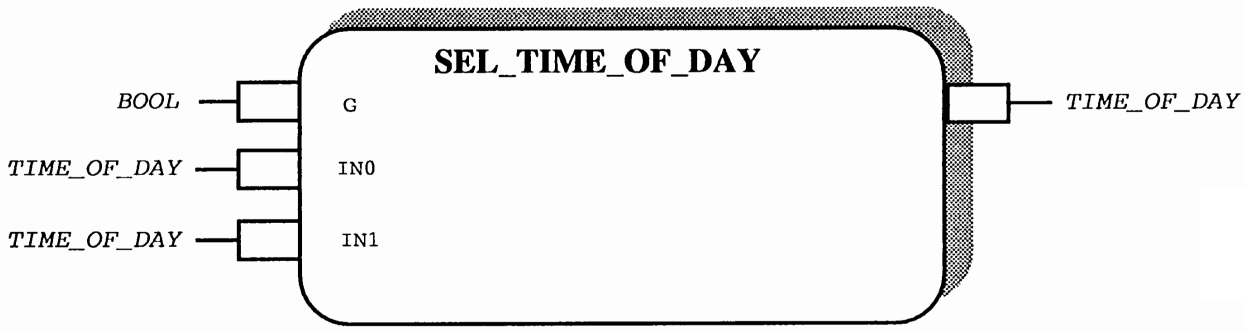
Example ST:

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

endDate.Val is set to value of specDate if jobType equals the recipe value, otherwise it is set to the value of normDate.Val.

SEL_TIME_OF_DAY

Selects between two time of day (TIME_OF_DAY) inputs IN0 and IN1 depending on the boolean value of G. The result is IN0 if the value of G is 0 (i.e FALSE), otherwise it is the value IN1 for G is 1 (i.e. TRUE).



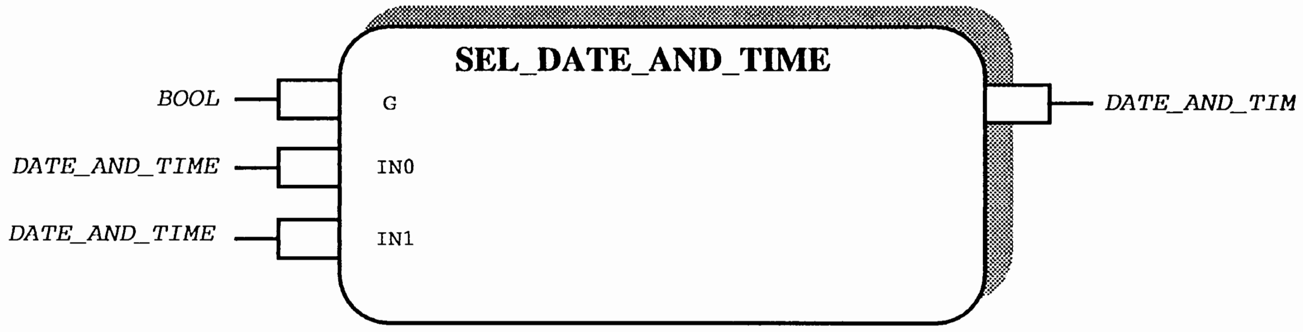
Example ST:

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

If RT_Clock.day is 1 (i.e. Monday) then startime.Val is set to 06:00:00, otherwise it is set to 08:00:00.

SEL_DATE_AND_TIME

Selects between two date and time (DATE_AND_TIME) inputs IN0 and IN1 depending on the boolean value of G. The result is IN0 if the value of G is 0 (i.e FALSE), otherwise it is the value IN1 for G is 1 (i.e. TRUE).



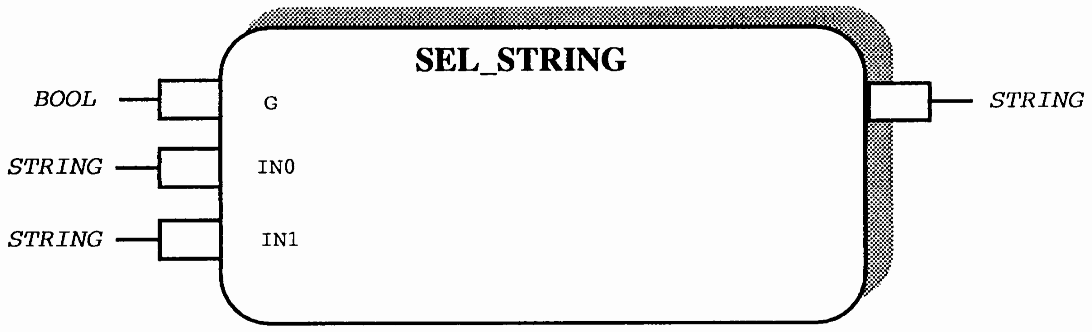
Example ST:

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

If logFlag.Val is TRUE then logDate.Val is set to RT_Clock.DateAndTime, otherwise it is set to nullDate.Val.

SEL_STRING

Selects between two character string (STRING) inputs IN0 and IN1 depending on the boolean value of G. The result is IN0 if the value of G is 0 (i.e. FALSE), otherwise it is the value IN1 for G is 1 (i.e. TRUE).



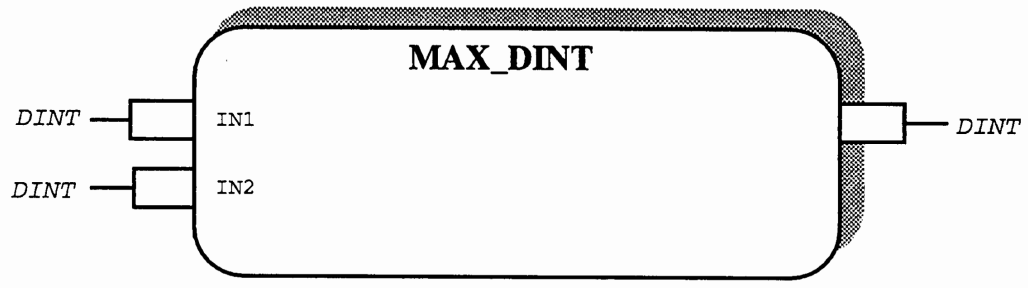
Example ST:

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

If Alarm.Val is TRUE then message.Val is set to 'ALARM!', otherwise it is set to 'No Alarm'.

MAX_DINT

Selects the maximum of two integer (DINT) inputs.



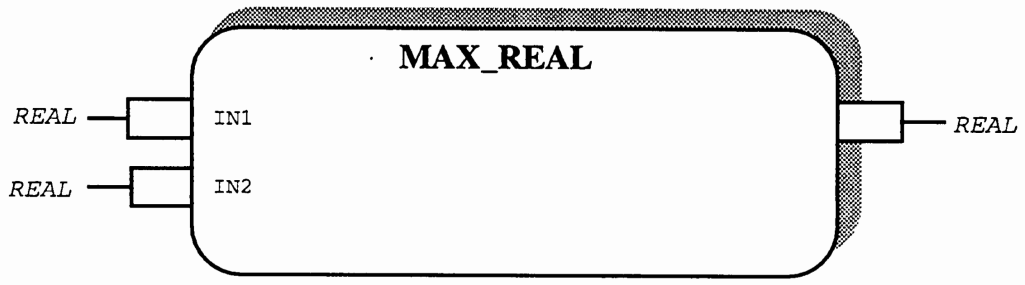
Example ST:

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

rate.Val is set to pulseCt.Val while pulseCt is greater than 30, otherwise rate.Val is set to 30.

MAX_REAL

Selects the maximum of two floating point (REAL) inputs.



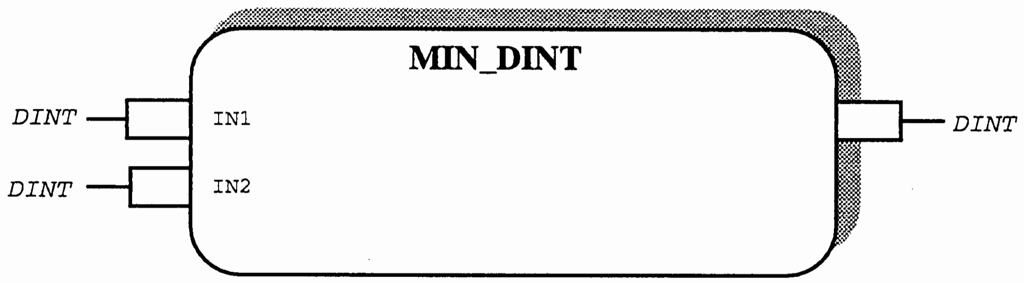
Example ST

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

tempMax.Val is set to the highest value of zone1, zone2 and zone3 process values.

MIN_DINT

Selects the minimum of two integer (DINT) inputs.



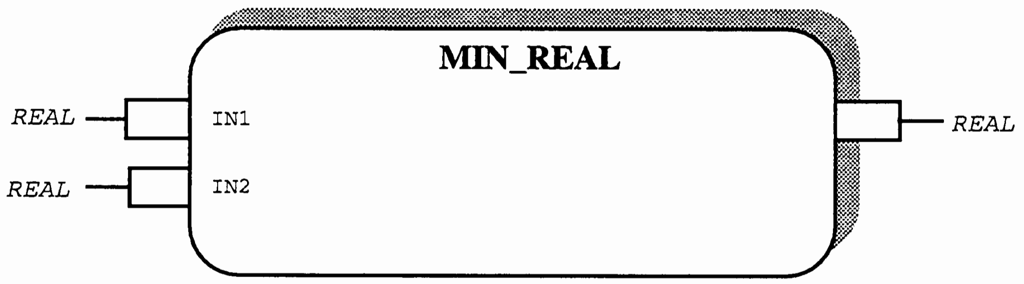
Example ST:

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

cntMin.Val is set to the minimum of count1.Val and count2.Val.

MIN_REAL

Selects the minimum of two floating point (REAL) inputs.



Example ST:

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

loop.Setpoint is set to working.Val limited between a minimum set by low.Val and a maximum set by high.Val.

Chapter 4

STRING FUNCTIONS

Edition 1

Contents

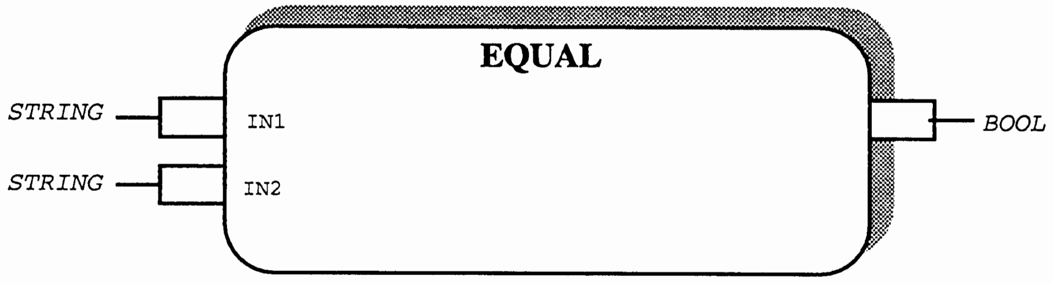
Overview

A very comprehensive range of character string functions is provided to allow strings to be constructed and manipulated. For example, it is possible to construct a message from a number of function block parameter values of different data types.

Typically these functions will be required for creating: panel messages, messages to printers for report generation, creation and decomposition of communications request and response messages. In the following string Functions, the character positions within a string shall be considered to be numbered 1, 2, ...L, where L is the total length of the string.

EQUAL

Checks for equality between two character strings (STRINGs)



Example ST:

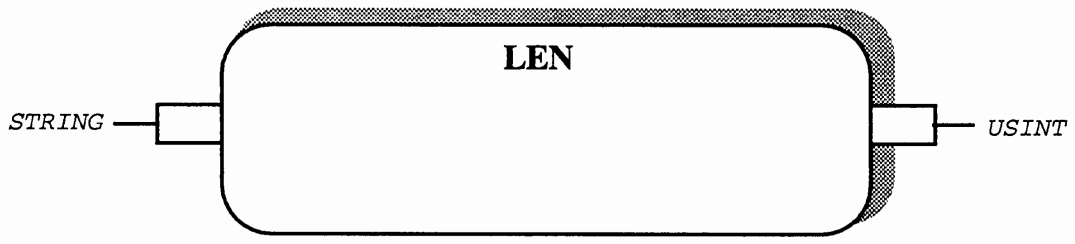
```
IF EQUAL (IN1:= ans.Val, IN2:= 'YES') THEN
            response.Val:= 10;
ELSE
            response.Val:= 0;
END_IF ;
```

If the string contained in ans.Val is identical to 'YES', response.Val is set to 10.

However for other values of ans.Val, such as ' YES', 'YEs' or ' YES ', response.Val is set to zero.

LEN

Returns the length of a character (STRING) string as an unsigned short integer (USINT).



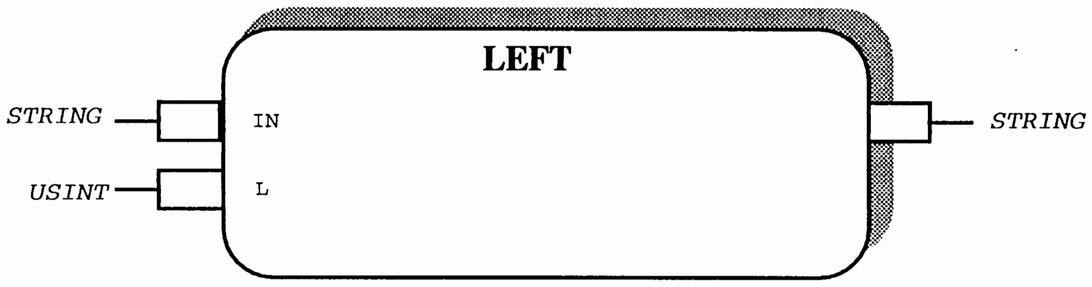
Example ST

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

A.Val is set to 8.

LEFT

Extracts a given number of characters L, from the left end of a character string (STRING) IN, i.e. from the beginning of the string. However, if parameter IN is less than L characters long, a null zero length string is returned i.e. ''.



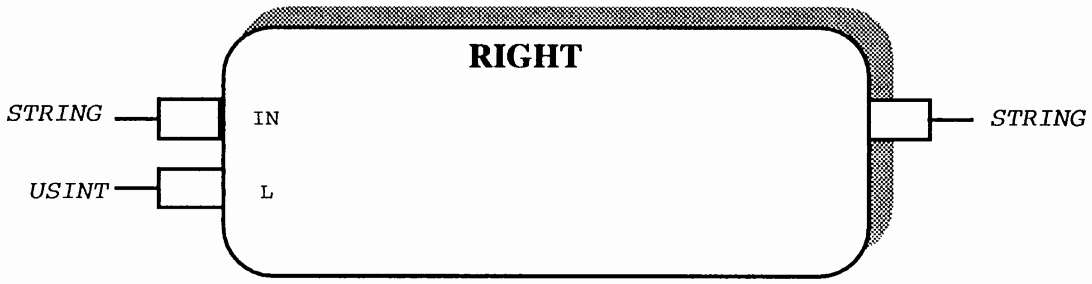
Example ST:

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

Sets product.Val to 'A34F'.

RIGHT

Extract a given number of characters L, from the right end of a character string (STRING) IN, i.e. up to the end of the string. However, if parameter IN is less than L characters long, a null zero length string is returned i.e. ''.



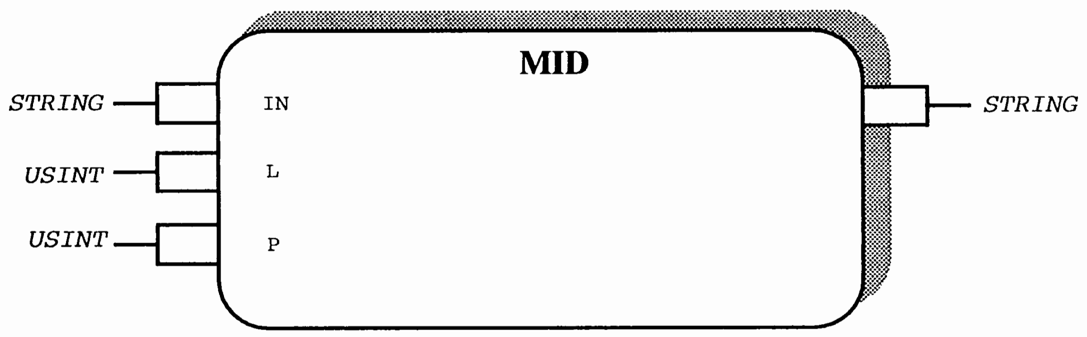
Example ST:

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

Sets recipe.Val to 'Blue'

MID

Extracts a character string (STRING) of a given length L, from a given position P, within a string IN. However, if parameter IN is less than (L + P) -1 characters long or if P is 0, the function returns a zero length null string i.e. ''.



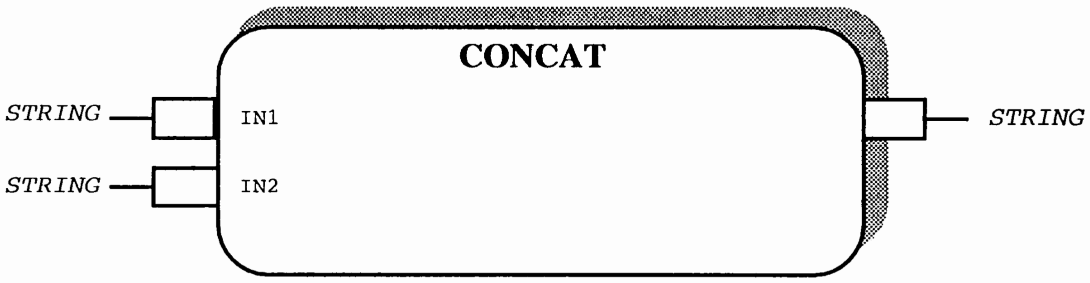
Example ST:

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

Sets log.Val to '12_56'

CONCAT

Joins two character strings (STRING) IN1 and IN2 together, with IN1 string being placed at the beginning of the returned string, i.e. concatenates two strings.



Example ST:

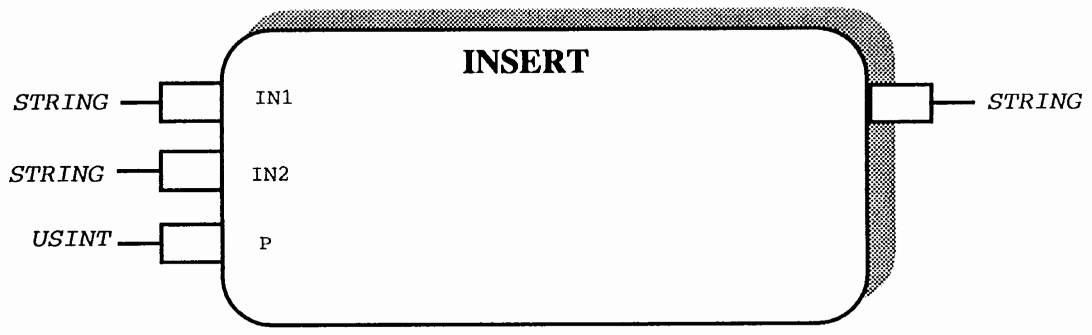
```
report.Val:= CONCAT(IN1:= 'Alarm Status ',
                    IN2:= alarm.Val);
```

If, for example parameter alarm.Val is 'OK', the string report.Val is set to 'Alarm Status OK'

INSERT

Inserts one character string (STRING) IN2 into another string IN1 starting at a given position P in string IN1. However, if parameter IN1 is less than P characters long, the function returns a zero length null string i.e. ''. The previous contents of string IN1 are overwritten by the inserted string IN2.

If P equals zero, string IN2 is inserted in front of string IN1.



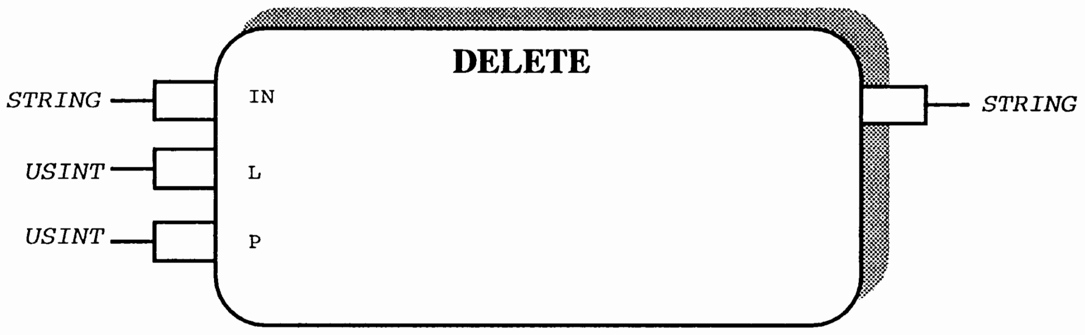
Example ST:

```
log.Val:= INSERT(IN1:= 'log:    :',
                    IN2:= '03', P:= 5);
```

Sets string log.Val to 'log:03:'

DELETE

Deletes a part of a character string (STRING) L characters long from a string IN starting at position P. However if parameter IN is less than (L+P)-1 characters long or P is 0, the function returns a null zero length string i.e. ''.



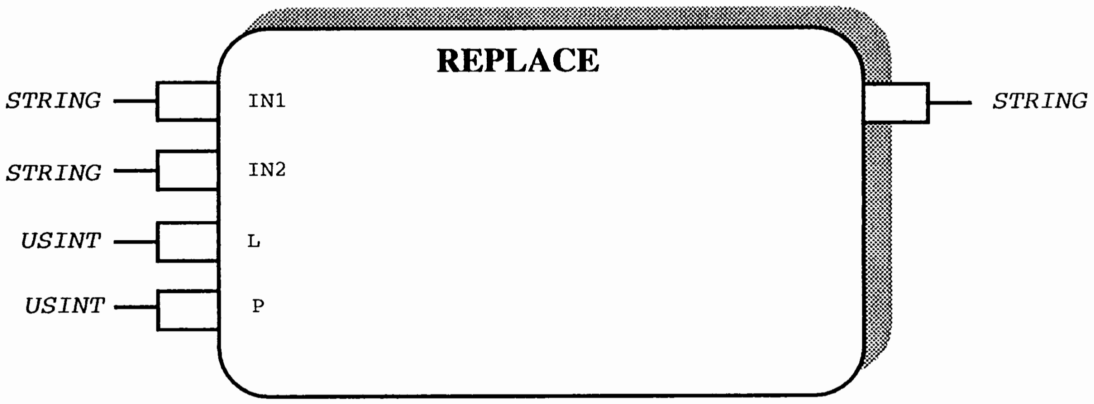
Example ST:

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

Sets new.Val to 'Batch Ready'

REPLACE

Replaces L characters of a character string (STRING) IN1 with another string IN2 beginning at position P in string IN1. However if parameter IN1 is less than (L+P)-1 characters long or P is 0, the function returns a null zero length string i.e. ''.



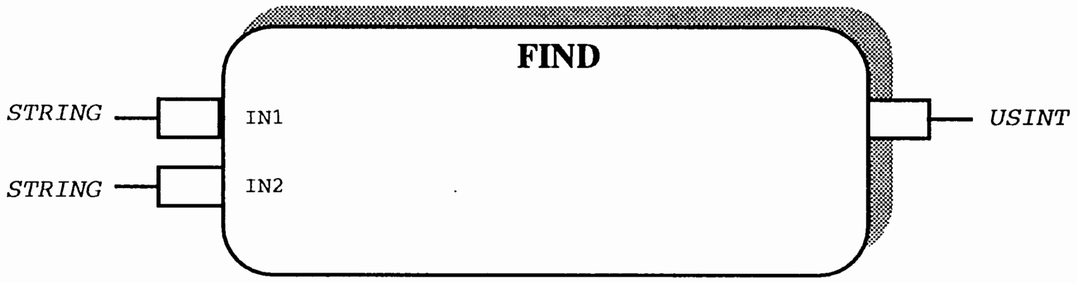
Example ST:

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

Sets response.Val to 'Valve 5C Open'

FIND

Returns as an unsigned integer (USINT), the character position of the beginning of the first occurrence of a string IN2 in another string IN1. If no occurrence is found then returns 0. If string IN2 is null i.e. '', returns 1.



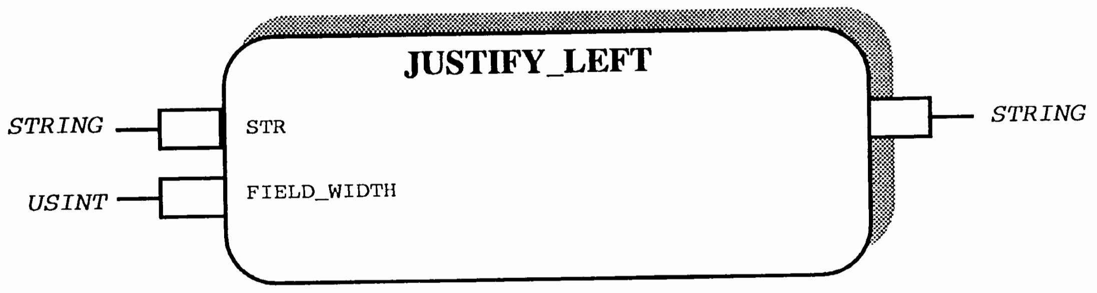
Example ST:

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

Sets A.Val to 2.

JUSTIFY_LEFT

This function left justifies, i.e. positions a character string (STRING) within a specified field at the left most position. The returned string is extended with spaces to fill the field width. However, if the length of string STR is greater than FIELD_WIDTH, the function returns a null zero length string i.e. ''.



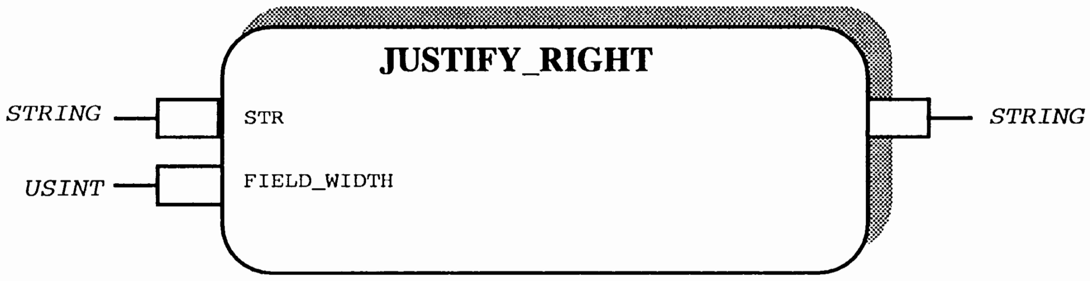
Example ST:

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

Sets report.Val to 'Report: '

JUSTIFY_RIGHT

This function right justifies, i.e. positions a character string (STRING) within a specified field at the right most position. The returned string is prefixed with spaces to fill the field width. If the length of STR is greater than FIELD_WIDTH, the function returns a null string i.e. ''.



Example ST:

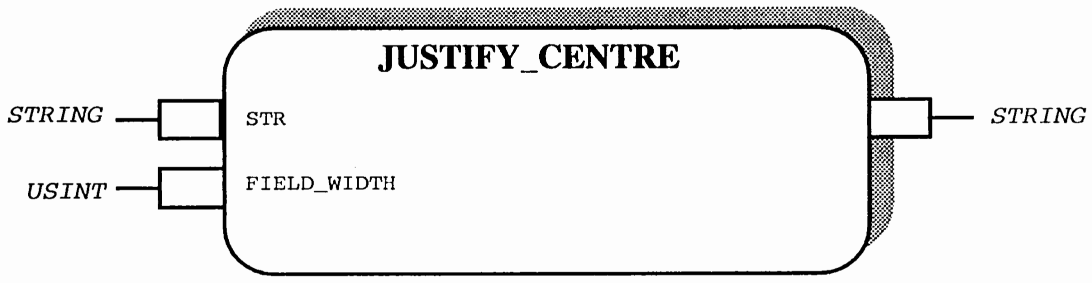
```
report.Val:=  JUSTIFY_RIGHT(STR:=  'Report:',
                                   FIELD_WIDTH:=  12);
```

Sets report.Val to '   Report:'

JUSTIFY_CENTRE

This function centrally justifies, i.e. positions a character string (STRING) within a specified field at the central position with equal spaces added to the left and right ends of the string. The returned string is prefixed and postfixed with spaces to fill the field width. However if the length of string STR is greater than FIELD_WIDTH, the function returns a null zero length string i.e. ''.

If the string cannot be centred exactly it will be biased to the left.



Example ST:

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

Sets banner.Val to '   ZONE1   '

Chapter 5

TYPE CONVERSION FUNCTIONS

Edition 2

Contents

Overview

With these functions, it is possible to convert one data type to another. A wide range of conversion functions is provided, e.g. TIME_TO_REAL can be used to convert a duration (TIME) into a floating point (REAL) value as a number of seconds.

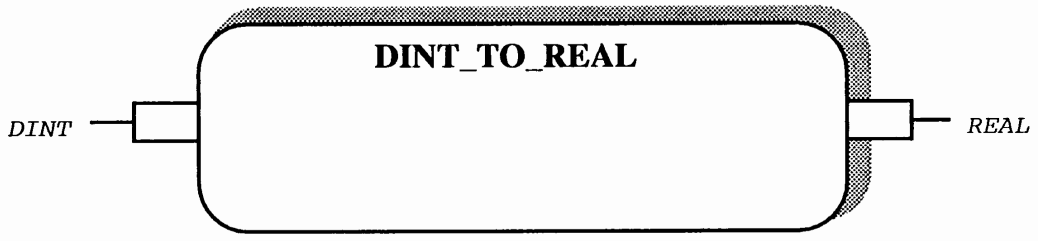
With most Structured Text functions and operations, it is important that parameters with the correct data type are used. The PC3000 Programming Station will normally produce an error message when an ST statement is created that involves an inappropriate data type. However, currently there are no checks regarding the different integer data types such as DINT and USINT, see caution notes on page 1-2.

In some cases, it will be illogical to attempt to use the wrong data type. For example: `a.Val := b.Val * c.Val` would be meaningless if a.Val, b.Val and c.Val were all character string (STRING) data types. However, there are many cases where it is necessary to take a parameter of one data type and use it in an expression with other data types. For example, it is frequently necessary to perform calculations with both floating point (REAL) and integer parameters.

The following conversion functions can be used where there is a meaningful conversion of one data type to another. Appendix D provides additional guidelines.

DINT_TO_REAL

Converts a signed integer (DINT) into a floating point (REAL).



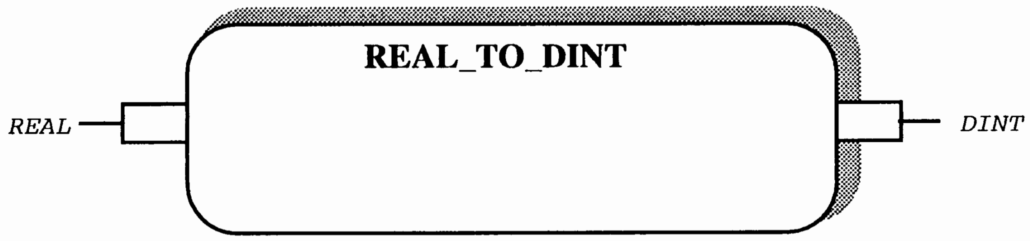
Example ST

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

Sets a.Val to 12.000

REAL_TO_DINT

Converts a floating point (REAL) into a signed integer (DINT). Rounding takes place to the nearest integer.



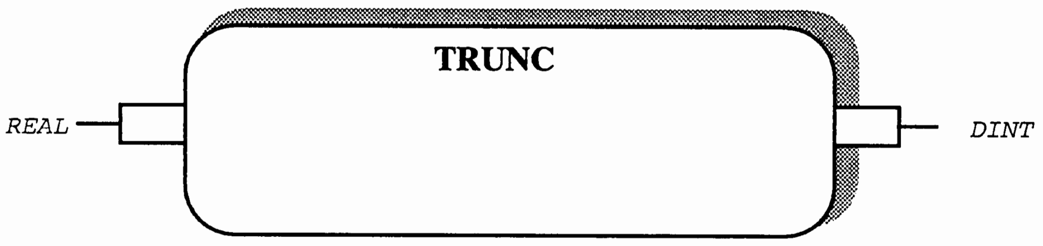
Example ST

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

Sets a.Val to 13

TRUNC

Converts the integer part of a floating point (REAL) into a signed integer (DINT) by truncating the decimal part of the value.



Example ST:

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

Sets a.Val to 12

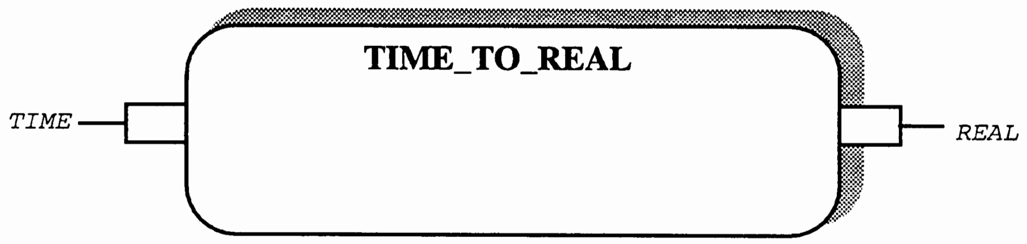
Example ST:

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

Sets a.Val to -6

TIME_TO_REAL

Converts a duration (TIME) into a floating point (REAL) number of seconds.



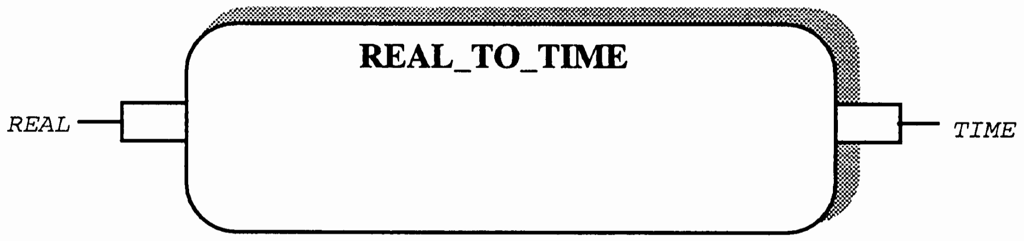
Example ST :

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

Sets a.Val to 10.5

REAL_TO_TIME

Converts a floating point (REAL) number of seconds into a duration (TIME). The input IN should be a positive number less than $(2^{32})/1000$ i.e. less than 4294967·295 to convert to a valid duration.



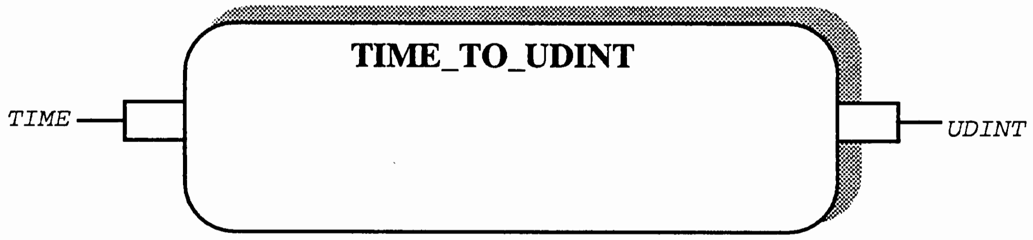
Example ST:

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

Sets a.Val to T#10s_500ms

TIME_TO_UDINT

Converts a duration (TIME) into an integer (UDINT) number of milliseconds.



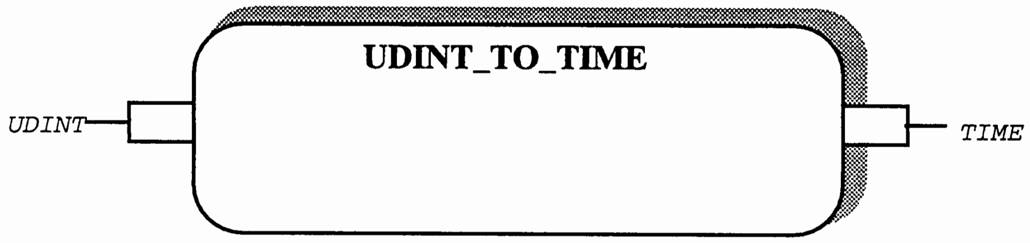
Example ST:

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

Sets a.Val to 10500

UDINT_TO_TIME

Converts an integer (UDINT) number of milliseconds into a duration (TIME). The input IN, should be a positive number to produce a valid duration.



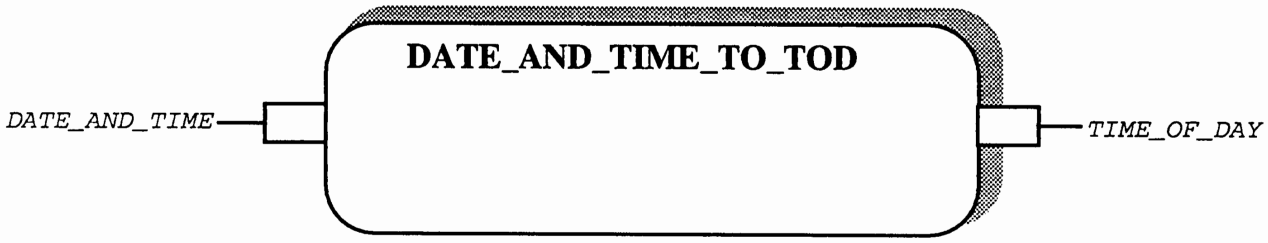
Example ST:

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

Sets a.Val to T#10s_500ms

DATE_AND_TIME_TO_TOD

Converts a date and time (DATE_AND_TIME) into a time of day (TIME_OF_DAY) by removing the date (DATE) component.



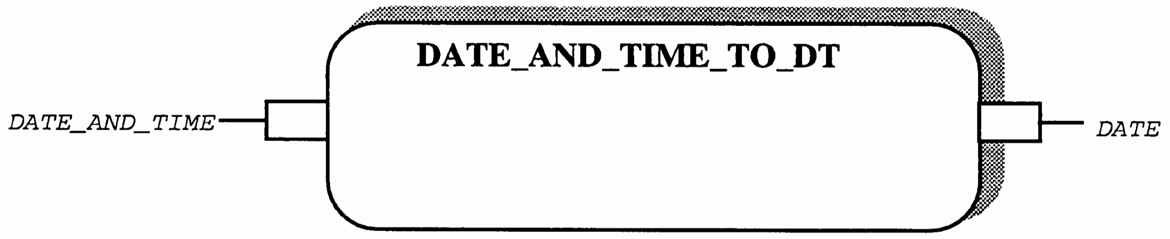
Example ST:

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

Sets a.Val to TOD#14:30:00

DATE_AND_TIME_TO_DT

Converts a date and time (DATE_AND_TIME) into a date (DATE) by removing the time of day (TIME_OF_DAY) component.



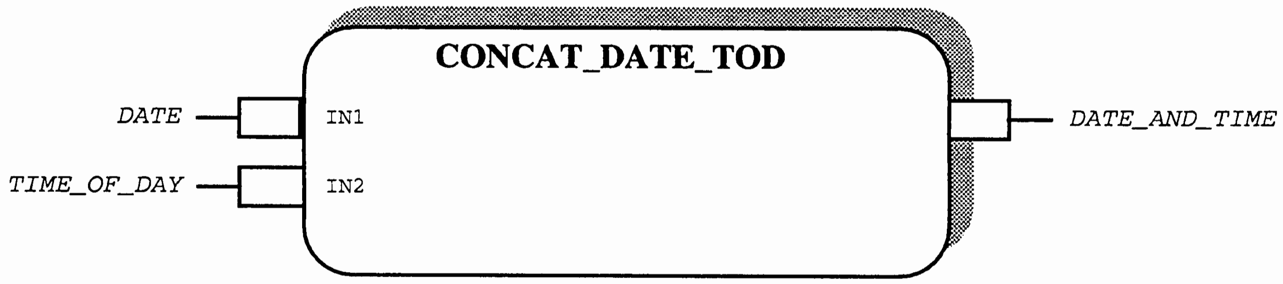
Example ST:

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

Sets a.Val to D#02-Sep-1991

CONCAT_DATE_TOD

Combine a date (DATE) and a time of day (TIME_OF_DAY) to produce a date and time (DATE_AND_TIME).



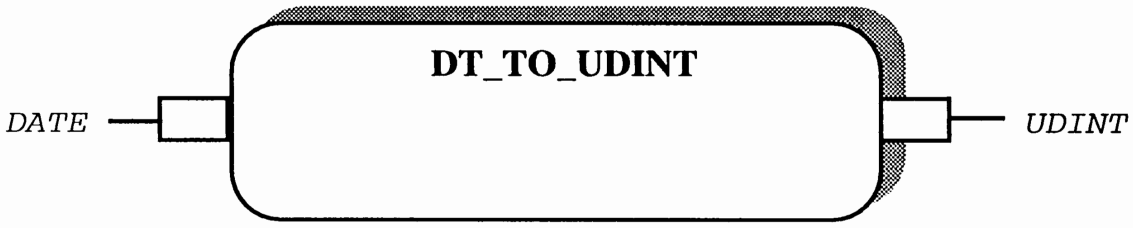
Example ST:

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

Sets a.Val to DT#02-Sep-1991-14:30:00

DT_TO_UDINT (version 2.27 and later)

Converts a date (DATE) into an unsigned integer (UDINT)

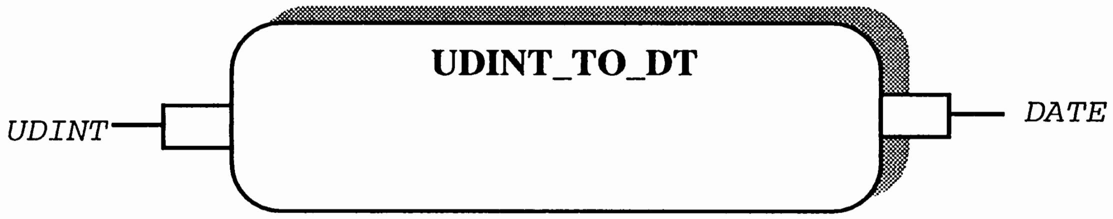


Example ST:

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

UDINT_TO_DT (version 2.27 and later)

Converts an unsigned integer (UDINT) into a date (DATE)



Example ST:

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

Chapter 6

STRING CONVERSION FUNCTIONS

Edition 2

Contents

Contents (continued)

Overview

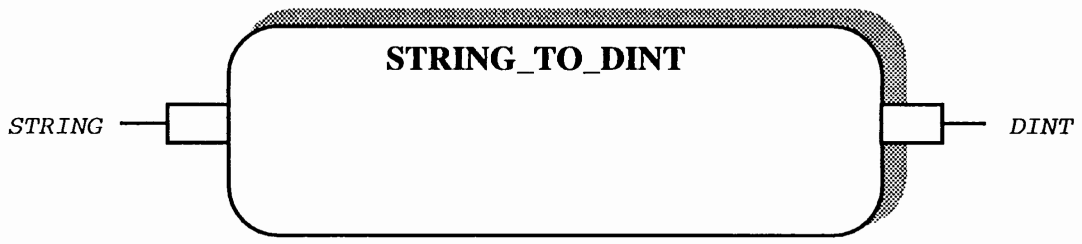
These functions are provided to convert data held in strings into a particular data type or to convert a given data type value into a string. Typically these are used with applications involving communications, such as where a value has been received in a string as ASCII text and requires conversion to an IEC DIS 1131 data type or where it is necessary to transmit a value of a parameter as ASCII text to a terminal. For example : STRING_TO_REAL can be used to convert a character string (STRING) containing '1.23' into a floating point (REAL) value, TIME_TO_STRING can be used to convert a duration into a string such as '18s100ms'.

With string conversion functions, a null zero length string i.e. '' is returned if the input string does not contain valid characters at the start of the string.

The type of characters that define valid strings will vary for the different string conversion functions and are specified in the function descriptions. String conversion functions all ignore leading spaces and terminate on the first character which is not valid for the particular input type.

STRING_TO_DINT

Converts a character string (STRING) into an integer (DINT) containing digits 0-9. Leading sign characters "+" or "-" will be interpreted as the sign for the output.



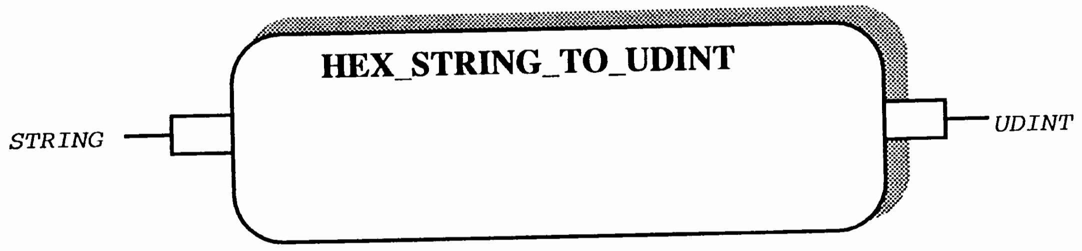
Example ST:

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

Sets a.Val to 234.

HEX_STRING_TO_UDINT

Converts a character string (STRING) containing ASCII encoding of hexadecimal using characters 0-9,A, B, C, D, E, F into an unsigned integer (UDINT). The input should be not greater than '7FFFFFFF'. Leading spaces are ignored, conversion terminates with first non hexadecimal digit.



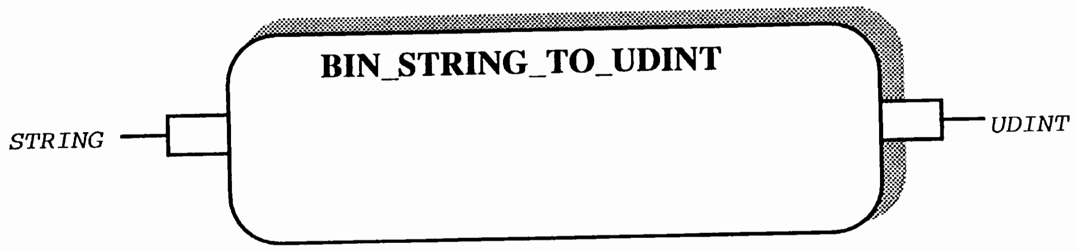
Example ST:

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

Sets a.Val to 42

BIN_STRING_TO_UDINT

Converts a character string (STRING) containing the digits "1" and "0" into a integer (UDINT), the input can be up to 31 '1's and '0's. Leading spaces are ignored, conversion terminates with first character that is not '1' or '0'.



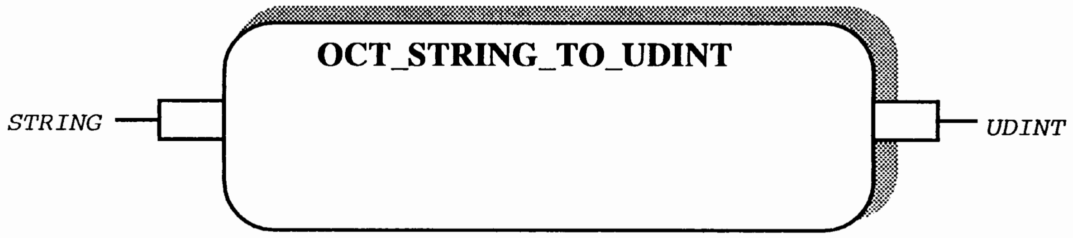
Example ST:

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

Sets a.Val to 165

OCT_STRING_TO_UDINT

Converts a character string (STRING) containing the digits '0', '1' through '7' that represents an octal value, into an integer (UDINT). The input should not represent an octal value greater than '17777777777'. Leading spaces in the input string are ignored. Conversion terminates with the first character that is not in the range '0' to '7'.



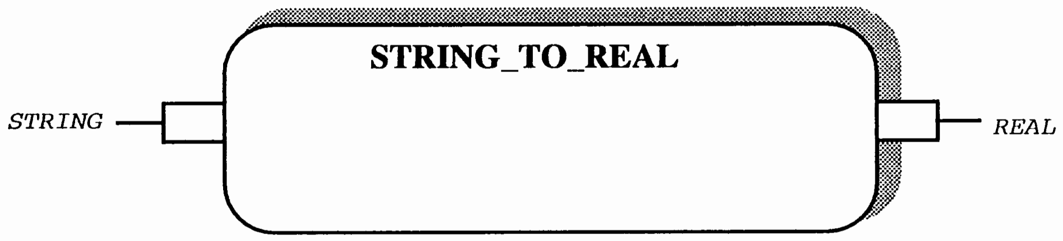
Example ST

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

Sets a.Val to 56

STRING_TO_REAL

Converts a character string (STRING) into a floating point (REAL). The input string can have leading spaces, a sign + or -, spaces between the sign and the first digit. A leading zero before the decimal point is optional. After the optional sign input should only contain digits '0' to '9' and a single decimal point. The conversion terminates on the first non-valid character. Exponential format is not supported.



Example ST :

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

Sets a.Val to -0.23

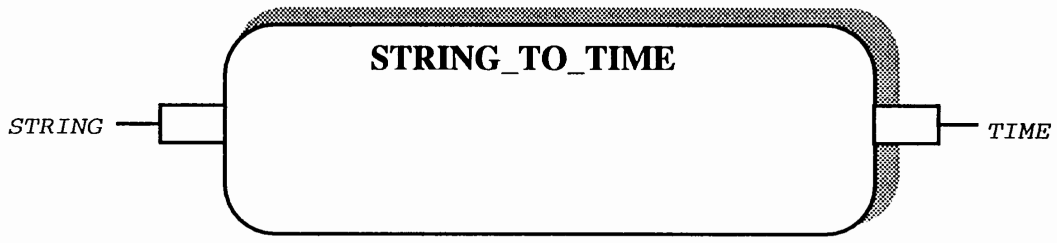
Example ST:

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

Sets a.Val to 123.82

STRING_TO_TIME

Converts a character string (STRING) into a duration (TIME). The format of the string should be in the IEC format , e.g. 5d14h12m18s100ms. Null fields can be omitted e.g. 5d18s. The least significant field present can have a decimal component.



Example ST:

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

Sets a.Val to T#5d_14h

Example ST:

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

Sets a.Val to T#10h_30m

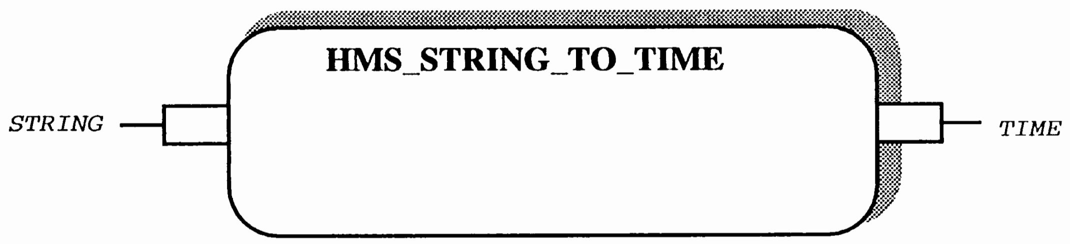
Example ST:

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

Sets a Val to T#12m0s500ms

HMS_STRING_TO_TIME

Converts a character string (STRING) into a duration (TIME). The format of the string should be hours, minutes, seconds i.e. HH:MM:SS .



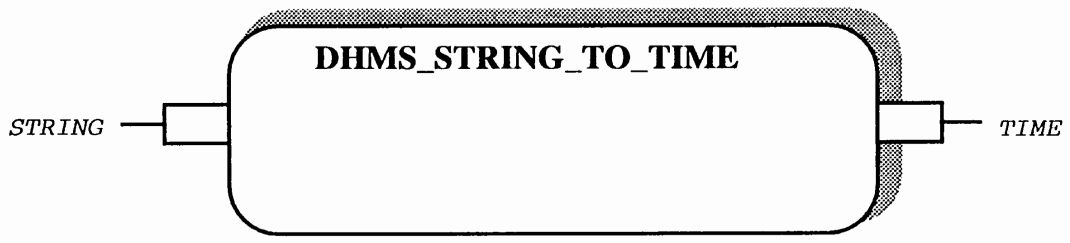
Example ST :

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

Sets a.Val to T#10h_20m_10s

DHMS_STRING_TO_TIME

Converts a character string (STRING) into a duration (TIME). The format of the string should be day, hour, minute, second i.e. DD:HH:MM:SS .



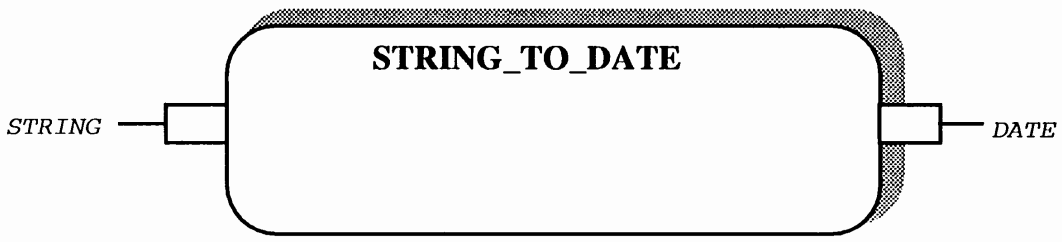
Example ST:

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

Sets a.Val to T#20d_10h_20m_10s

STRING_TO_DATE

Converts a character string (STRING) into a date (DATE). The format of the STRING should be in the IEC format, i.e. year, month, day i.e. YYYY-MM-DD.



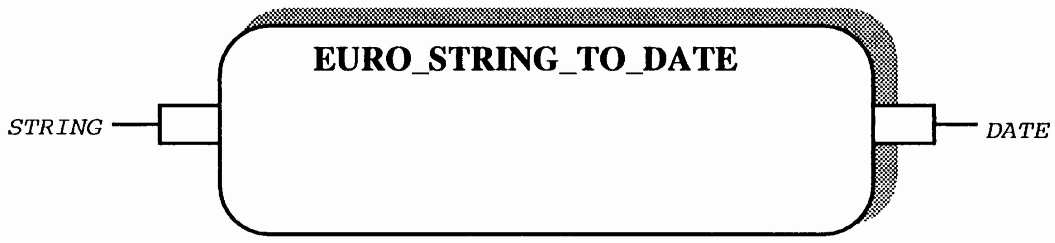
Example ST:

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

Sets a.Val to D#02-Sep-1991

EURO_STRING_TO_DATE

Converts a European format date character string (STRING) into a date (DATE). The format of the string should be day, month, year i.e. DD-MM-YY



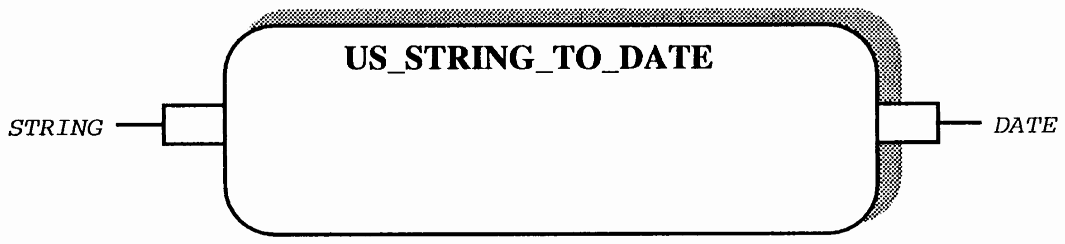
Example ST:

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

Sets a.Val to D#02-Sep-1991

US_STRING_TO_DATE

Converts a American (US) format STRING into a DATE. The format of the string should be in month, day, year i.e. MM-DD-YY.



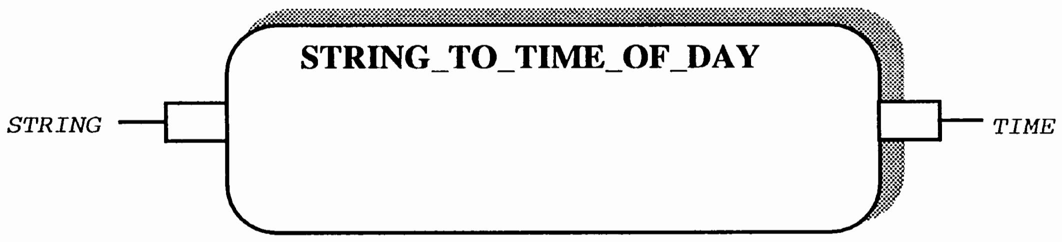
Example ST:

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

Sets a.Val to D#02-Sep-1991

STRING_TO_TIME_OF_DAY

Converts a character string (STRING) into a time of day (TIME_OF_DAY). The format of the string should be in the IEC format, hour, minute, second i.e. HH:MM:SS .



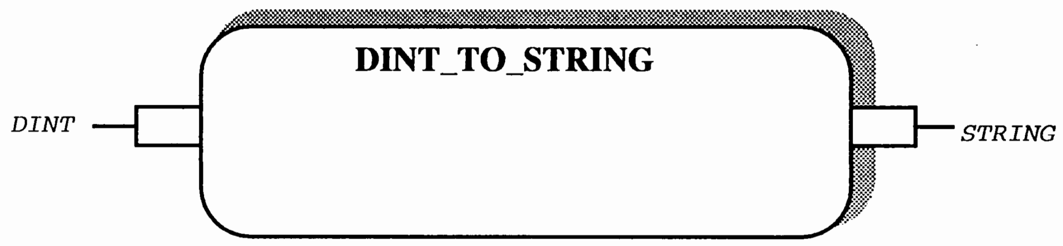
Example ST:

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

Sets a.Val to TOD#12:30:10

DINT_TO_STRING

Converts double integer (DINT) into a signed decimal format character string (STRING).



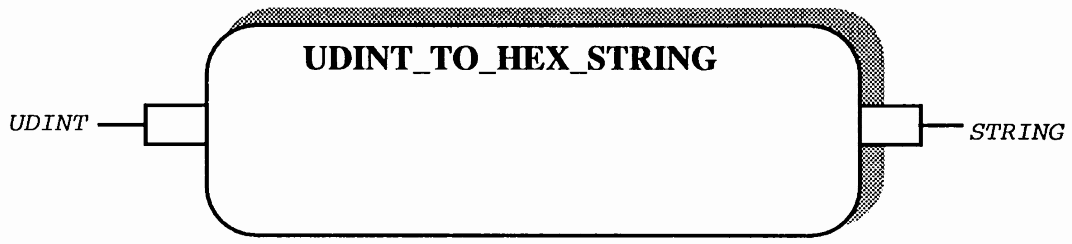
Example ST:

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

Sets a.Val to '-33'

UDINT_TO_HEX_STRING

Converts unsigned double integer (UDINT) into a hexadecimal format character string (STRING). The input value should not be negative.



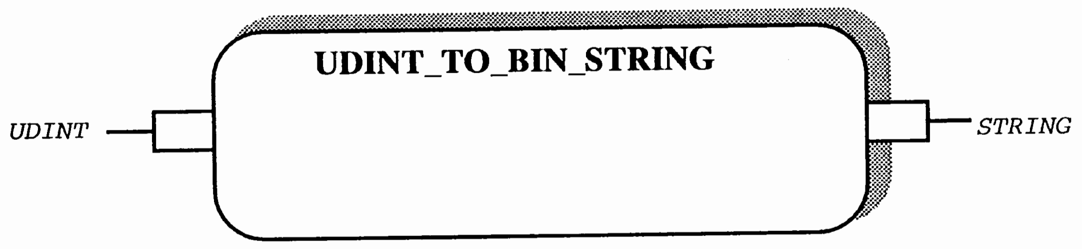
Example ST:

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

Sets a.Val to '2A'

UDINT_TO_BIN_STRING

Converts unsigned double integer (UDINT) into a binary format character string (STRING). Only as many digits are created as are needed to represent the value; there are no extra leading zeroes added. The input value should not be negative.



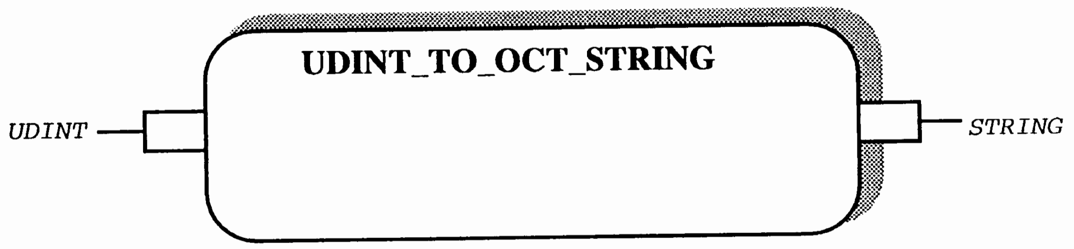
Example ST:

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

Sets a.Val to '101010'

UDINT_TO_OCT_STRING

Converts unsigned double integer (UDINT) into an octal character string (STRING). Only as many digits are created as are needed to represent the value; there are no extra leading zeroes added. The input value should not be negative.



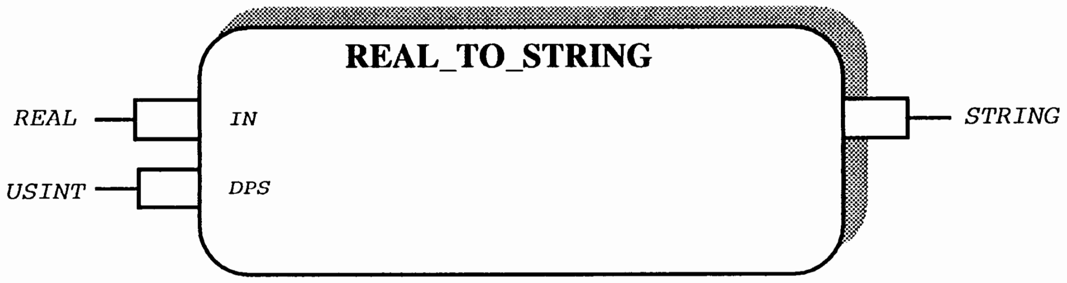
Example ST :

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

Sets a.Val to '52'

REAL_TO_STRING

Converts a floating point (REAL) into a character string is equal to (STRING). The DPS parameter defines the number of decimal places. If DPS is equal to 0, there are no decimal places but the last character is a decimal point. If DPS is equal to -1, there are no decimal places and no decimal point.



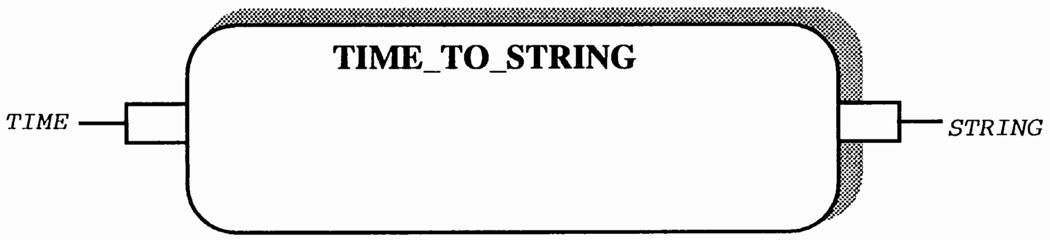
Example ST :

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

Sets a.Val to '42.34'

TIME_TO_STRING

Converts a duration (TIME) into a character string (STRING). The format of the string will be in the IEC format , e.g. 5d14h12m18s100ms.



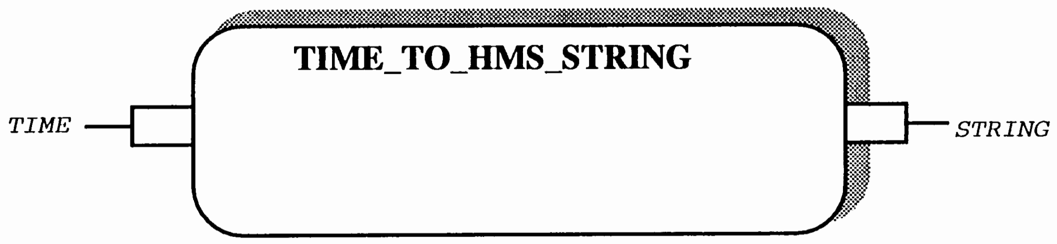
Example ST:

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

Sets a.Val to '12h30m20s'

TIME_TO_HMS_STRING

Converts a duration (TIME) into a character string (STRING). The format of the string will be hour, minute, second i.e. HH:MM:SS .



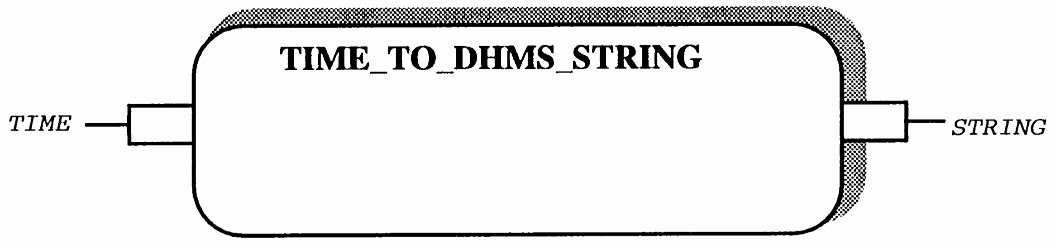
Example ST :

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

Sets a.Val to '12:30:20'

TIME_TO_DHMS_STRING

Converts a duration (TIME) into a character string (STRING). The format of the string will be day, hour, minute, second i.e. DD:HH:MM:SS .



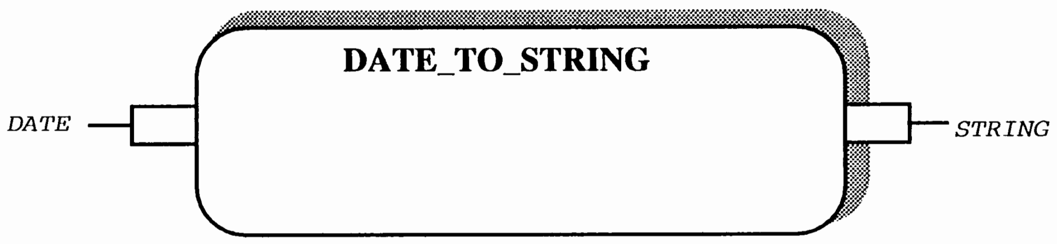
Example ST :

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

Sets a.Val to '02:12:30:20'

DATE_TO_STRING

Converts a date (DATE) into a character string (STRING). The format of the string will be the IEC format year, month, day i.e. YYYY-MM-DD.



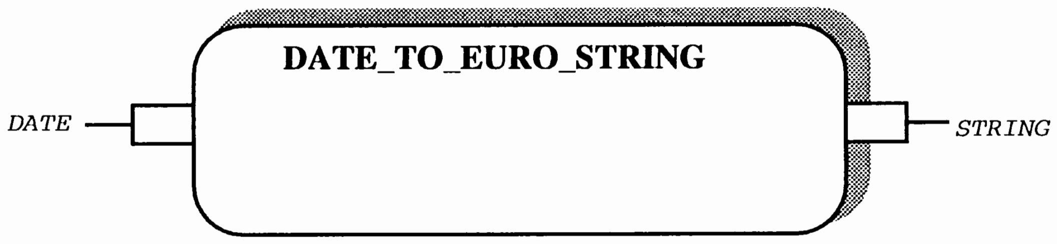
Example ST:

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

Sets a.Val to '1991-09-02'

DATE_TO_EURO_STRING

Converts a date (DATE) into a European format character string (STRING). The format of the string will be day, month, year i.e. DD-MM-YY.



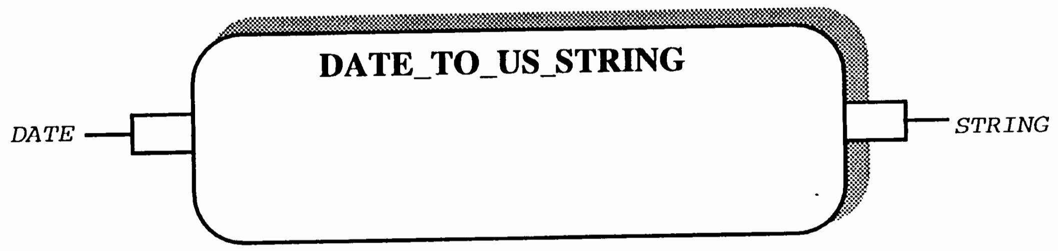
Example ST :

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

Sets a.Val to '02-09-91'

DATE_TO_US_STRING

Converts a date (DATE) into a American (US) format character string (STRING). The format of the string will be month, day, year i.e. MM-DD-YY.



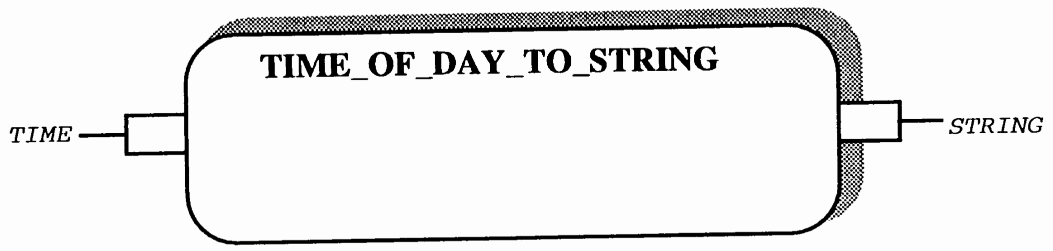
Example ST :

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

Sets a.Val to '09-02-91'

TIME_OF_DAY_TO_STRING

Converts a time of day (TIME_OF_DAY) into a character string (STRING). The format of the string will be in the IEC format hour, minute, second i.e. HH:MM:SS.



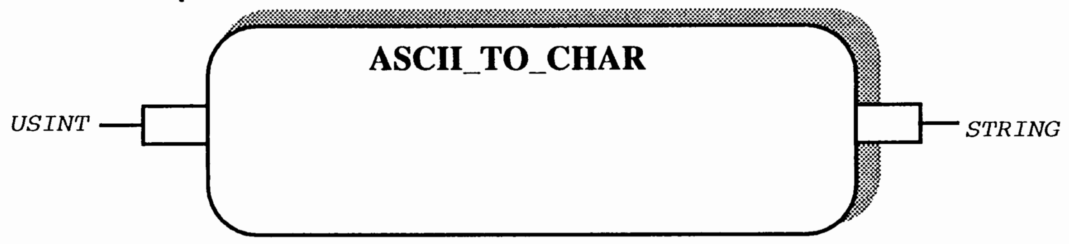
Example ST :

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

Sets a.Val to '12:30:10'

ASCII_TO_CHAR

Converts an integer (USINT) in the range 0 to 255 into a single ASCII character string. If the integer is outside this range, the returned value is 0.



Example ST :

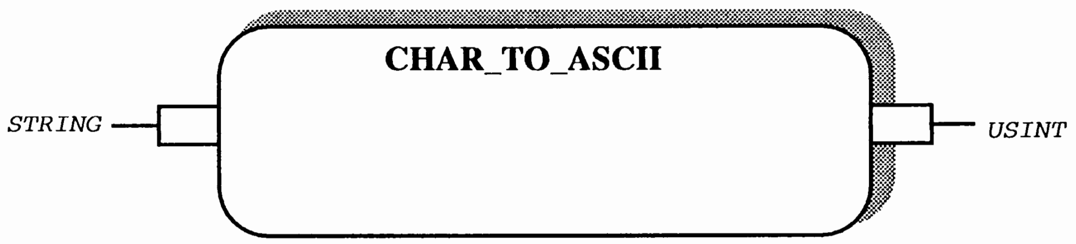
```
a.Val:= ASCII_TO_CHAR(IN:= 66);
```

Sets a.Val to 'B'

CHAR_TO_ASCII

Converts an ASCII character (STRING) into an integer (USINT) in the range 0 to 255.

An empty string will result in the value 0.



Example ST :

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

Sets a.Val to 66

Chapter 7

TIME ARITHEMTIC FUNCTIONS

Edition 2

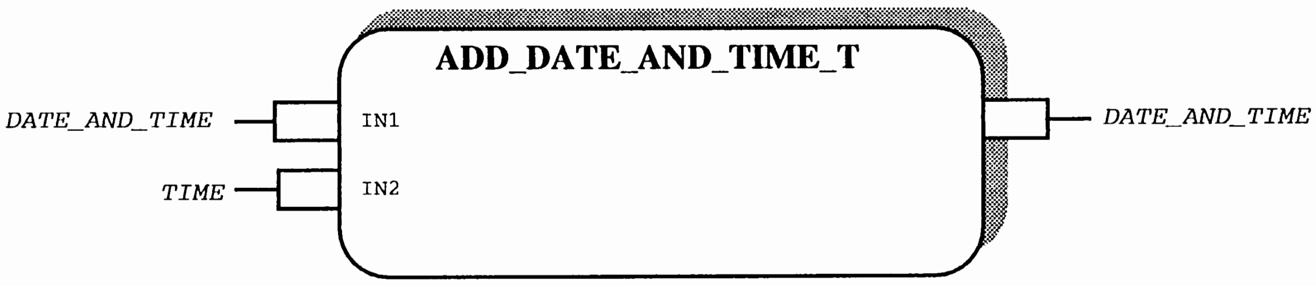
Contents

Overview

Handling time and date calculations is facilitated using a range of time functions. Functions provided include most valid addition and subtraction operations between the different time and date data types. For example, ADD_DATE_AND_TIME_T allows a duration expressed as a duration (TIME) to be added to a date and time (DATE_AND_TIME) to create a future date and time (DATE_AND_TIME).

ADD_DATE_AND_TIME_T

Adds a duration (TIME) (IN2) to a date and time (DATE_AND_TIME) (IN1) to give a date and time (DATE_AND_TIME).



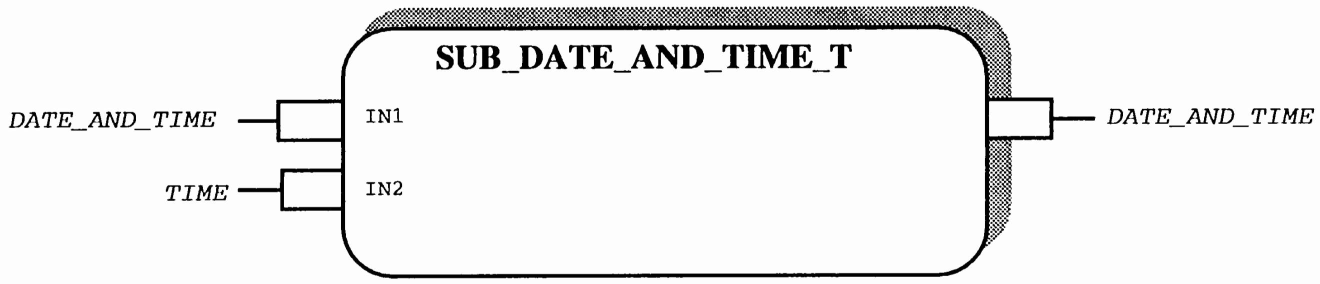
Example ST :

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

Sets a.Val to DT#03-Sep-1991-13:31:11

SUB_DATE_AND_TIME_T

Subtracts a duration (TIME) (IN2) from a date and time (DATE_AND_TIME) (IN1) to give to give a date and time (DATE_AND_TIME).



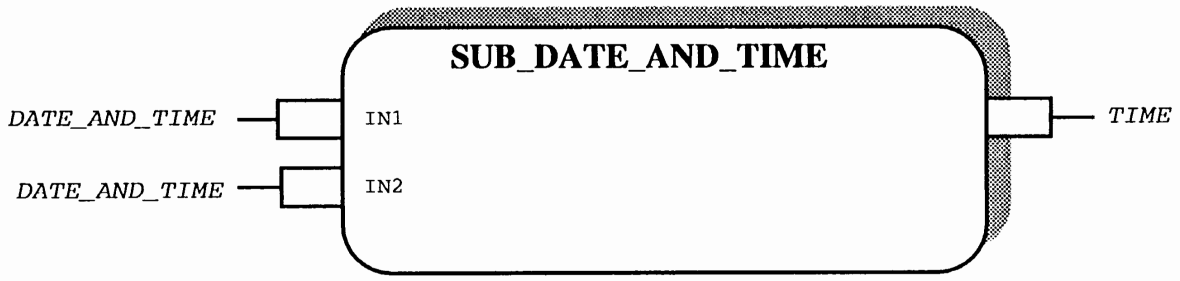
Example ST :

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

Sets a.Val to DT#01-Sep-1991-11:29:09

SUB_DATE_AND_TIME

Subtracts a date and time (DATE_AND_TIME) (IN2) from another date and time (DATE_AND_TIME) (IN1) to give a duration (TIME).



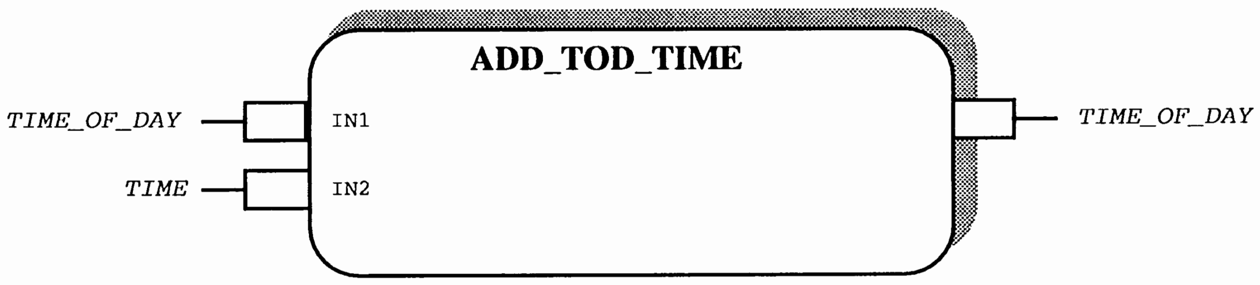
Example ST :

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

Sets a.Val to T#01d_01h_01m_01s

ADD_TOD_TIME

Adds a duration (TIME) (IN2) to a time of day (TIME_OF_DAY) (IN1) to give a time of day (TIME_OF_DAY).



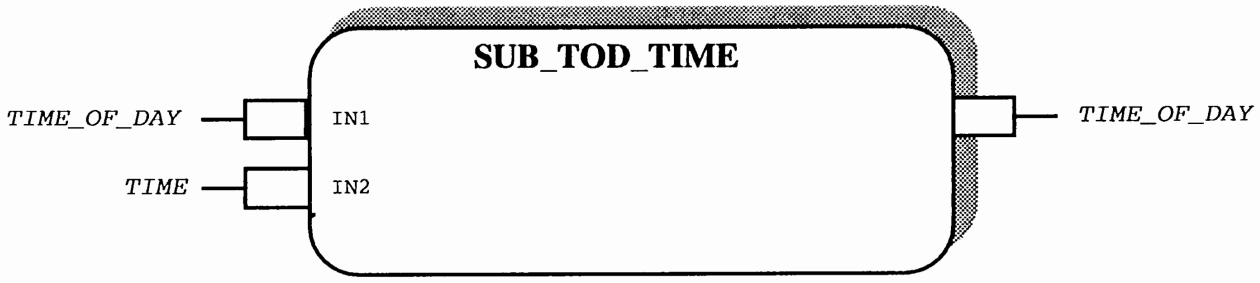
Example ST :

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

Sets a.Val to TOD#12:50:20

SUB_TOD_TIME

Subtracts a duration (TIME) (IN2) from a time of day (TIME_OF_DAY) (IN1) to give a time of day (TIME_OF_DAY).



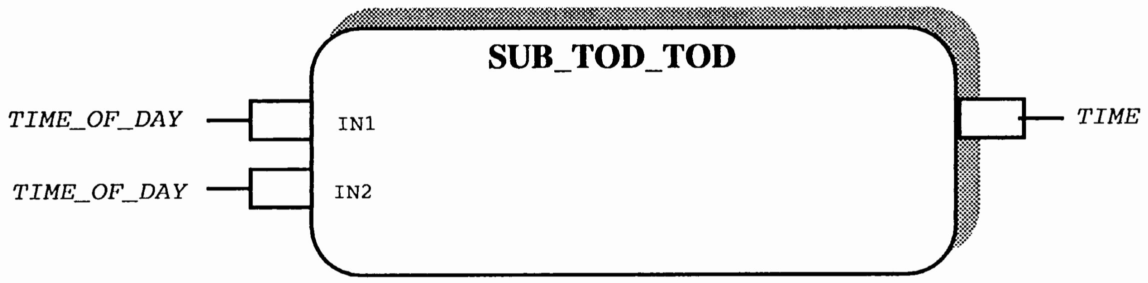
Example ST :

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

Sets a.Val to TOD#12:10:00

SUB_TOD_TOD

Subtracts a time of day (TIME_OF_DAY)(IN2) from another time of day (TIME_OF_DAY)(IN1) to give a duration (TIME). Note that if IN2 is later in the day than IN1, the TIME will be returned as T#0s.



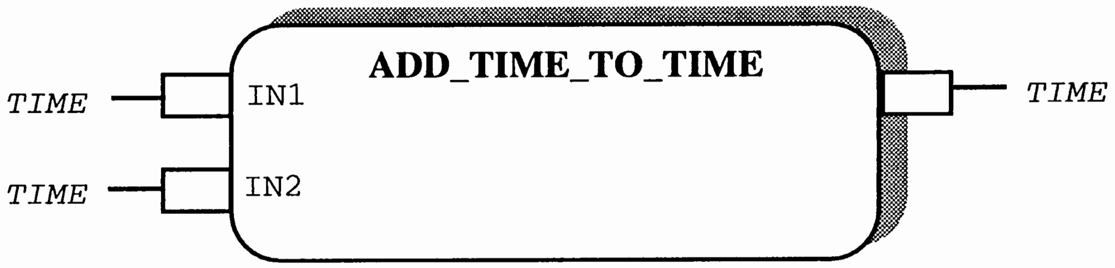
Example ST :

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

Sets a.Val to T#2h_10m_2s

ADD_TIME_TO_TIME (version 2.27 and later)

Adds a duration (TIME) to a second duration (TIME). The result is also a duration (TIME).



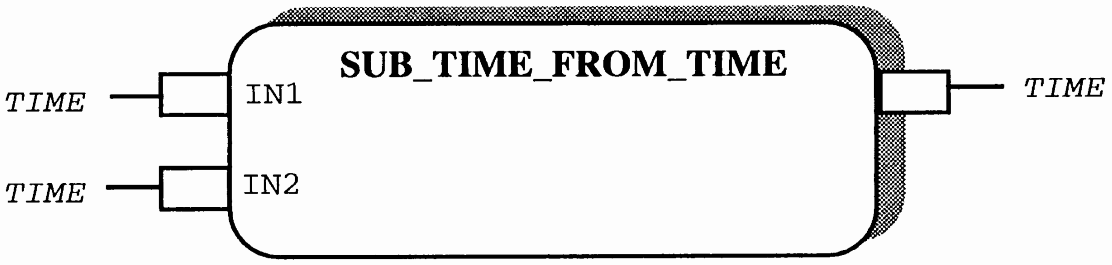
Example ST:

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

sets time.Val to T#4h_31m_13s

SUB_TIME_FROM_TIME (version 2.27 and later)

Subtracts a duration (TIME) (IN2) from a second duration (TIME)(IN1).
The result is also duration (TIME). If the IN2 is greater than IN1 then the function will return a value of T#0mS.



Example ST:

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

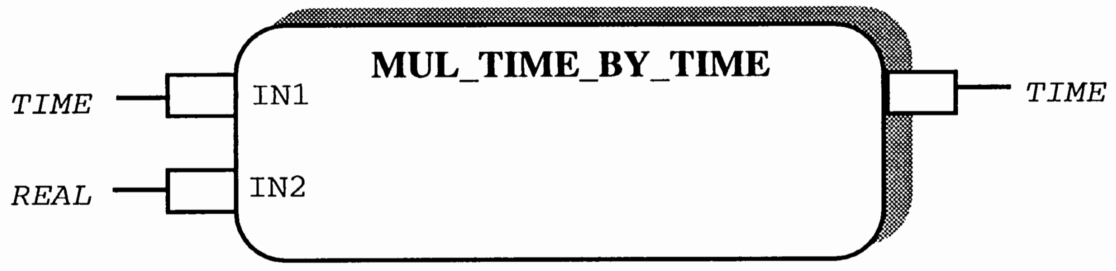
sets time.Val to T#3d_1h_27m_8s

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

sets time.Val to T#0mS

MUL_TIME_BY_REAL (version 2.27 and later)

Multiplies a duration (TIME) by a numeric value (REAL). The result is a duration (TIME).



Example ST:

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

sets time.Val to T#2d_7h_12m_24s

Chapter 8

COMPACT FUNCTIONS

Edition 1

Contents

Contents (continued)

Overview

A range of compact functions is provided so that an array of values of a particular data type can be packed and unpacked from a string. These functions may be used in a number of applications including compacting a set of values into a string for transmission via communications or holding a set of values as part of a simple recipe system. For example, REP_SINT_IN_STR replaces a short integer in a string at a prescribed position.

The compact functions are provided to allow character strings such as Long_String User Variables, and Remote_Str Remote Variables, to be treated as arrays of values of specified data types. There is a pair of functions to support each basic data type. One to replace a value in the string (REP_***_IN_STR) and one to extract a value from a string (EXT_***_FROM_STR) where *** is the data type name.

The replace function is also used to insert an initial value in the string. In general the strings should only be accessed by these functions. Only one sort of data type should normally be stored in a particular string. The INDEX parameter is used to select the element of the 'array' to be read or written. The first element has an INDEX of zero. The elements are stored from left to right in the string and packed in adjacent bytes; there are no empty bytes between values.

If an attempt is made to read an element which is past the end of the string, the default value for that data type (normally 0) is returned. If an attempt is made to write an element which is past the end of the string, the string will be extended so that it includes that element. When extending strings, new elements are set to the default value for that data type. For example, when a User Variable Long_String is first created, it will default to a null zero length string i.e. ''. If a value is the added to the string using INDEX 3, the string will be extended to contain 4 elements, (0,1,2 and 3); the values for elements 0,1 and 2 will be initialised to take null values i.e. 0 for numeric elements.

Example ST:

```
str1.Val:=  '';  (* Initialise 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 );
```

Sets int0.Val to 0, int1.Val to 0, and int3.Val to 3000.

Note: Strings manipulated using the COMPACT functions should not be further manipulated by use of the STRING functions. Internal values may be lost due to misalignment within the string storage area.

If the destination string for a REP_***_IN_STR cannot be extended to contain the result because it is past the maximum length for the string, the whole string will be set to a null string i.e. ''.

These functions are normally used with strings which have a maximum length of 255 bytes such as User Variable Long_String but can be used with shorter strings. In which case the number of values that can be packed will be reduced and care should be taken not to extract or replace values beyond the end of the string.

Refer to the REP_BOOL_IN_STR and EXT_BOOL_FROM_STR functions for ST examples. All the functions will follow the same model although they extract or replace different data types and the number of values that can be packed into a string will vary.

Byte reversal

If compact strings are to be transmitted to or received from other equipment for example via serial communications, the byte reversal variant of these functions may be required. These take the form

REP_***_IN_STR_X and EXT_***_FROM_STR_X.

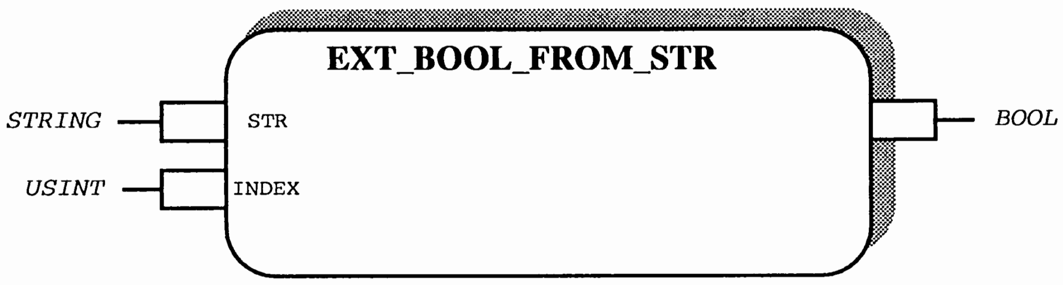
Byte reversal may be required for data types when the compact strings are transferred between systems that use a different memory layout to the PC3000.

For example, if the PC3000 receives strings containing blocks of 32 bit integers (double integer DINT) which have been created on certain PLCs, the EXT_DINT_FROM_STR_X function may be needed.

The REP_DINT_IN_STR_X may be needed to create a string containing a block of integers to be transmitted to some PLCs.

EXT_BOOL_FROM_STR

Extracts a boolean (BOOL) value from a string. Eight booleans are stored in each character, so a maximum of 2040 may be stored in a 255 character string (STRING). INDEX lies in the range 0 to 2039.



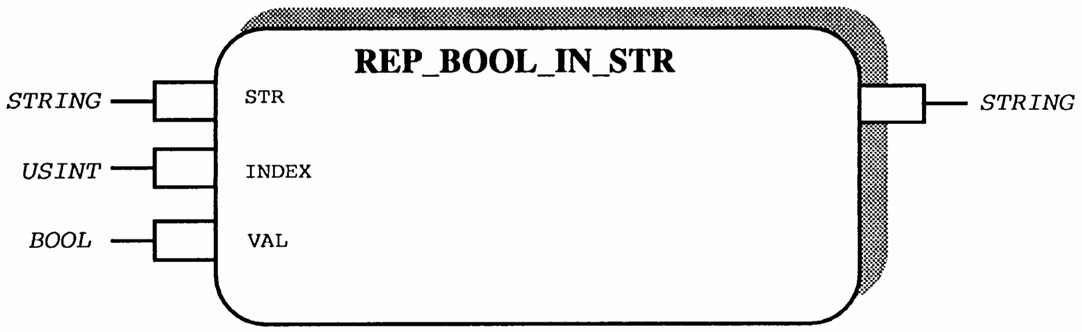
Example ST :

```
bool.Val:=  EXT_BOOL_FROM_STR(STR  :=  str1.Val,
                                      INDEX:=  300);
```

bool.Val is set to the state of the 301th bit packed into string str1.Val .

REP_BOOL_IN_STR

Replaces a Boolean value in a string. Eight booleans are stored in each character, so a maximum of 2040 booleans (BOOLs) may be stored in a 255 character string (STRING). INDEX lies in the range 0 to 2039.



Example ST :

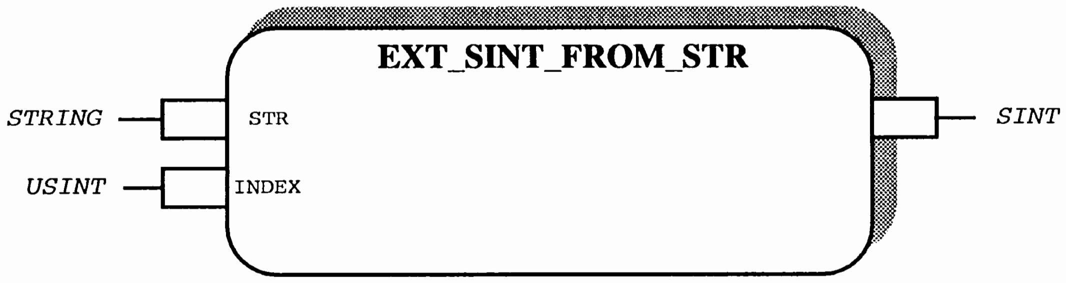
```
str1.Val  :=  REP_BOOL_IN_STR  (  STR  :=  str1.Val,
                                   INDEX  :=  300,
                                   VAL  :=  1  (*  TRUE  *)  );
```

The 301th bit packed into string str1.Val is set to 1, i.e. TRUE.

EXT_SINT_FROM_STR

Extracts a signed short integer (SINT) value from a character string (STRING). Each value is stored as one character of the string, so a maximum of 255 Short integer values may be stored in a 255 character string. INDEX lies in the range 0 to 254.

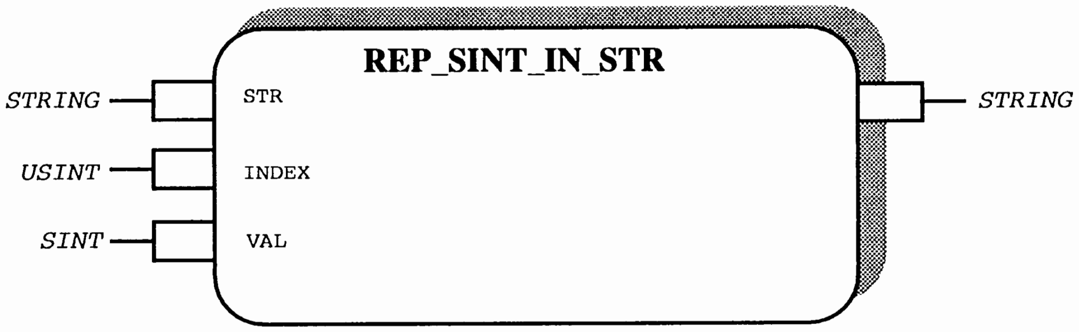
Returns a value in the range -128 to 127.



REP_SINT_IN_STR

Replaces a signed short integer (SINT) value in a character string (STRING). Each value is stored as one character of the string, so a maximum of 255 Short integer values may be stored in a 255 character string. INDEX lies in the range 0 to 254.

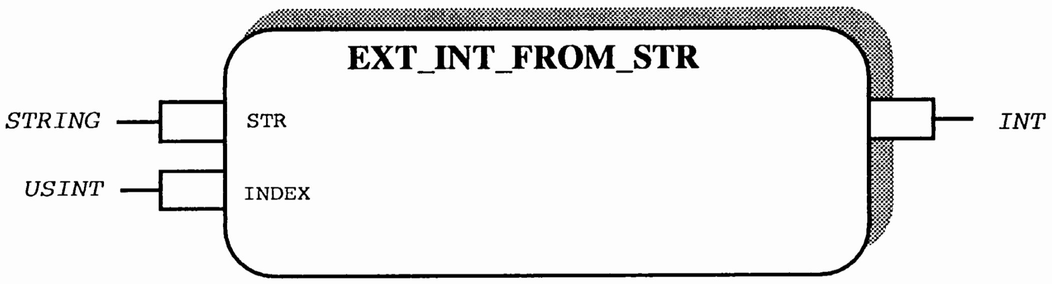
VAL lies in the range -128 to 127.



EXT_INT_FROM_STR

Extracts a signed integer (INT) value from a character string (STRING) string. Each value is stored as two characters of the character string (STRI NG), so a maximum of 127 signed integer values may be stored in a 255 character string. INDEX lies in the range 0 to 126.

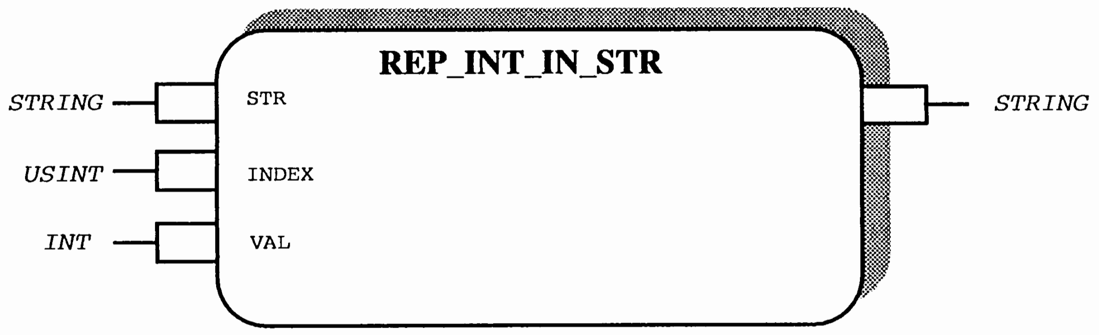
Returns a value in the range -32768 to 32767.



REP_INT_IN_STR

Replaces a signed integer (INT) value in a character string (STRING). Each value is stored as two characters of the string, so a maximum of 127 may be stored in a 255 character string. INDEX in the range 0 to 126.

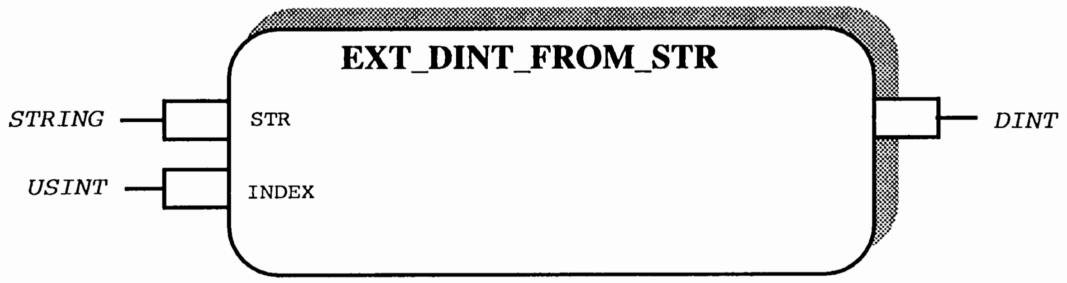
VAL in the range -32768 to 32767.



EXT_DINT_FROM_STR

Extracts a signed double integer (DINT) value from a character string (STRING). Each value is stored as four characters of the string, so a maximum of 63 may be stored in a 255 character string. INDEX in the range 0 to 62.

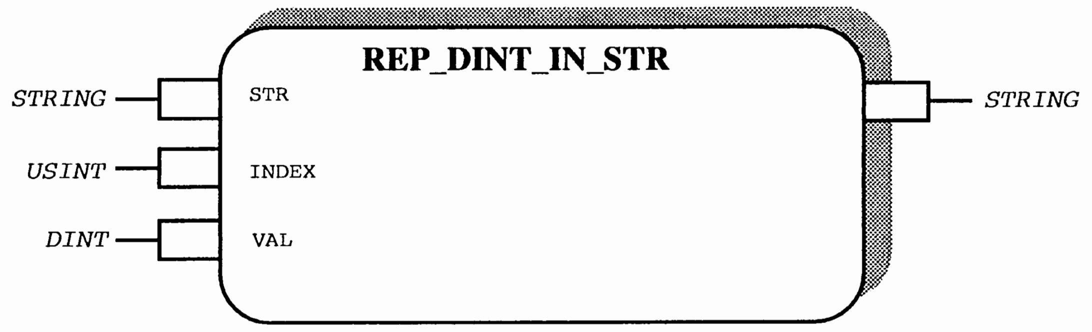
Returns a value in the range -2147483648 to 2147483647.



REP_DINT_IN_STR

Replaces a signed double integer (DINT) value in a character string (STRING). Each value is stored as four characters of the string, so a maximum of 63 may be stored in a 255 character string. INDEX in the range 0 to 62.

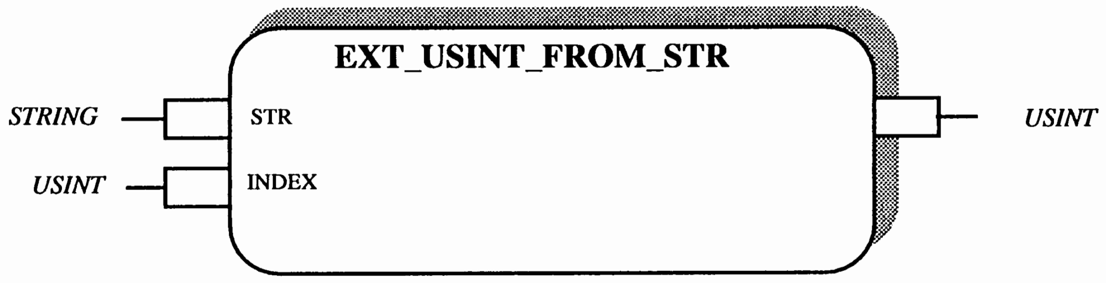
VAL in the range -2147483648 to 2147483647.



EXT_USINT_FROM_STR

Extracts an unsigned short integer (USINT) value from a character string (STRING). Each value is stored as one character of the string, so a maximum of 255 may be stored in a 255 character string. INDEX in the range 0 to 254.

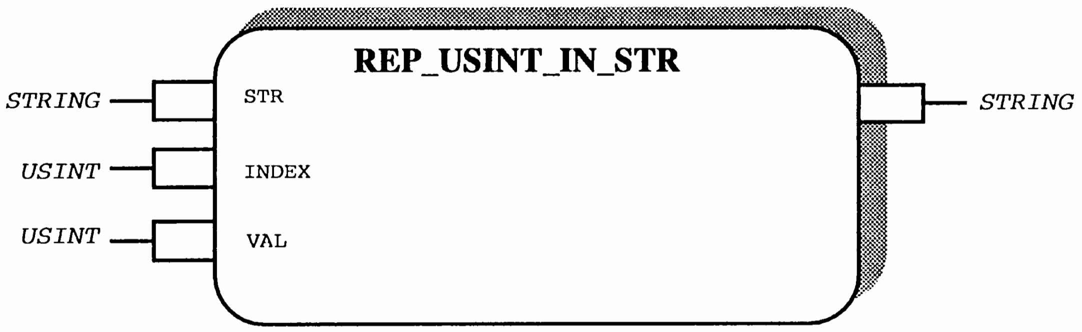
Returns a value in the range 0 to 255.



REP_USINT_IN_STR

Replaces an unsigned short integer (USINT) value in a character string (STRING). Each value is stored as one character of the string, so a maximum of 255 may be stored in a 255 character string. INDEX in the range 0 to 254.

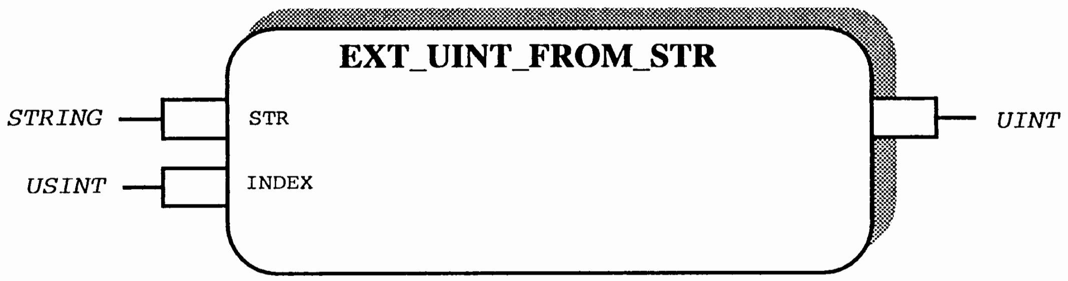
VAL in the range 0 to 255.



EXT_UINT_FROM_STR

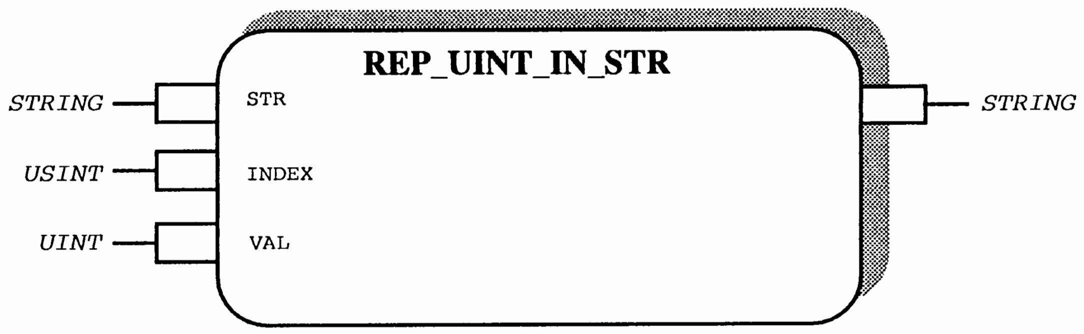
Extracts an unsigned integer (UINT) value from a string. Each value is stored as two characters of the string, so a maximum of 127 may be stored in a 255 character string. INDEX in the range 0 to 126.

Returns a value in the range 0 to 65535.



REP_UINT_IN_STR

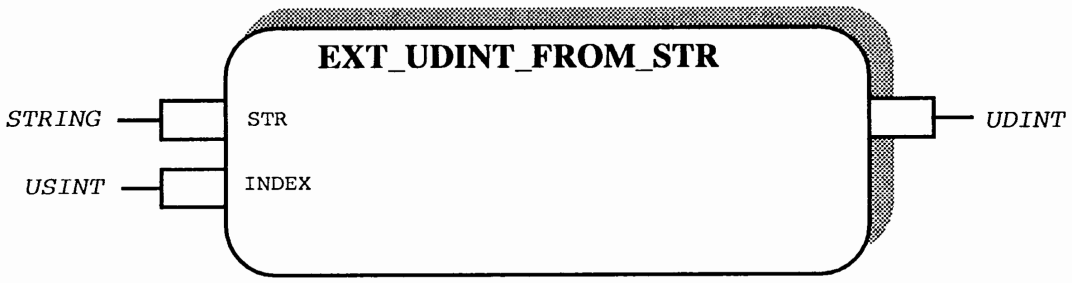
Replaces an unsigned integer (UINT) value in a character string (STRING). Each value is stored as two characters of the string, so a maximum of 127 may be stored in a 255 character string.



EXT_UDINT_FROM_STR

Extracts an unsigned double integer (UDINT) value from a character string (STRING). Each value is stored as four characters of the string, so a maximum of 63 may be stored in a 255 character string. INDEX in the range 0 to 62.

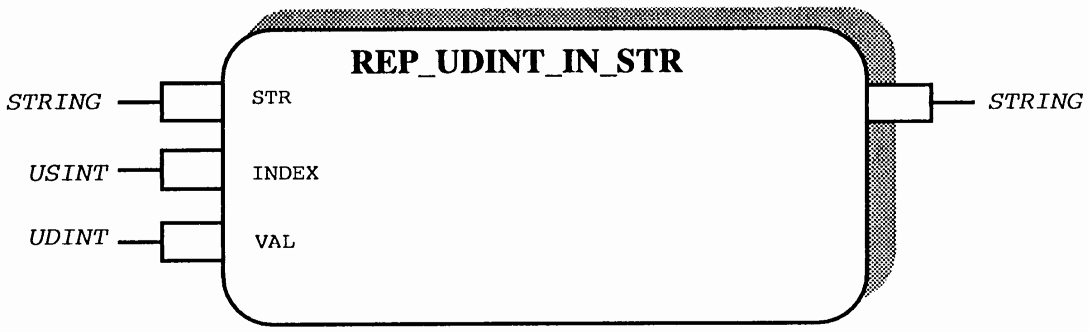
Returns a value in the range 0 to 4294967295.



REP_UDINT_IN_STR

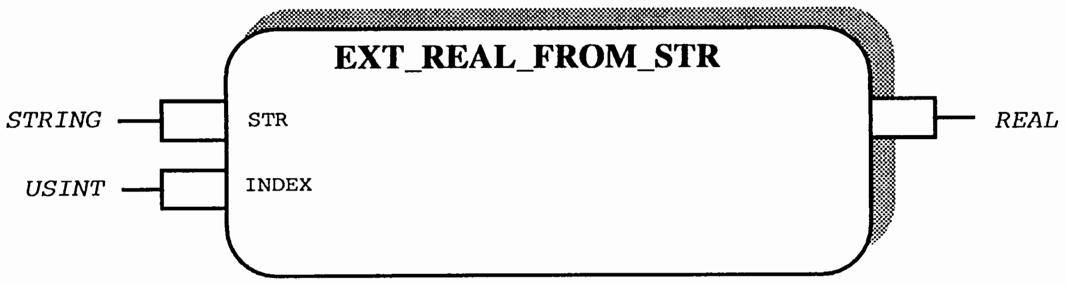
Replaces an unsigned double integer (UDINT)value in a character string (STRING). Each value is stored as four characters of the string, so a maximum of 63 may be stored in a 255 character string. INDEX in the range 0 to 62.

VAL in the range 0 to 4294967294.



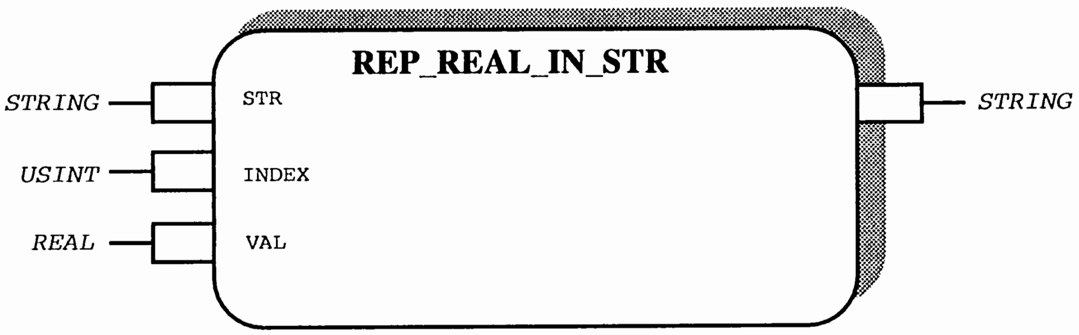
EXT_REAL_FROM_STR

Extracts a floating point (REAL) value from a character string (STRING). Each value is stored as four characters of the string, so a maximum of 63 may be stored in a 255 character string. The value is stored in IEEE Single Precision Binary Real Format. INDEX in the range 0 to 62.



REP_REAL_IN_STR

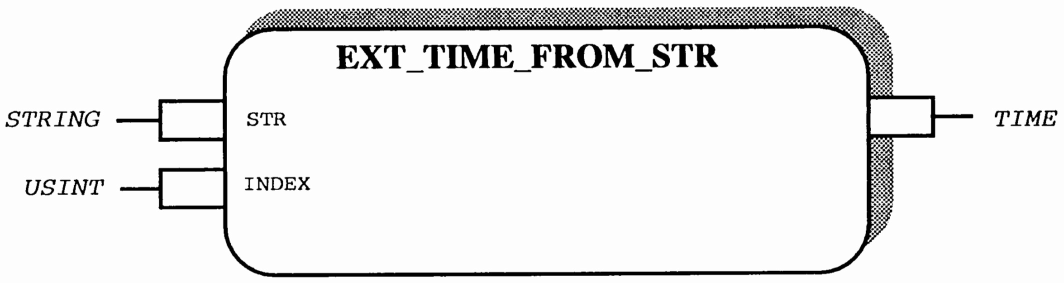
Replaces a floating point (REAL) value in a character string (STRING). Each value is stored as four characters of the string, so a maximum of 63 may be stored in a 255 character string. The value is stored in IEEE Single Precision Binary Real Format. INDEX in the range 0 to 62.



EXT_TIME_FROM_STR

Extracts a duration (TIME) value from a character string (STRING). Each value is stored as four characters of the string, so a maximum of 63 may be stored in a 255 character string. The value is stored as an integer number of milliseconds. INDEX in the range 0 to 62.

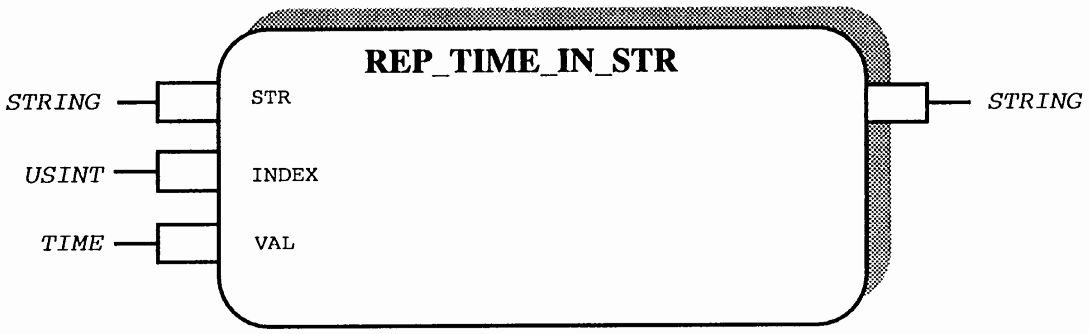
Returns a value in the range 0mS to 49d17h2m47s295ms.



REP_TIME_IN_STR

Replaces a duration (TIME) value in a character string (STRING). Each value is stored as four characters of the string, so a maximum of 63 may be stored in a 255 character string. INDEX in the range 0 to 62.

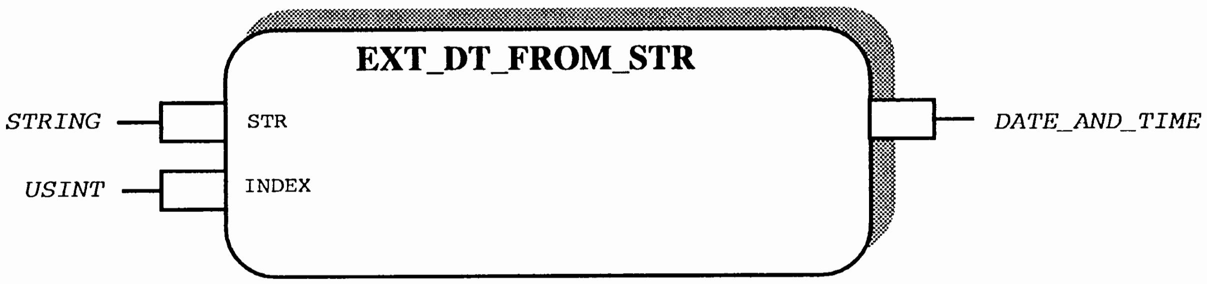
VAL in the range 0mS to 49d17h2m47s295ms.



EXT_DT_FROM_STR

Extracts a Date and Time value from a string. Each value is stored as four characters of the string, so a maximum of 63 may be stored in a 255 character string.

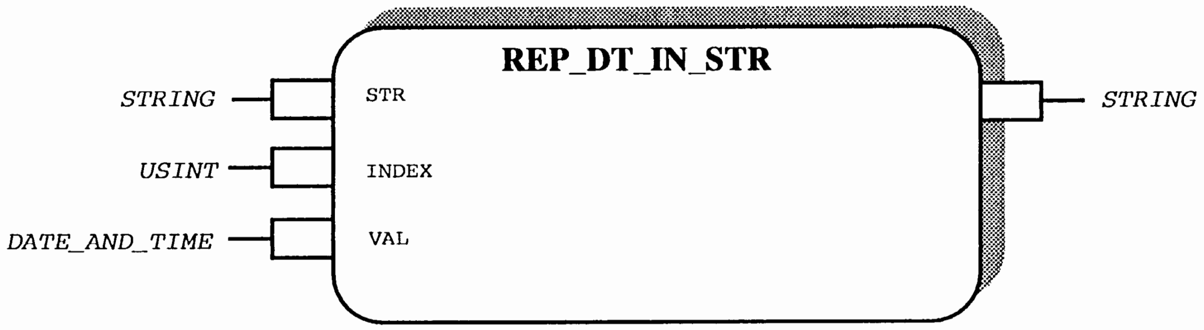
The value is stored as an integer number of seconds since 1970-01-01-00:00:00. INDEX in the range 0 to 62.



REP_DT_IN_STR

Replaces a date and time (DATE_AND_TIME) value in a character string (STRING). Each value is stored as four characters of the string, so a maximum of 63 may be stored in a 255 character string.

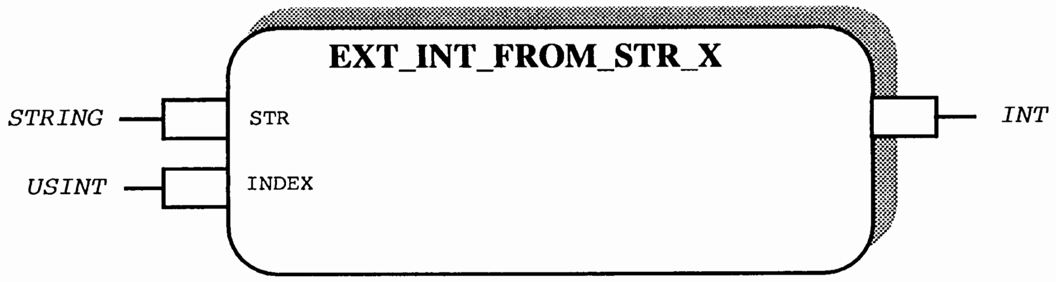
The value is stored as an integer number of seconds since 1970-01-01-00:00:00. INDEX in the range 0 to 62.



EXT_INT_FROM_STR_X

Extracts a signed integer value (INT), in reverse byte order, from a character string (STRING). Each value is stored as two characters of the string, so a maximum of 127 may be stored in a 255 character string. INDEX in the range 0 to 126.

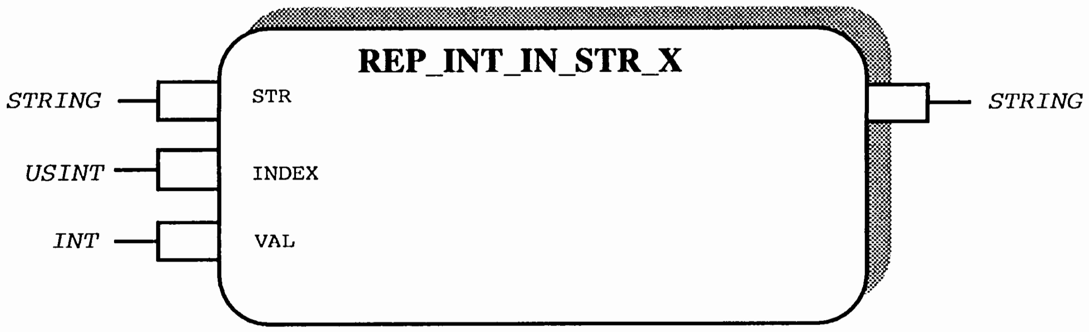
Returns a value in the range -32768 to 32767.



REP_INT_IN_STR_X

Replaces a signed integer (INT) value, in reverse byte order, in a character string (STRING). Each value is stored as two characters of the string, so a maximum of 127 may be stored in a 255 character string. INDEX in the range 0 to 126.

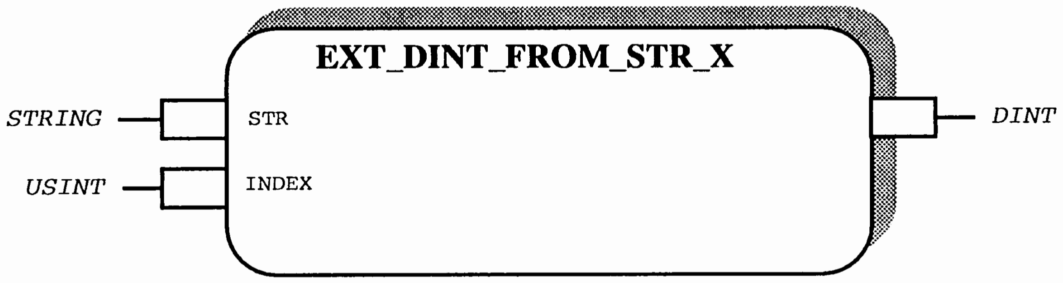
VAL in the range -32768 to 32767.



EXT_DINT_FROM_STR_X

Extracts a signed double integer (DINT) value, in reverse byte order, from a character string (STRING). Each value is stored as four characters of the string, so a maximum of 63 may be stored in a 255 character string. INDEX in the range 0 to 62.

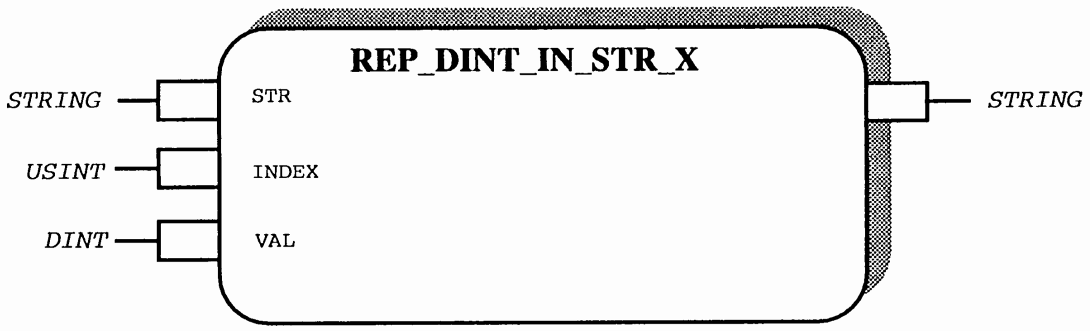
Returns a value in the range -2147483648 to 2147483647.



REP_DINT_IN_STR_X

Replaces a signed double integer (DINT) value, in reverse byte order, in a character string (STRING). Each value is stored as four characters of the string, so a maximum of 63 may be stored in a 255 character string. INDEX in the range 0 to 62.

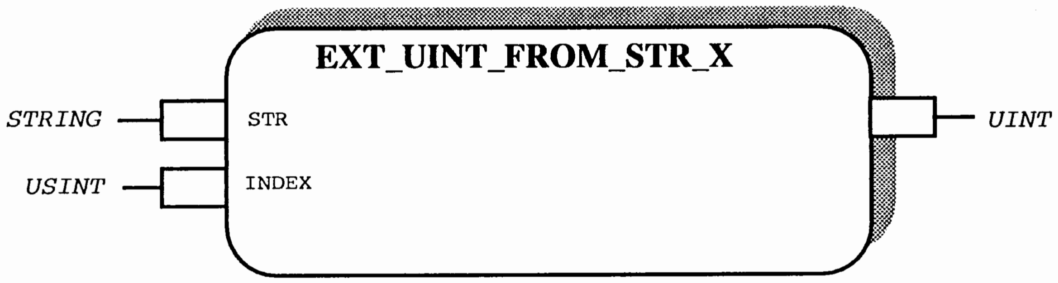
VAL in the range -2147483648 to 2147483647.



EXT_UINT_FROM_STR_X

Extracts an unsigned integer (UINT) value, in reverse byte order, from a character string (STRING). Each value is stored as two characters of the string, so a maximum of 127 may be stored in a 255 character string. INDEX in the range 0 to 126.

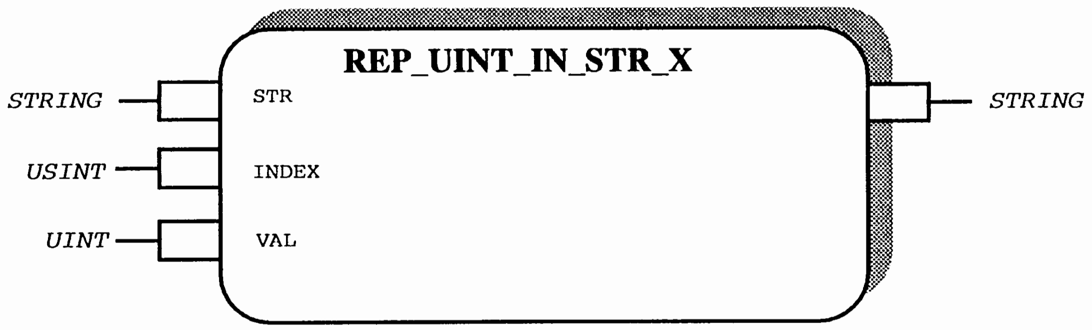
Returns a value in the range 0 to 65535.



REP_UINT_IN_STR_X

Replaces an unsigned integer (UINT)value, in reverse byte order, in a character string (STRING). Each value is stored as two characters of the string, so a maximum of 127 may be stored in a 255 character string. INDEX in the range 0 to 126.

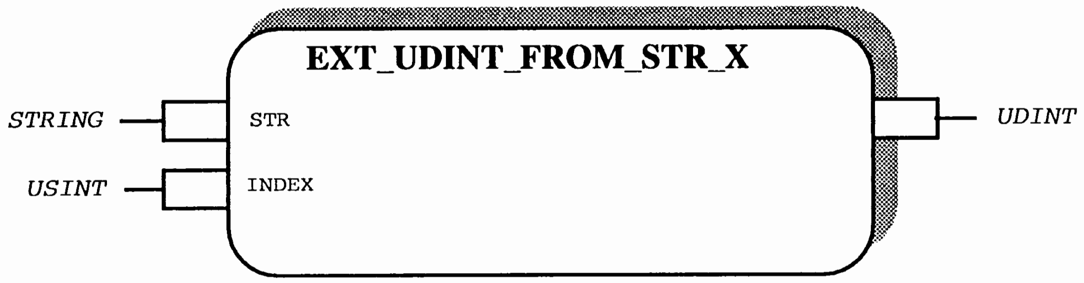
VAL in the range 0 to 65535.



EXT_UDINT_FROM_STR_X

Extracts an unsigned double integer (UDINT) value, in reverse byte order, from a character string (STRING). Each value is stored as four characters of the string, so a maximum of 63 may be stored in a 255 character string. INDEX in the range 0 to 62.

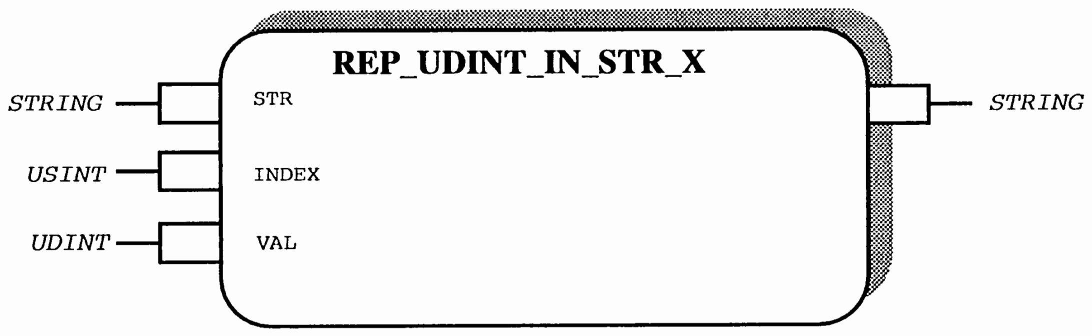
Returns a value in the range 0 to 4294967294.



REP_UDINT_IN_STR_X

Replaces an unsigned double integer (USINT) value, in reverse byte order, in a character string (STRING). Each value is stored as four characters of the string, so a maximum of 63 may be stored in a 255 character string. INDEX in the range 0 to 62.

VAL in the range 0 to 4294967294.



Chapter 9

COMMUNICATIONS SUPPLEMENTARY FUNCTIONS

Edition 2

Contents

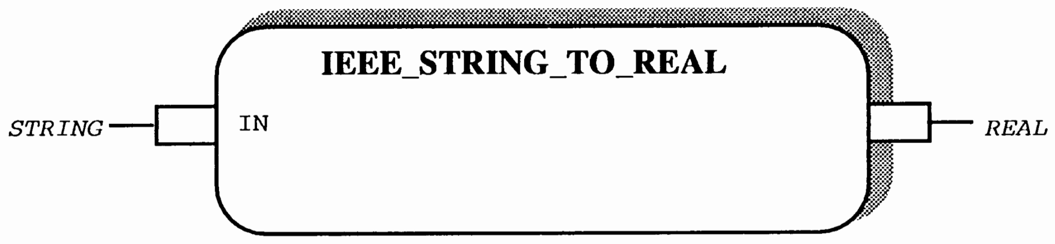
Overview

Two functions are provided to pack and unpack IEEE numbers stored in strings using the EI Bisync IEEE encoding format. These functions are provided to support the encoding or decoding of parameter values for transmission using EI Bisync protocol.

They will only be needed for specialist applications where it is necessary to construct or decode composite messages. In the majority of applications, the encoding and decoding of parameter values are handled automatically be the EI Bisync driver function block.

IEEE_STRING_TO_REAL

Converts an IEEE floating point (REAL) variable packed into a character string (STRING) using the EI Bisync encoding format, to a floating point (REAL) variable. See the PC3000 Communications Overview document and the chapter covering "EI Bisync Slave " Communication in the "PC3000 Function Block Reference" for details on the encoding of IEEE values.



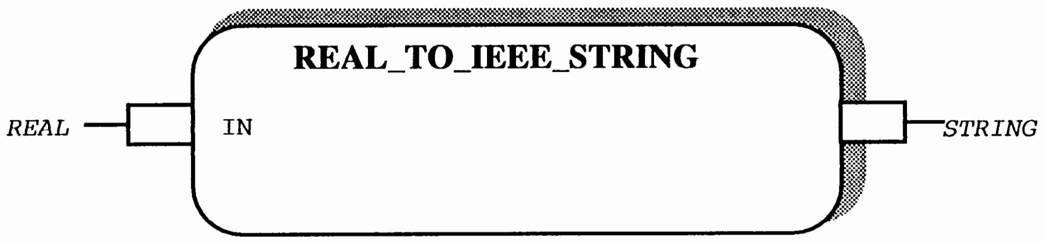
Example ST :

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

Sets real1.Val to -0.5

REAL_TO_IEEE_STRING

Converts a floating point (REAL) variable into a character string (STRING) variable using the EI Bisync IEEE encoding.



Example ST :

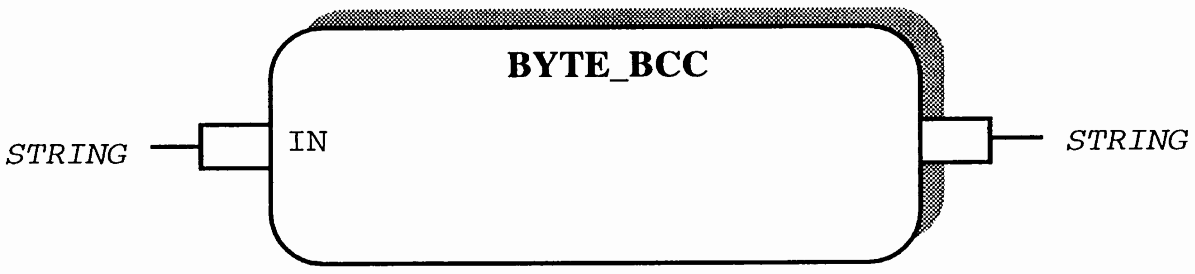
```
    str1.Val:=  IEEE_REAL_TO_STRING(IN:=  -0.5);
```

Sets str1.Val to '@op'

New Functions in v 3.12

BYTE_BCC (version 3.12 and later)

Returns a one character long string which is the exclusive-or of all the characters in the sting.



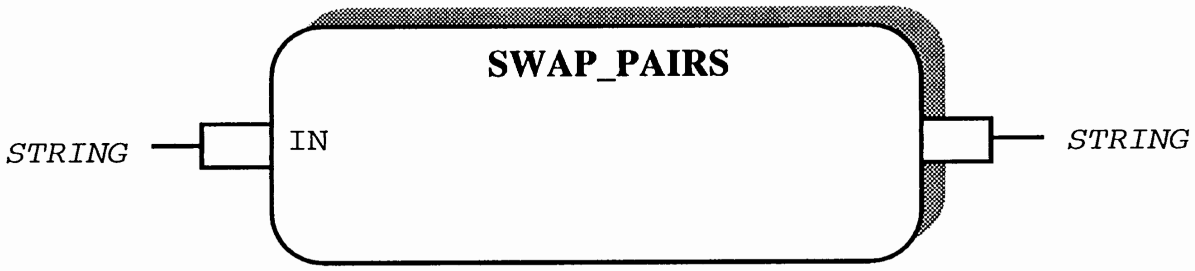
Example ST:

```
bcc.Val := BYTE_BCC (IN:='AbCdE');
```

Sets bcc. Val to 'A'

SWAP_PAIRS (version 3.12 and later)

Returns a copy of the input string with each pair of characters reversed. When the input string has a non-even number of characters an extra null is inserted.



Example ST:

```
swaped. Val:= SWAP_PAIRS (IN: 'AbCdE');
```

Sets swaped. Val to 'bAdC$00E'

Chapter 10

Bit Manipulation Functions

Edition 1

Contents

Overview

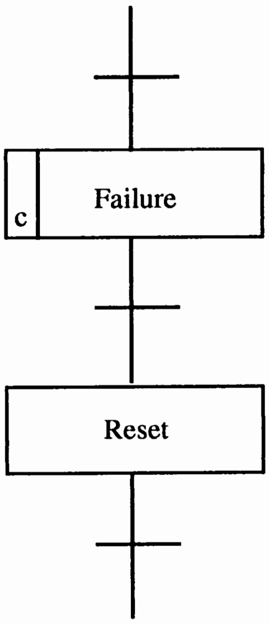
It is often the case that integers are interpreted as bit patterns. This occurs particularly when serial communications handling has to be done at a low level, or where alarm states are being assembled into reason codes.

To make this use of integers straightforward the BIT_MANIPUL class of functions enables individual bits in integers to be set, individual bits to be extracted from integers, and integer bit patterns to be combined and compared using AND, OR, XOR and NOT operations in a bitwise fashion.

For example, in a batch process we may wish to monitor all the machines to see if any failures occur during a particular batch, and then to reset this record at the end of the batch. If the states of the various machines arrive at PC3000 as digital inputs, and the status of various machines on the plant is represented by a single integer value which is monitored by a supervisory computer, we might find the following forming part of the sequence program:

Sequence program:

...from start of batch steps



...to the rest of end of batch steps

ST listing:

```
(* 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
```

Note that this program gives a different effect to using a BoolToDint function block. Using this program the various faults are “latched in” to the dint variable “failures”, whereas using a BoolToDint function block would only allow the continuously updating fault situation to be produced.

In a similar way, the value of one bit in an integer value could be captured at a particular point in the process cycle. This integer might be from a continuous stream of integer values being generated by a simple communicating device. At the moment in the cycle when the value should be captured a step containing the following line of ST would be executed:

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

More complex operations could be carried out by also utilising the AND, OR, XOR and NOT bit manipulation functions.

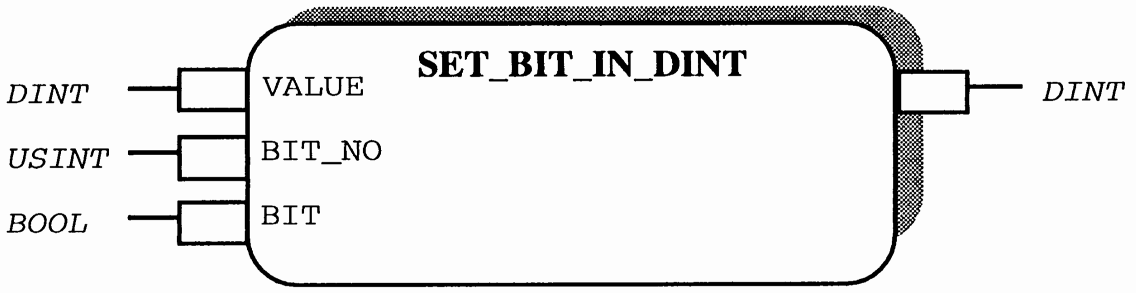
DOUBLE INTEGERS (DINTS)

Unlike many programmable logic controllers (PLCs), the mathematics inside the LCM uses signed 32-bit integers in two's complement mode. Bit numbers run from zero (least significant) to 31 (most significant). Bits zero to 30 have a binary weight equal to their bit number and are positive. For example, the decimal number 123 represents bit numbers 0, 1, 3, 4, 5 and 6 being set, and the decimal number 1,073,741,824 represents bit number 30 being set. However, bit thirty one carries a negative sign as well as a binary weight of thirty one. So if bit thirty one alone is set then the integer has a decimal value of -2,147,483,648. If all bits are set then the integer has a decimal value of -1.

It is important for those used to working with unsigned sixteen bit integers to understand this difference, particularly where comparisons are being made using the decimal values of bit patterns.

SET_BIT_IN_DINT (version 3.0 and later)

Sets one bit in a signed double integer (DINT). The DINT on which the operation is to be carried out, the bit to be set, and the state that this bit should be set to are supplied as arguments to the function. The function may be used to modify a dint, or to transfer a modified value to a different dint.



Example of modifying a dint:

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

the third bit of "dint" is set to on (1).

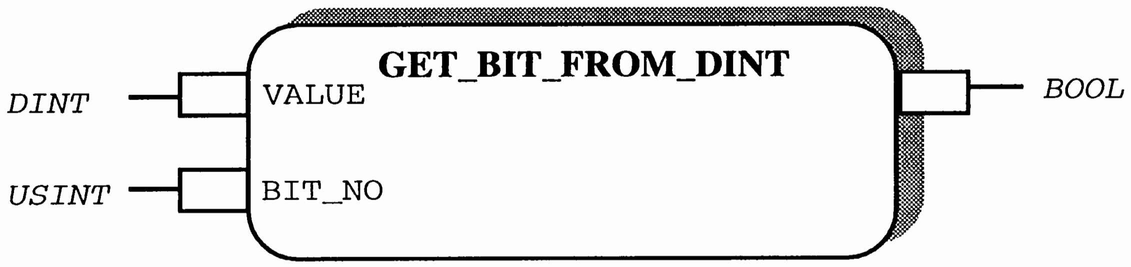
Example of passing a modified value to another dint:

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

the value of "thisone" is passed to "other" with the fifth bit set, whatever its state in "thisone".

GET_BIT_FROM_DINT (version 3.0 and later)

Extracts one bit from a signed double integer (DINT). The DINT on which the operation is to be carried out and the bit number to be extracted are supplied as arguments to the function. The result of the operation is a boolean value (BOOL).



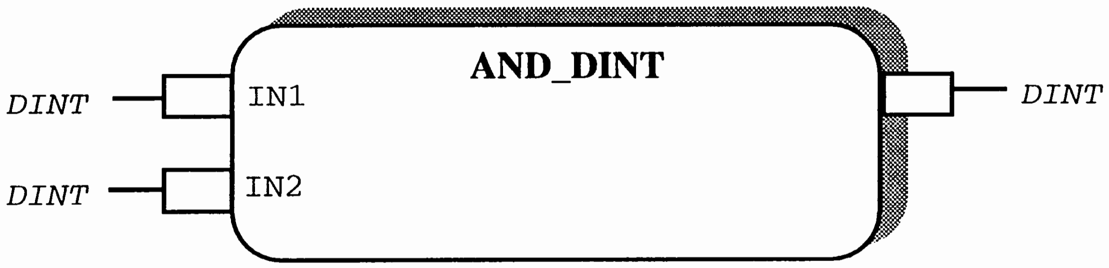
Example ST:

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

if the third bit of dint is set, (i.e. the value 8 is part of the binary makeup of dint), then bool.Val will be true(1).

AND_DINT (version 3.0 and later)

Allows two double integers (DINTs) to be logically "AND"ed with each other, bit by bit. Every bit that is true in both of the input integers will be set in the resultant of this function, and any bit that is only true in one or other of the input integers will not be set in the resultant.



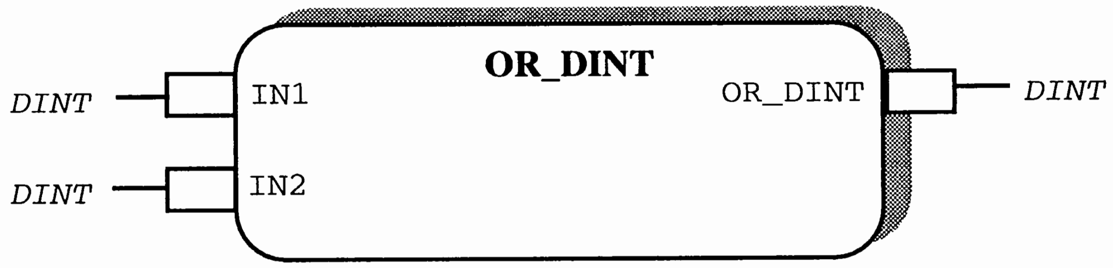
Example ST:

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

if dint1 is 100 (bits 2, 5 and 6 set) and dint2 is 90 (bits 1, 3, 4 and 6 set), then dintres will be 64 (bit 6 set, binary 1000000).

OR_DINT

Allows two double integers (DINTs) to be logically, non-exclusively "OR"ed with each other, bit by bit. Every bit that is true in one or other or both of the input integers will be set in the resultant of this function, and any bit that is not set in either of the input integers will not be set in the resultant.



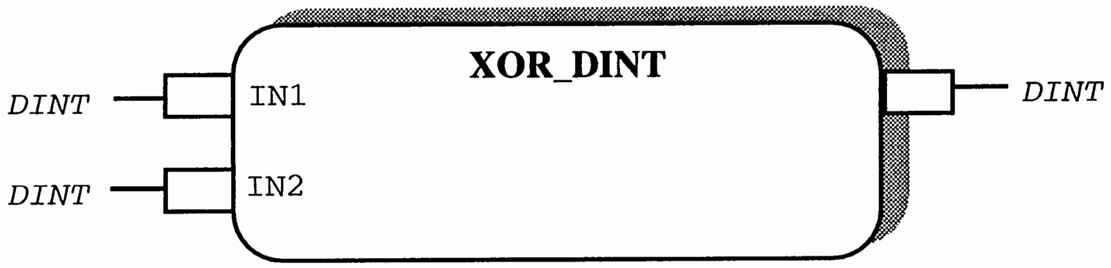
Example ST:

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

if dint1 is 100 (bits 2, 5 and 6 set) and dint2 is 90 (bits 1, 3, 4 and 6 set), then dintres will be 126 (bits 1 to 6 set, binary 1111110).

XOR_DINT

Allows two double integers (DINTs) to be logically, exclusively "OR"ed with each other, bit by bit. Every bit that is true in either one or other but not both of the input integers will be set in the resultant of this function, and any bit that is not set in either of the input integers will not be set in the resultant.



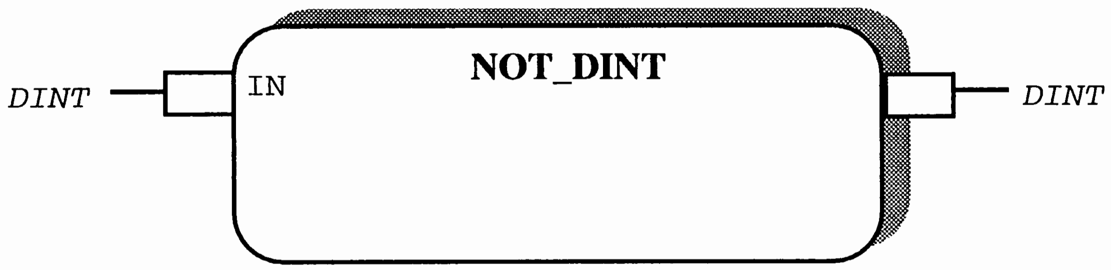
Example ST:

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

if dint1 is 100 (bits 2, 5 and 6 set) and dint2 is 90 (bits 1, 3, 4 and 6 set), then dintres will be 62 (bits 1 to 5 set, binary 111110).

NOT_DINT

Allows a single double integer (DINT) to be logically "NOT"ed. Every bit that is true in the input integer will not be set in the resultant of this function, and any bit that is not set in the input integer will be set in the resultant.



Example ST:

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

if dint is 100 (bits 2, 5 and 6 set) then dintres will be -101 (bits 0, 1, 3, 4 and 7 to 31 set, binary 11111111111111111111111110011011).

Chapter 11

Generic Functions

Edition 1

Contents

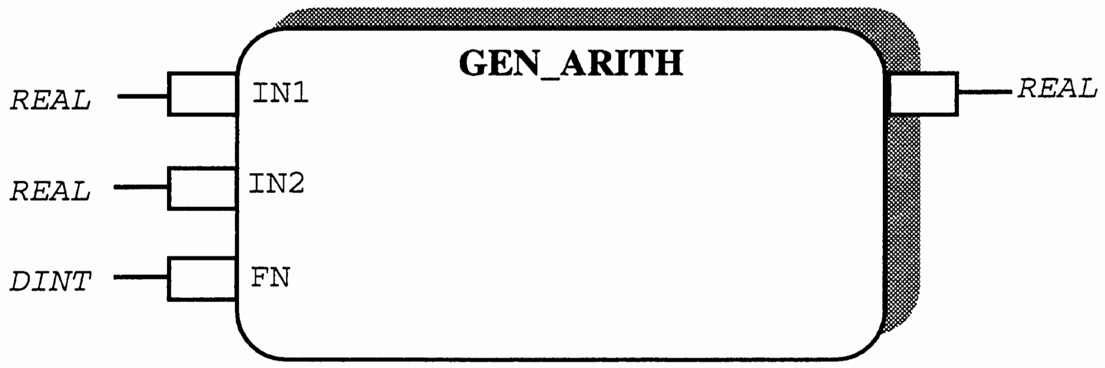
Overview

The generic functions described in this section permit applications to be written where the arithmetic, comparison or boolean combination function required is not known at the time of configuration. This permits more generic coding of applications which can then be tailored to specific needs at commissioning time. It also allows easy implementation of applications which must perform differently depending on the recipe being run by the application.

GEN_ARITH (version 3.12 and later)

The GEN_ARITH function permits any one of the four basic arithmetic functions to be performed on two values selected by a third input to the function. The functions performed are as follows -

fn	GEN_ARITH(IN1:=x; IN2:=y; FN:=fn)
0	x+y
1	x-y
2	x*y
3	x/y
Other	0.0



Example ST:

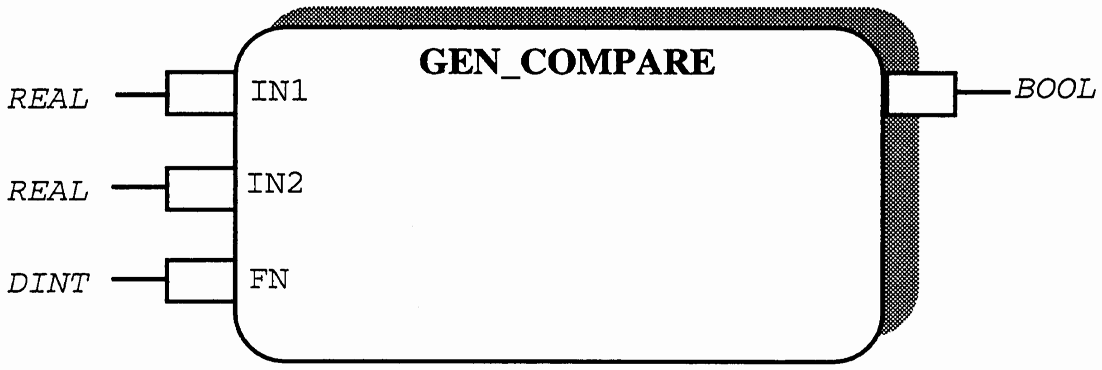
```
result.val := GEN_ARITH( IN1:=1.5 ; IN2:=2.5 ; FN:=2 );
```

sets result.val to 3.75

GEN_COMPARE (version 3.12 and later)

The GEN_COMPARE function permits any one of six comparisons to be performed on two values selected by a third input to the function. The comparisons performed are as follows -

fn	GEN_COMPARE(IN1:=x; IN2:=y; FN:=fn)
0	x=y
1	x>y
2	x<y
3	x≥y
4	x≤y
5	x≠y
Other	False (0)



Example ST:

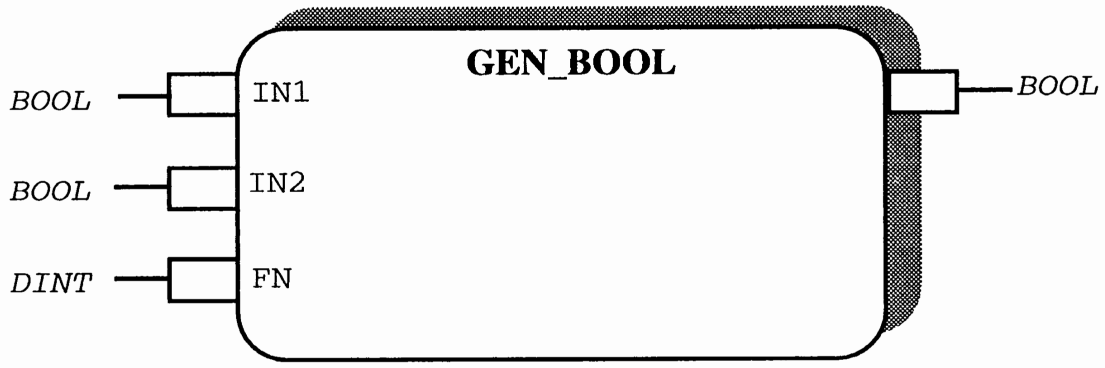
```
result.val := GEN_COMPARE( IN1:=2.5 ; IN2:=1.5 ; FN:=2 );
```

sets result.val to False (0)

GEN_BOOL (version 3.12 and later)

The GEN_BOOL function permits any one of six boolean functions to be performed on two values selected by a third input to the function. The functions performed are as follows -

fn	GEN_BOOL(IN1:=x; IN2:=y; FN:=fn)
0	x AND y
1	x OR y
2	x XOR y
3	NOT (x AND y)
4	NOT (x OR y)
5	NOT (x XOR y)
Other	False (0)



Example ST:

```
result.val := GEN_BOOL( IN1:=0 ; IN2:=1 ; FN:=2 );
```

sets result.val to True (1)

APPENDIX A FUNCTIONS EXECUTION TIMES

The timing results of the functions vary depending on the operands to the functions, the length of strings, etc. These figures should be regarded as approximate. Together with the data in Appendix B, these figures may be used to estimate the execution time of a specific block of Structured Text.

FUNCTION AND EXECUTION TIME (µS)		**FUNCTION AND EXECUTION TIME (µS)**	
NUMERICAL		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	**TYPE CONVERSION**	
SEL_REAL	5.9	DINT_TO_REAL	46
SEL_TIME	4.7	REAL_TO_DINT	313
SEL_DATE	4.7	TRUNC	59
SEL_TIME_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

FUNCTION AND EXECUTION TIME (μS)		FUNCTION AND EXECUTION TIME (μS)	
DATE_AND_TIME_TO_TOD	79	**TIME ARITHMETIC**	
DATE_AND_TIME_TO_DT	102	ADD_DATE_AND_TIME_T	82
CONCAT_TO_DINT	3.8	SUB_DATE_AND_TIME_T	82
STRING CONVERSION		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_X	93
CHAR_TO_ASCII	6.3	REP_INT_IN_STR_X	129
EXT_DINT_FROM_STR_X	99	**COMMS SUPPLEMENTARY**	
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		

Note 1: Typical figure, depends upon string length.

APPENDIX B OPERATOR EXECUTION TIMES

The execution times for operators are shown in Table 1 below:

Table 1 Execution times

Operator	**Real Operands µS**	**Integer Operands µS**	**Boolean Operands µS**
+	66	2.6	N/A
x	66	23	N/A
-	66	2.6	N/A
/	72	43	N/A
>	31	6.5	N/A
<	31	5.7	N/A
=	31	5.8	N/A
<>	31	6.8	N/A
>=	31	6.8	N/A
<=	31	5.8	N/A
- (monadic)	8	3.7	N/A
mod	N/A	46	N/A
and	N/A	N/A	6
or	N/A	N/A	6
xor	N/A	N/A	5
not	N/A	N/A	5

Note: In evaluating an expression such as,

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

There are additional overheads associated with converting the real operands from single precision to double precision format prior to evaluation. The resultant double precision value, real1.Val, is then converted back into a single precision floating point format.

This conversion is performed because internally, all ST statements are converted to "C" prior to compilation. The compiler used in the Programming Station does not support the single precision format. All calculations are performed using the double precision format.

The additional overheads are as follows:

Operation	Add
single format to double format	17µs
double format to single format	30µs

In the example given the total time is:

```
    reall.Val    :=  reall2.Val  +  reall3.Val    ;
```

30µs (double to single)   17µs (single to double)   66µs (+)   17µs (single to double)

Total: 30 + 17 + 66 + 17 = 130µs

It should be noted that all conversions occur automatically. This information is provided in order to enable the program developer to predict the execution time in a specific section of ST.

Table 2 Order of precedence for operators

No.	Operation	Symbol	Precedence
1	Parenthesization Function Evaluation	(expression) identifier (argument list) E.g. LN(A), MAX(X,Y) etc	HIGHEST
2	Exponentiation	** Note: A**B=EXP(B*LN(A))	
3	Negation Complement	- NOT	
4	Multiply Divide Modulo	* / MOD	
5	Add Subtract Comparison	+ - <, >, <=, >=	
6	Equality Inequality	= <>	
7	Boolean AND	&, AND	
8	Boolean Exclusive OR	XOR	
9	Boolean OR	OR	LOWEST

APPENDIX C ERROR CODES

The PC3000 provides a system error log for tracking real time events such as power failure, system restart, communications failures and other diagnostic information All error codes are time stamped at the time of occurrence.

Error codes associated with mathematical operators are also generated and are entered into the log in real time. Mathematical errors may include division by zero or overflow. This type of error may occur within the execution of the Function Blocks, the 'soft' wiring or the sequence program of the user program.

In all cases the entry in the log will be represented as

<5xx> <maths operation code> <diagnostic information>

The mathematical operation codes are tabulated separately and provide an indication of the type of operation which caused the error.

The diagnostic information is provided for use internally by Eurotherm Controls Limited.

In the event of errors the user program should be investigated for conditions that result in illegal values or invalid operations such as divide by zero.

Table 1 Error codes description

Error Code	Error description	Field 1	Field 2
500	**Error** A mathematical operation involving a floating point (REAL) value has resulted in an infinite result. **Cause** A mathematical operation such as division by zero has been performed which gave an infinite result. Once an infinity has occurred it may propagate through many mathematic operations causing multiple errors.	See table 2	

Error Code	Error description	Field 1	Field 2
501	**Error** A REAL mathematical operation was performed with an illegal operand. **Cause** A mathematical operation has been performed on a value which is not valid such as ASIN (IN :=2). Such an operation will result in the generation of an invalid value known as IEEE "not a number" or NAN (similar to an infinity). This may propogate throughout many maths operations causing multiple errors to be reported.	See table 2	diagnostic information
502	**Error** A mathematical operation on a REAL has resulted in an overflow. **Cause** A mathematical operation has been performed which results in a value which is too large to be represented. Such an operation will result in an IEEE infinity and it is likely that it will propogate through many mathematical operations causing multiple errors to be reported.	See table 2	diagnostic information
550	**Error** INTEGER division by zero. **Cause** An attempt has been made to divide an integer by zero.	0	diagnostic information

Table 2 Mathematical operation codes

Operation Code	**Mathematical Operation Description**
1	double to integer conversion
2	double to single precision conversion
3	single to integer conversion
11	single precision integer addition
12	single precision subtraction
13	single precision multilpication
14	single precision division
21	double precision addition
22	double precision subtraction
23	double precision multiplication
24	double precision division
25	double precision sqrt
26	double precision ln
27	double precision log
28	double precision exp [1]
29	double precision sin
30	double precision cos
31	double precision tan
32	double precision asin
33	double precision acos
34	double precision atan
35	double precision mod
36	double precision expt [2]

Note 1. Natural exponent

Note 2. Exponentiation

Note 3. In table 2, "single" and "double" precision refers to the 32 and 64 bit words used to store IEEE floating point values. PC3000 uses both of these types of IEEE number format in different parts of the user program.

APPENDIX D RULES FOR MIXED DATA TYPES

The PC3000 supports a subset of the data types defined by the IEC DIS 1131 as part of the Structured Text (ST) language definition. These data types apply to Function Block parameters and to variables created by the user by means of the Function Block class, **USER_VAR**.

Keyword is the term used to describe a group of characters which have a special meaning in ST. The data types supported are as follows:

Keyword	**Data type**	**Bits**	**Range**
IO_ADDRESS	Channel address	32	1:01: to 12:14 (note 2)
BOOL	Boolean	1	0 or 1
REAL	Real Number	32	$\pm 10^{\pm 38}$
SINT	Short Integer	8	-128 to 127
USINT	Unsigned Short Integer	8	0 to 255
INT	Integer	16	-32768 to 32767
UINT	Unsigned Integer	16	0 to 65535
DINT	Double Integer	32	-2147483648 to 2147483647
UDINT	Unsigned Double Integer	32	0 to 42944967295
TIME	Duration	32	Up to 49 days
TIME_OF_DAY DATE	Time of Day Date	32 32	00:00:00 to 23:59:59 01-Jan-1970 to 01-Jan-2136
DATE AND TIME	Date AND Time of Day	32	01-Jan-1970-00:00:00 to 01-Jan-2136-23:59:59
STRING	Variable length character string		(Note 3)
ENUM	Enumerated item	32	0 to 2^{32} (Note 4)

Note 1 Although the data types SINT, USINT, INT and UINT are all listed they are treated as 32 bits within the PC3000 thus ensuring compatibility with other products.

Note 2 IO_ADDRESS is a special data type used to identify the address of an input or output hardware channel. It takes the form:

<RACK> : <SLOT> : <CHANNEL>
(1 - 8) (1 - 12) (1 to max channel no.):

MODULE Function Blocks have IO_ADDRESS represented as a character string (String) data type.

Note 3 String lengths: A 'String'(String) **USER_VAR** is limited to 80 characters. A 'Long String' **USER_VAR** may be up to 255 characters. Both are characterised as data type STRING.

Note 4 Enumerated data types appear on lists where a fixed number of options may be selected. Alternatively, a number of states may be reported using this data type. It takes the form :

Name 1 (0)

Name 2 (1)

etc.

Addition, subtraction etc. of enumerated parameters is invalid.

Expressions

An expression is a construct which, when evaluated will produce a value corresponding to one of the listed data types.

Expressions comprise operators (+, -, *, etc.) and operands which can be literals (2.5, 3, T#2s, etc.), variables (int1.Val, real1.Val, etc., or a Function Block parameter), a Function, or another expression.

In the IEC standard, 1131, all data types are strictly type checked in all expressions. This means that any expression must contain 'matched' data types or data types should be converted using the type conversion functions.

The PC3000 relaxes these rules such that:

A. `real1.Val:= real2.Val` *`operator`* `Int1.Val;`

Where *`operator`* `is *, +, - or /` produces a floating point (REAL) value

B. `Int1.Val:=real1.Val` *`operator`* `Int1.Val;`

Where *`operator`* `is *, +, - or /` produces a truncated integer value

It is however good programming discipline to ensure that all data types in an expression are of the same generic type.

This means that example A, would become:

C.

```
real1.Val:= real2.Val operator DINT_TO_REAL
        (IN:= Int1.Val);
```

produces a floating point (REAL) value

Expressions involving time data types

Use of the normal arithmetic operations is not supported such as:

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

✗

Time conversion functions should be used in expressions that require mathematical operations between time data types. For example:

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

✓

Time multiplication and division

Rules are similar to those illustrated for addition and subtraction. Time should be converted to an integer format prior to evaluation as part of an expression involving more than one time parameter.

APPENDIX E NESTING OF PC3000 ST FUNCTIONS

Functions may be nested to create complex control strategies such as automatic switchover of a 'redundant' thermocouple to protect against sensor failure.

Additionally type conversion functions may be used repeatedly to match data type within an expression.

Functions associated with string manipulation may be nested to create concatenated message strings for use with, say, a panel. It is recommended, however, that since string functions use large areas of the PC3000 system stack space, string function nesting is restricted to two deep. Greater nesting depth may be achieved by using the USER_VAR string Function Block to store the intermediate values.

The following examples illustrate nesting:

Example 1.

Automatic thermocouple switchover

```
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);
```

The input to the PID Function Block, loop1, is taken from one of two analogue inputs depending on which one has a GO status. If both sensors are NOGO, the PID input is forced to retain its previous value.

Example 2.

Type Conversion

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

Time type conversion functions used to convert durations (TIME) to integers (UDINT) prior to addition. The result is then converted to a duration (TIME).

Example 3.

String Functions

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

Display.Val is a string variable used to hold the concatenated string. This could be used to create an operator panel display.