Libraries for programing Tecomat PLC according to IEC 61 131-3

TXV 003 22.02
8th Issue
March 2006
All rights reserved

History of changes

Date	Issue	Description of changes
August 2004 to February 2006	1 To 7	The description of libraries is part of the document TXV 003 21.02 See TXV 003 21.02 for changes
March 2006	8	Library description transfered into separate document TXV 003 22.02

1 LIBRARIES

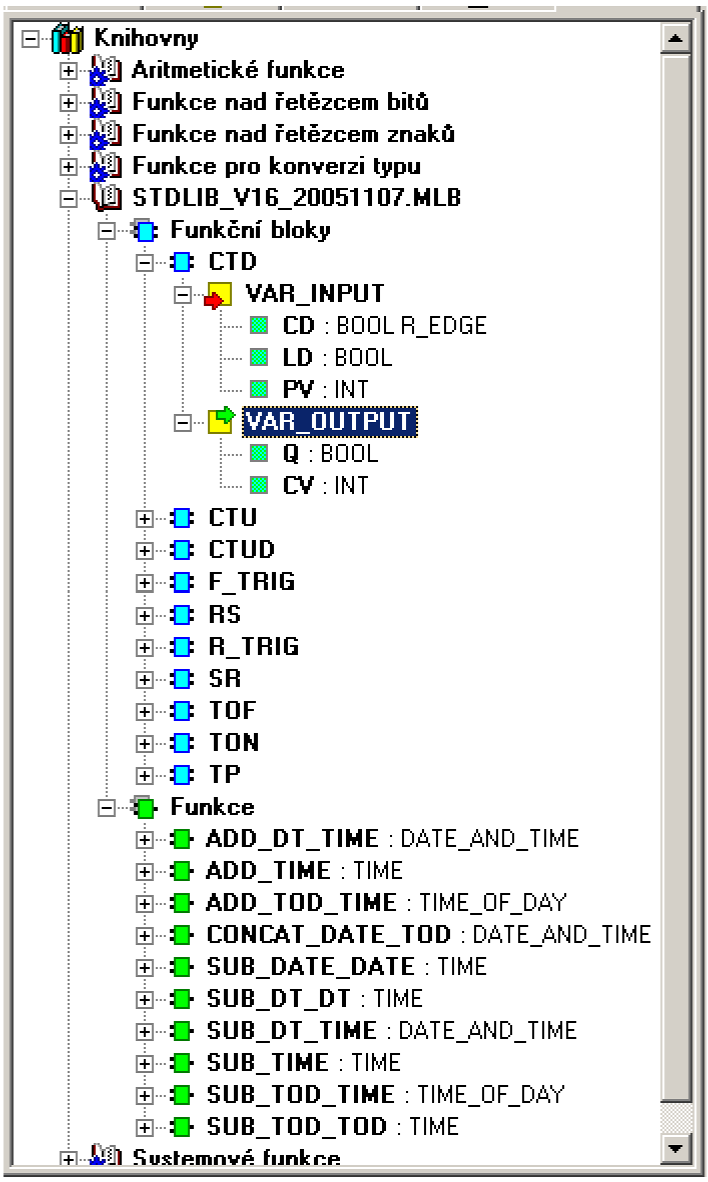
Libraries of functions and function blocks are an inseparable part of the installation of the Mosaic programming environment. It is possible to divide them into the following types:

- built-in libraries
- standardly supplied external libraries
- user defined libraries

A library may contain declarations of functions, function blocks, data types and global variables.

2 STANDARD STDLIB LIBRARIES

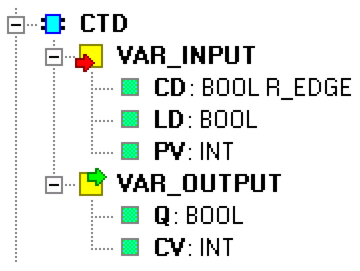
The standard library contains function blocks of counters CTD, CTU and CTUD, timers TON, TOF and TP, flip-flop circuits RS and SR and blocks of edges detection R_TRIG and F_TRIG. There is shown integration of function blocks to the library in the Mosaic development environment on the following example.



```
Knihovny
  Aritmetické funkce
  Funkce nad řetězcem bitů
  Funkce nad řetězcem znaků
  Funkce pro konverzi typu
  STDLIB_V16_20051107.MLB
    Funkční bloky
      CTD
        VAR_INPUT
          CD : BOOL R_EDGE
          LD : BOOL
          PV : INT
        VAR_OUTPUT
          Q : BOOL
          CV : INT
      CTU
      CTUD
      F_TRIG
      RS
      R_TRIG
      SR
      TOF
      TON
      TP
    Funkce
      ADD_DT_TIME : DATE_AND_TIME
      ADD_TIME : TIME
      ADD_TOD_TIME : TIME_OF_DAY
      CONCAT_DATE_TOD : DATE_AND_TIME
      SUB_DATE_DATE : TIME
      SUB_DT_DT : TIME
      SUB_DT_TIME : DATE_AND_TIME
      SUB_TIME : TIME
      SUB_TOD_TIME : TIME_OF_DAY
      SUB_TOD_TOD : TIME
  Systemové funkce
```

2.1 *Function block of counter down CTD*

Function block used for counting down CTD.



```
CTD
  VAR_INPUT
    CD : BOOL R_EDGE
    LD : BOOL
    PV : INT
  VAR_OUTPUT
    Q : BOOL
    CV : INT
```

Input variables :

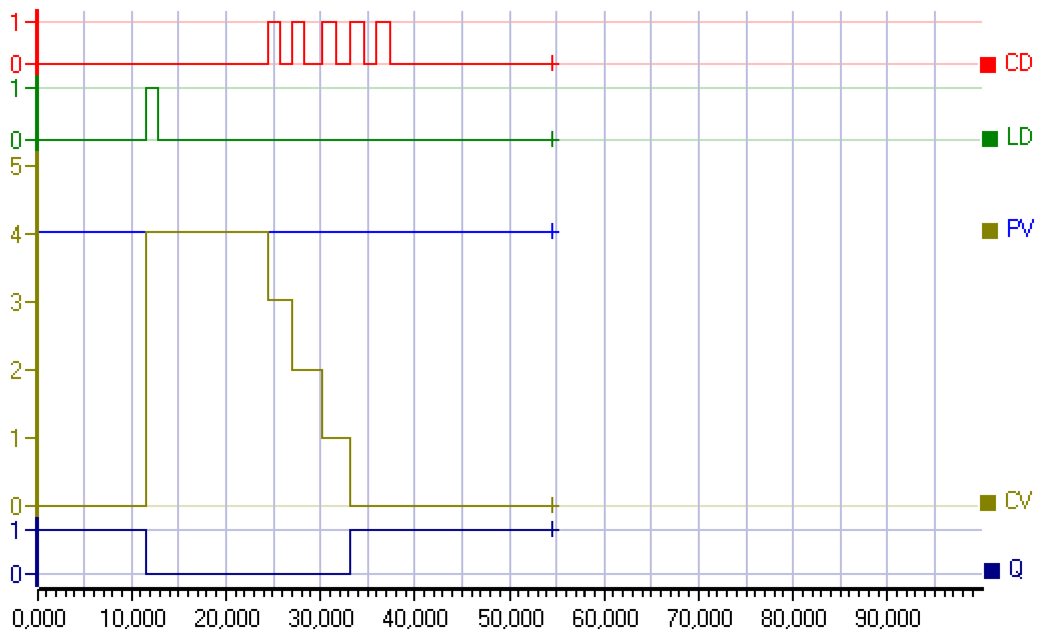
CD	input for counting down
LD	input for preset setting
PV	counter preset

Output variables:

Q	counter output
CV	counter value

If the value of input variable LD is TRUE, then the counter value CV is set to preset value given in PV variable. If the value of input variable LD is FALSE and the input variable CD changes the status from FALSE to TRUE (rising edge), the counter value CV is decreased by 1. If the counter value is equal to 0, the counter output Q is set to TRUE, otherwise it is FALSE.

The behavior of CTD counter is explained on the following figure.

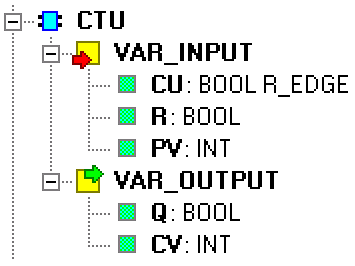


An example of program where function block CTD is called:

```
PROGRAM Counter
  VAR
    pulses          : BOOL;
    setCTD          : BOOL;
    counterCTD      : CTD;              // counter instance
    output          : BOOL;
  END_VAR

  counterCTD( CD := pulses, LD := setCTD, PV := 4, Q => output);
END_PROGRAM
```

2.2 *Function block of counter up CTU*

Function block used for counting up.

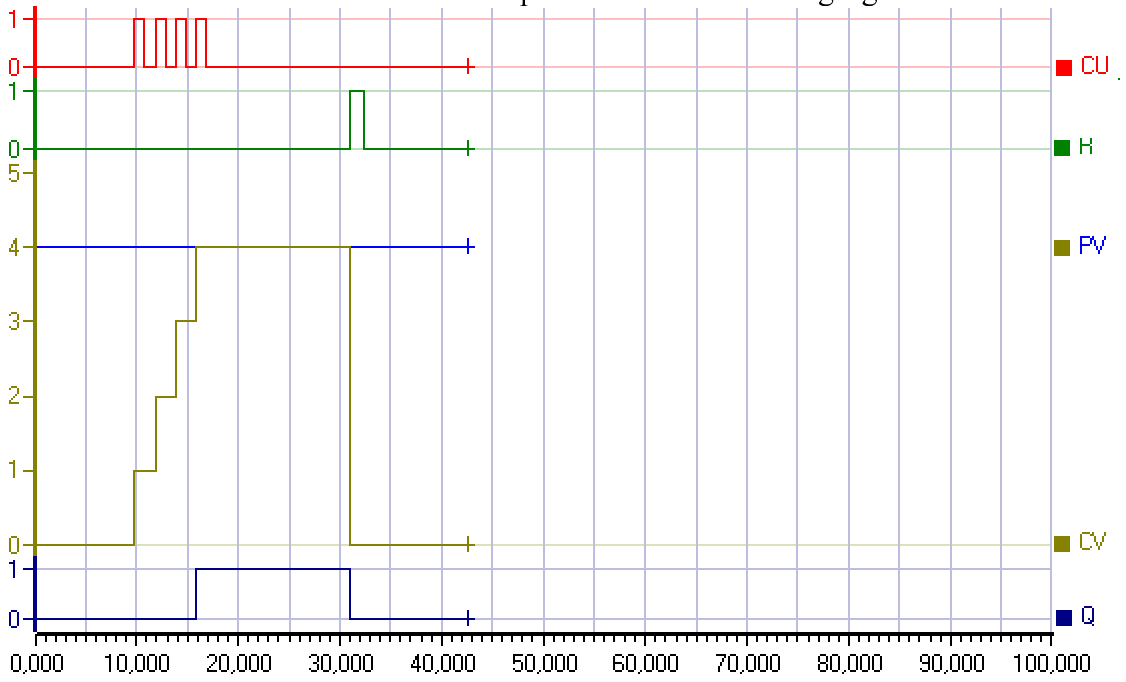


```
CTU
  VAR_INPUT
    CU : BOOL R_EDGE
    R : BOOL
    PV : INT
  VAR_OUTPUT
    Q : BOOL
    CV : INT
```

Input variables :	
CU	input for counting up
R	counter reset
PV	counter preset
Output variables	
Q	counter output
CV	counter value

If the value of input variable R is TRUE, the counter value is set to zero. If the value of input variable R is FALSE and input variable CU changes the status from FALSE to TRUE (rising edge), the counter value is increased by 1. If the counter value CV is greater or equal to preset value PV, the counter output Q is set to TRUE, otherwise it is FALSE.

The behavior of CTU counter is explained on the following figure.

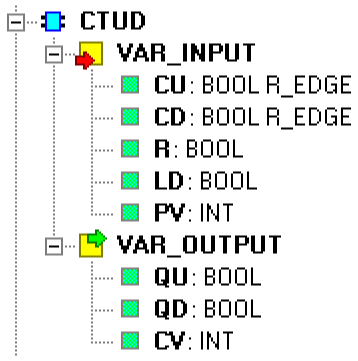


An example of program where function block CTU is called:

```
PROGRAM Counter
  VAR
    pulses          : BOOL;
    resetCTU       : BOOL;
    counterCTU     : CTU;                    // counter instance
    output         : BOOL;
  END_VAR

  counterCTU( CU := pulses, R := resetCTU, PV := 4, Q => output);
END_PROGRAM
```

2.3 *Function block of bidirectional counter CTUD*

Function block for bidirectional counting:



```
CTUD
  VAR_INPUT
    CU : BOOL R_EDGE
    CD : BOOL R_EDGE
    R : BOOL
    LD : BOOL
    PV : INT
  VAR_OUTPUT
    QU : BOOL
    QD : BOOL
    CV : INT
```

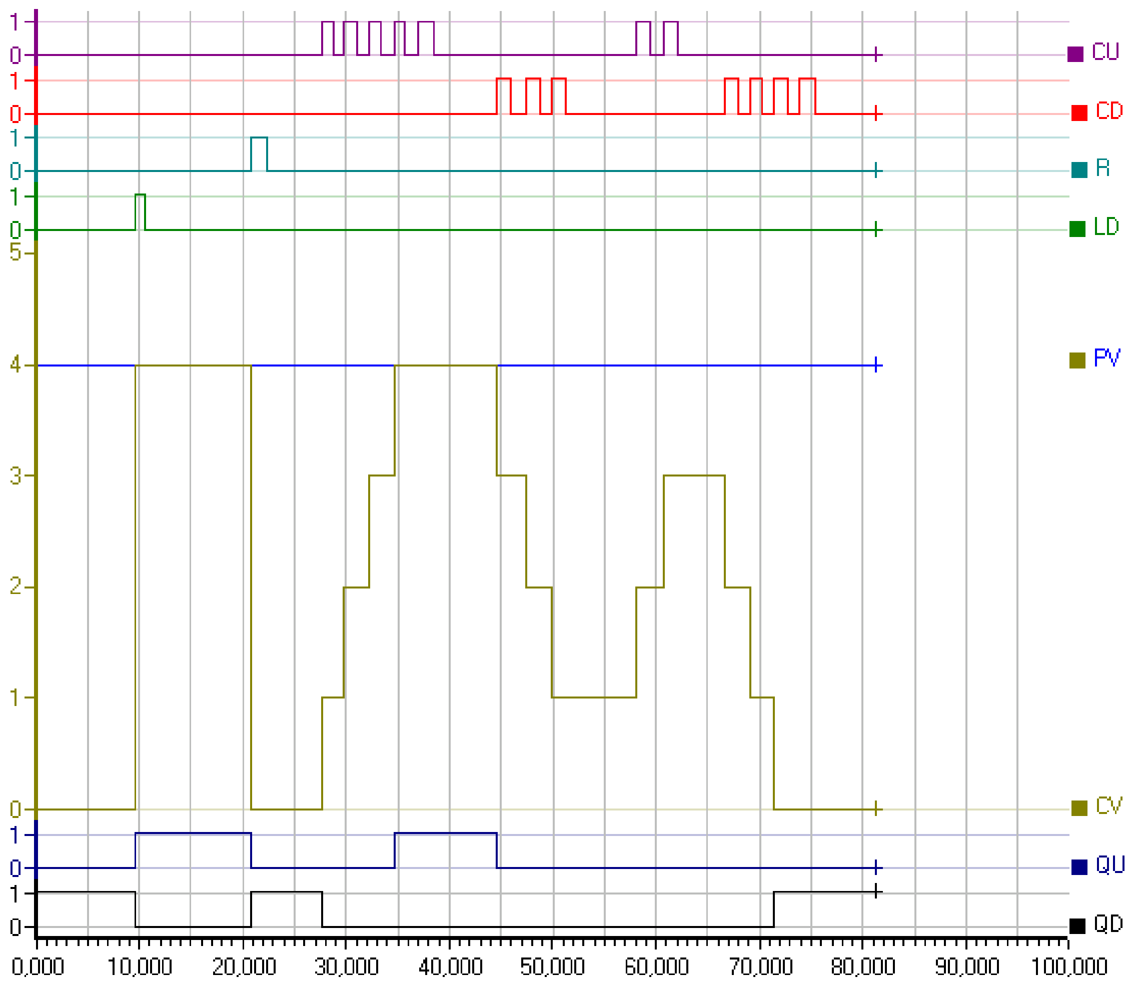
Input variables:

CU	input for count up
CD	input for count down
R	counter reset
LD	input for preset setting
PV	counter preset

Output variables:

QU	output of counter up
QD	output of counter down
CV	counter value

The behavior of CTUD counter is explained on the following figure.



If the value of input variable R is TRUE, the counter value CV is set to zero. If the value of input variable LD is TRUE, then the counter value CV is set to preset value PV.

When the input variable CU changes the status from FALSE to TRUE (rising edge), the counter value CV is increased by 1, the counter counts up. Similarly, if the input variable CD changes the status from FALSE to TRUE (rising edge), the counter status CV is decreased by 1, the counter counts down.

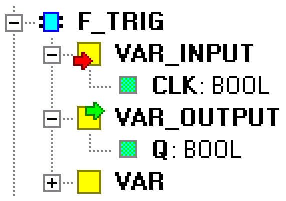
If the counter value CV is greater or equal to preset value PV, the counter output QU is set to TRUE, otherwise it is FALSE. If the counter value CV is equal to 0, the counter output QD is set to TRUE, otherwise it is FALSE.

An example of program where function block CTUD is called:

```
PROGRAM Counter
  VAR
    pulsesUP        : BOOL;
    pulsesDOWN      : BOOL;
    resetCTUD      : BOOL;
    setCTUD        : BOOL;
    counterCTUD    : CTUD;                  // counter instance
    limitUP        : BOOL;
    limitDOWN      : BOOL;
  END_VAR

  counterCTUD( CU := pulsesUP,   CD := pulsesDOWN,
               R  :=  resetCTUD, LD := setCTUD,
               PV := 4,
               QU => limitUP, QD => limitDOWN );
END_PROGRAM
```

2.4 Function block F_TRIG

Function block is used for detection of falling edge of signal.

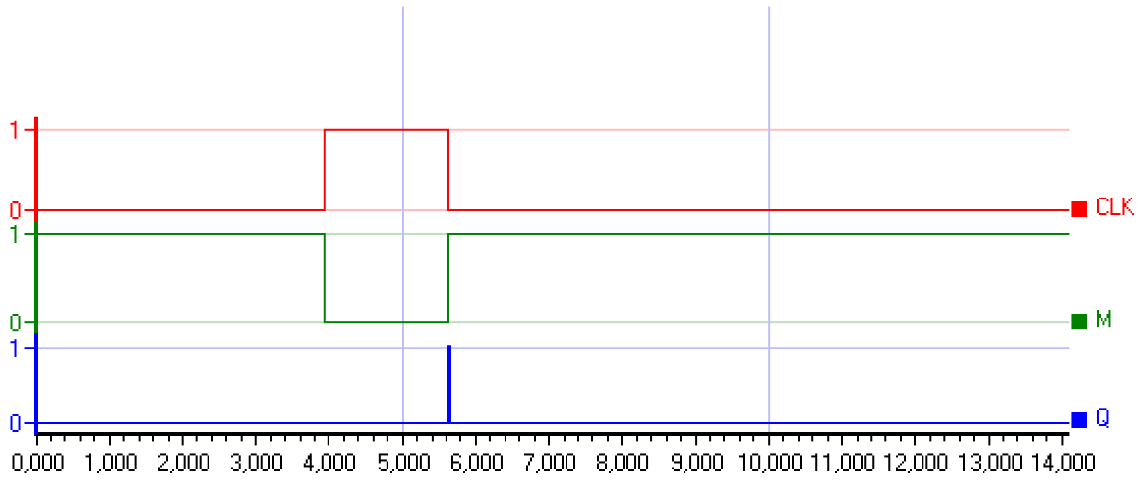

F_TRIG
VAR_INPUT
CLK : BOOL
VAR_OUTPUT
Q : BOOL
VAR

Input variable :
CLK

Output variable :
Q

The behavior of F_TRIG function block responds to the following program in ST language.

```
function_block F_TRIG
//-------------------------------------
//  Falling Edge Detector
//
  var_input  CLK : BOOL;          end_var
  var_output Q   : BOOL;          end_var
  var        M   : BOOL := TRUE; end_var

  Q := not CLK and not M; M := not CLK;
end_function_block
```

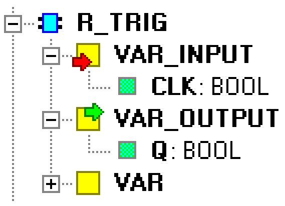


An example of program where function block F_TRIG is called:

```
PROGRAM EdgeDetect
  VAR
    input        : BOOL;
    inst_FTRIG   : F_TRIG;              // instance of FB F_TRIG
    output       : BOOL;
  END_VAR

  inst_FTRIG( CLK := input, Q => output);
END_PROGRAM
```

2.5 *Function block R_TRIG*

Function block is used for detection of falling edge of signal..



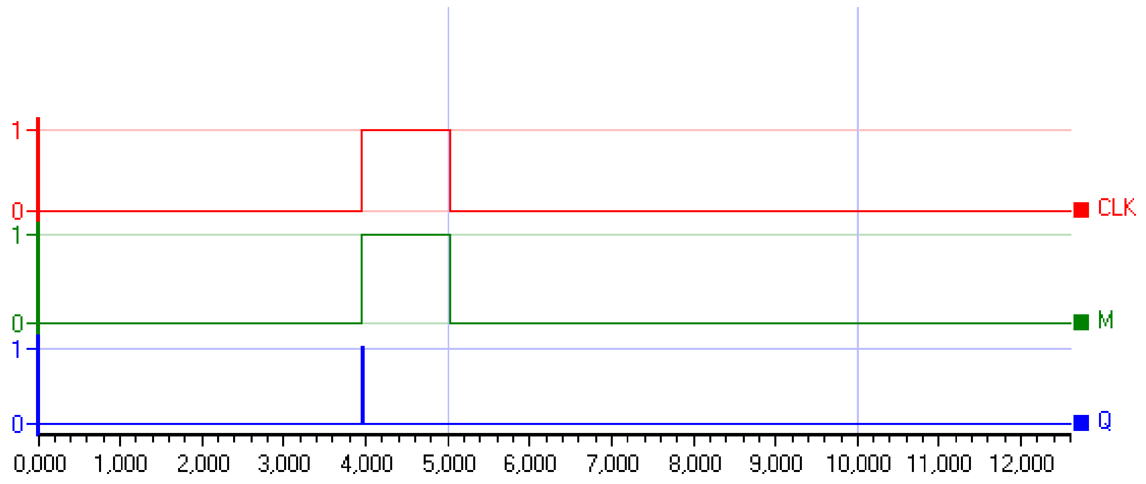
```
R_TRIG
  VAR_INPUT
    CLK : BOOL
  VAR_OUTPUT
    Q : BOOL
  VAR
```

Input variables :
CLK

Output variables:
Q

The behavior of R_TRIG function block responds to the following program in ST language

```
function_block R_TRIG
//-------------------------------
//  Rising Edge Detector
//
  var_input  CLK : BOOL; end_var
  var_output Q   : BOOL; end_var
  var        M   : BOOL; end_var

  Q := clk and not M; M := CLK;
end_function_block
```

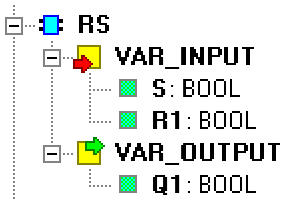


An example of program where function block R_TRIG is called:

```
PROGRAM EdgeDetect
  VAR
    input          : BOOL;
    inst_RTRIG     : R_TRIG;                  // instance of FB R_TRIG
    output         : BOOL;
  END_VAR

  inst_RTRIG( CLK := input, Q => output);
END_PROGRAM
```

2.6 *Function block RS*

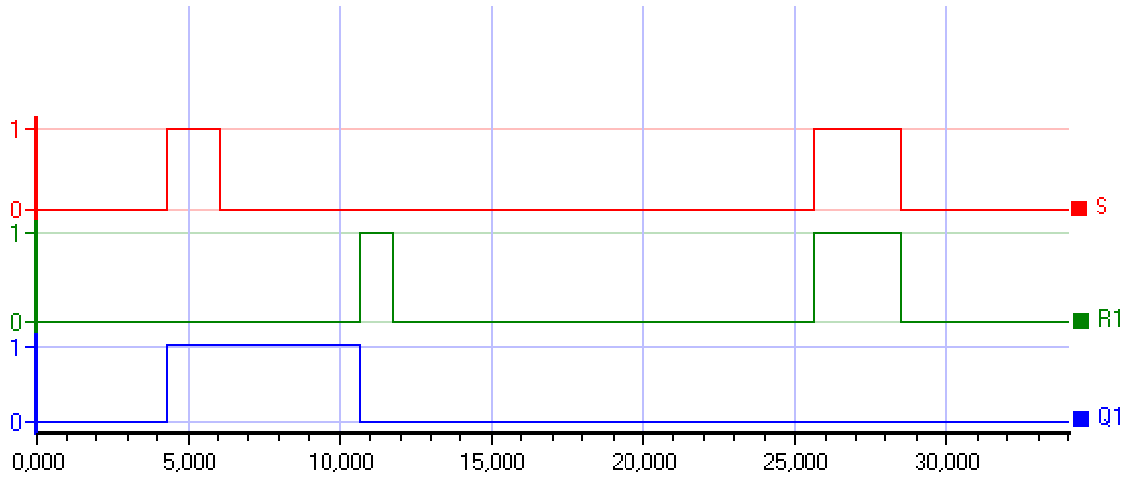
The RS function block is used for implementation of bistable flip-flop circuit with dominant RESET function.



```
RS
  VAR_INPUT
    S: BOOL
    R1: BOOL
  VAR_OUTPUT
    Q1: BOOL
```

The behavior of RS function block responds to the following program in ST language.

```
function_block RS
//-------------------------------------
//  Flip-Flop (Reset Dominant)
//
  var_input   S, R1      : BOOL; end_var
  var_output Q1          : BOOL; end_var

  Q1 := not R1 and (S or Q1);
end_function_block
```

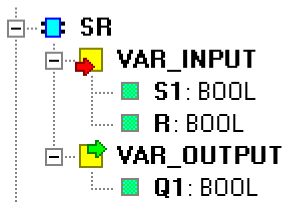


An example of program where function block RS is called:

```
PROGRAM BistableBlock
  VAR
    inputSET       : BOOL;
    inputRESET     : BOOL;
    inst_RS        : RS;
    output         : BOOL;
  END_VAR

  inst_RS( S := inputSET, R1 := inputRESET, Q1 => output);
END_PROGRAM
```

2.7 Function block SR

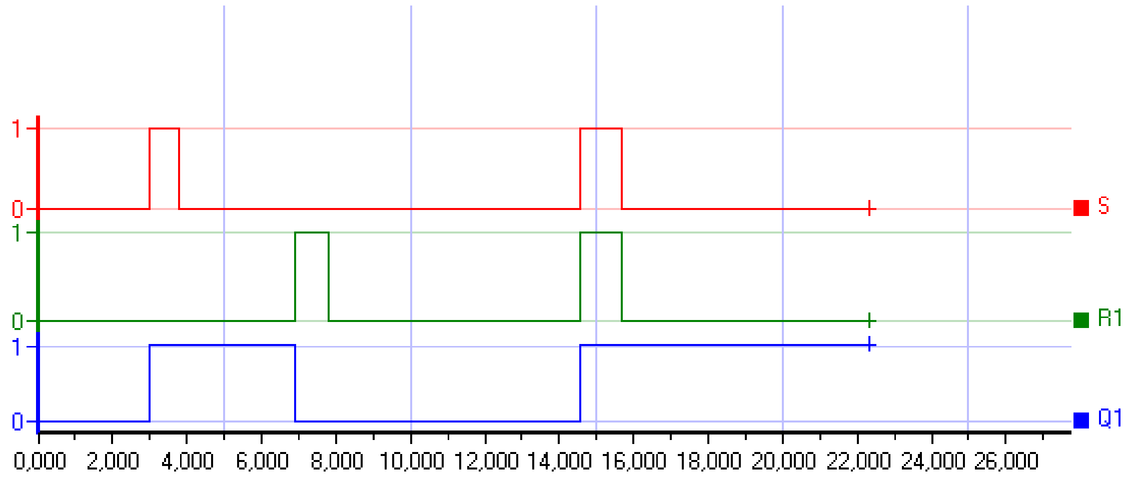
The SR function block is used for implementation of bistable flip-flop circuit with dominant SET function.



```
SR
  VAR_INPUT
    S1 : BOOL
    R : BOOL
  VAR_OUTPUT
    Q1 : BOOL
```

The behavior of SR function block responds to the following program in ST language.

```
function_block SR
//------------------------------------
//  Flip-Flop (Set Dominant)
//
  var_input  S1, R       : BOOL; end_var
  var_output Q1          : BOOL; end_var

  Q1 := S1 or (not R and Q1);
end_function_block
```

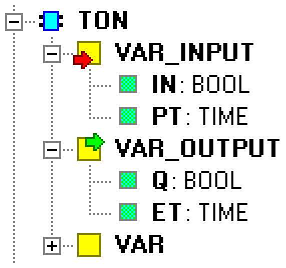


An example of program where function block SR is called:

```
PROGRAM BistableBlock
  VAR
    inputSET       : BOOL;
    inputRESET     : BOOL;
    inst_SR        : SR;
    output         : BOOL;
  END_VAR

  inst_SR( S1 := inputSET, R := inputRESET, Q1 => output);
END_PROGRAM
```

2.8 *Function block of TON timer*

The function block TON (Timer On Delay) realizes a delay at the rising edge, which represents a relay with delayed switching on.



```
TON
  VAR_INPUT
    IN : BOOL
    PT : TIME
  VAR_OUTPUT
    Q : BOOL
    ET : TIME
  VAR
```

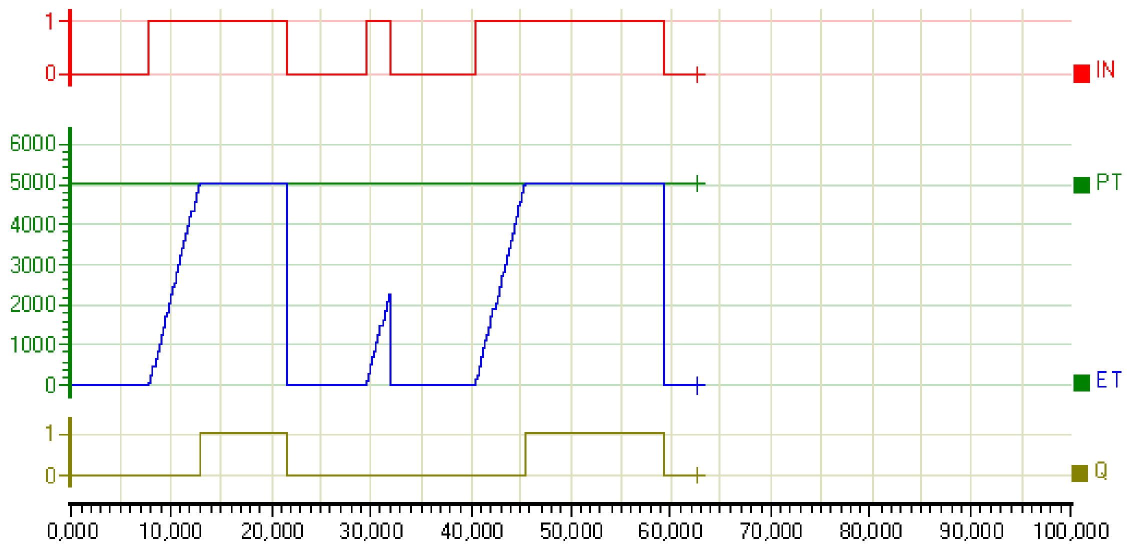
Input variables :

IN	timer input
PT	timer preset

Output variables:

Q	timer output
ET	timer actual value

If the input IN is FALSE, the output Q is set to FALSE and a value of ET is equal to 0. When the input IN changes the status to TRUE, the timer actual value starts to increase and at the time when it reaches the preset value PT, the output Q is set to TRUE. The counter actual value is not increasing next. The behavior of timer TON is explained on the following figure.

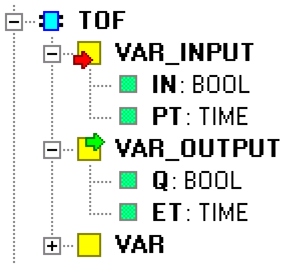


An example of program where the function block TON is called:

```
PROGRAM Timer
  VAR
    start          : BOOL;
    timerTON       : TON;
    output         : BOOL;
  END_VAR

  timerTON( IN := start, PT :=T#5s, Q => output );
END_PROGRAM
```

2.9 *Function block of TOF timer*

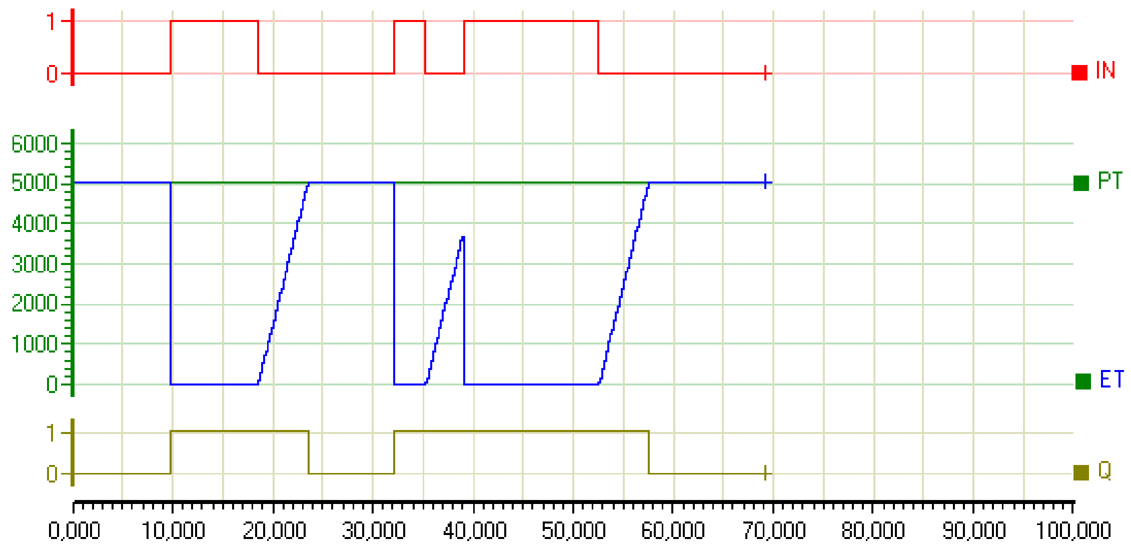
The function block TOF (Timer Off Delay) realizes a delay at the falling edge, which represents a relay with delayed switching off.



```
TOF
  VAR_INPUT
    IN : BOOL
    PT : TIME
  VAR_OUTPUT
    Q : BOOL
    ET : TIME
  VAR
```

Input variables:	
IN	timer input
PT	timer preset
Output variables:	
Q	timer output
ET	timer actual value

When the input IN changes the status to FALSE, the timer actual value starts to increase and at the time when it reaches the preset value PT, the output Q is set to TRUE. The counter actual value is not increasing next. The output Q is set to FALSE when the input IN is FALSE and the actual value of ET is equal to preset value PT at the same time. The behavior of timer TOF is explained on the following figure.

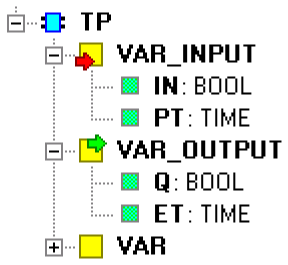


An example of program where the function block TOF is called:

```
PROGRAM Timer
  VAR
    start           : BOOL;
    timerTOF        : TOF;
    output          : BOOL;
  END_VAR

  timerTOF( IN := start, PT :=T#5s, Q => output );
END_PROGRAM
```

2.10 *Function block of TP timer*

The function block TP (Timer Pulse) generates an impulse of defined width at the falling edge.



```
TP
  VAR_INPUT
    IN : BOOL
    PT : TIME
  VAR_OUTPUT
    Q : BOOL
    ET : TIME
  VAR
```

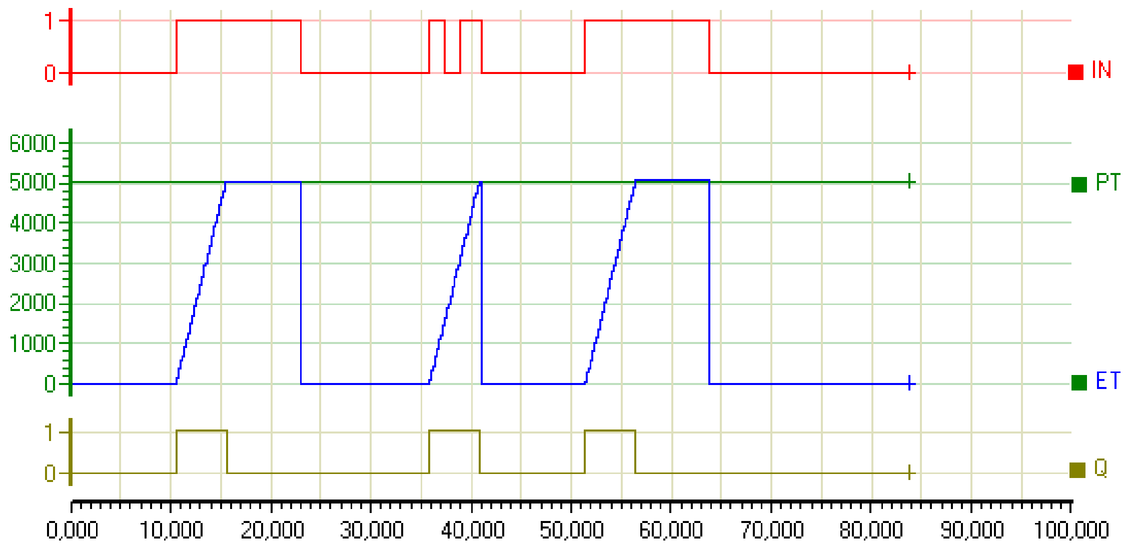
Input variables:

IN	timer input
PT	timer preset

Output variables:

Q	timer output
ET	timer actual value

When the input IN changes the status to TRUE, the timer actual value ET starts to increase until reaching the preset value PT. The timer actual value is not increasing next. The output Q is set to TRUE if the rising edge was detected on the input IN and the actual value ET is less then preset value PT at the same time. Otherwise the output Q is FALSE. The behavior of timer TP is decribed on the following figure.



An example of program where the function block TP is called:

```
PROGRAM Timer
  VAR
    start          : BOOL;
    timerTP        : TP;
    output         : BOOL;
  END_VAR

  timerTP( IN := start, PT :=T#5s, Q => output );
END_PROGRAM
```

2.11 ADD_TIME function

The ADD_TIME function sums up the two input variables of the TIME type. Return value of the function is of the TIME type.

```
PROGRAM Example_ADD_TIME
  VAR
    cas1, cas2, cas3  : TIME;
  END_VAR

  cas1 := TIME#11h12m13s;
  cas2 := ADD_TIME( IN1 := cas1, IN2 := T#1h);  // T#12h12m13.0s
  cas3 := ADD_TIME( T#10m, cas2);               // T#12h22m13.0s
END_PROGRAM
```

2.12 ADD_TOD_TIME function

The ADD_TOD_TIME function sums up the input variable of the TIME_OF_DAY type. Return value of the function is of the TIME_OF_DAY type.

```
PROGRAM Example_ADD_TOD_TIME
  VAR
    cas1    : TIME_OF_DAY := TOD#11:38:52.35;
    cas2    : TIME        := T#15:22:11.120;
    cas3    : TIME_OF_DAY;
  END_VAR

  cas3 := ADD_TOD_TIME( IN1 := cas1, IN2 := cas2);   // 27:01:03.155
END_PROGRAM
```

2.13 ADD_DT_TIME function

The ADD_DT_TIME function sums up the input variable of the DATE_AND_TIME type. Return value of the function is of the DATE_AND_TIME type.

```
PROGRAM Example_ADD_DT_TIME
  VAR
    varDT        : DATE_AND_TIME := DT#2004-05-30-00:00:00;
    varTIME      : TIME          := TIME#12:55:02.0;
    sum          : DATE_AND_TIME;
  END_VAR

  sum := ADD_DT_TIME(IN1 := varDT, IN2 := varTIME);
                                                 //DT#2004-05-30-12:55:02
END_PROGRAM
```

2.14 SUB_TIME function

The SUB_TIME function subtracts the two input variables of the TIME type. Return value of the function is of the DATE_AND_TIME type.

```
PROGRAM Example_SUB_TIME
  VAR
    cas1, cas2, cas3   : TIME;
  END_VAR

  cas1 := TIME#11h12m13s;
  cas2 := SUB_TIME( IN1 := cas1, IN2 := T#1h);  // T#10h12m13.0s
  cas3 := SUB_TIME( cas2, T#10m);               // T#10h02m13.0s
END_PROGRAM
```

2.15 SUB_DATE_DATE function

The SUB_DATE_DATE function subtracts the two input variables of the DATE type. Return value of the function is of the TIME type.

```
PROGRAM Example_SUB_DATE_DATE
  VAR
    varDate1      : DATE := D#2005-11-03;
    varDate2      : DATE := D#2005-10-21;
    varTime1      : TIME;
    varTime2      : TIME;
    varTime3      : TIME;
    maxTime       : TIME := T#24d20h31m23.647s;
    timeOverFlow  : BOOL;
    timeUnderFlow : BOOL;
  END_VAR

  varTime1 := SUB_DATE_DATE( IN1 := varDate1, IN2 := varDate2);
              // T#13d00h00m00s

  varTime2 := SUB_DATE_DATE( D#2005-11-03, D#2005-05-21);
              // T#24d20h31m23.647s

  IF varTime2 = maxTime THEN
    timeOverFlow := TRUE;
  ELSE
    timeOverFlow := FALSE;
  END_IF;

  varTime3 := SUB_DATE_DATE(IN1 := varDate2, IN2 := D#2005-10-22);
  IF varTime3 < T#0s THEN
    timeUnderFlow := TRUE;
  ELSE
    timeUnderFlow := FALSE;
  END_IF;

END_PROGRAM
```

2.16 SUB_TOD_TIME function

The SUB_TOD_TIME function subtracts the input variable of the TIME type from the TIME_OF_DAY variable type. Return value of the function is of the TIME_OF_DAY type.

```
PROGRAM Example_SUB_TOD_TIME
  VAR
    varTOD       : TIME_OF_DAY := TOD#19:22:33;
    varTIME      : TIME        := TIME#12:00:00;
    result1      : TIME_OF_DAY;
  END_VAR

  IF varTOD >= TIME_TO_TIME_OF_DAY( varTIME) THEN
    result1 := SUB_TOD_TIME(IN1 := varTOD, IN2 := varTIME); //TOD#07:22:33
  ELSE
    result1 := TOD#00:00:00;
  END_IF;

END_PROGRAM
```

2.17 SUB_TOD_TOD function

The SUB_TOD_TOD function subtracts the input variable of the TIME_OF_DAY type from the TIME_OF_DAY variable type. Return value of the function is of the TIME type.

```
PROGRAM Example_SUB_TOD_TOD
  VAR
    varTOD1      : TIME_OF_DAY := TOD#19:22:33;
    varTOD2      : TIME_OF_DAY := TOD#10:12:13;
    varTime      : TIME;
  END_VAR

  varTime := SUB_TOD_TOD( varTOD1, varTOD2); // T#09h10m20.0s
END_PROGRAM
```

2.18 SUB_DT_TIME function

The SUB_DT_TIME function subtracts the input variable of the TIME type from the DATE_AND_TIME variable type. Return value of the function is of the DATE_AND_TIME type.

```
PROGRAM Example_SUB_DT_TIME

  VAR
    varDateTime : DT    := DT#2005-12-05-06:00:00.0;
    varTime     : TIME  := T#12h;
    result      : DATE_AND_TIME;
    i  : INT  := 20;
  END_VAR

  result := SUB_DT_TIME(IN1 := varDateTime, IN2 := varTime);
            // DT#2005-12-04-18:00:00.0
END_PROGRAM
```

2.19 SUB_DT_DT function

The SUB_DT_DT function subtracts the input variable of the DATE_AND_TIME type from the DATE_AND_TIME variable type. Return value of the function is of the TIME type.

```
PROGRAM Example_SUB_DT_DT
  VAR
    varDT1      : DT  := DT#2005-06-12-13:00:30.0;
    varDT2      : DT  := DT#2005-06-11-12:00:15.0;
    varTIME     : TIME;
  END_VAR

  varTIME := SUB_DT_DT(IN1 := varDT1, IN2 := varDT2);
             // T#1d01h00m15.0s
END_PROGRAM
```

2.20 CONCAT_DATE_TOD function

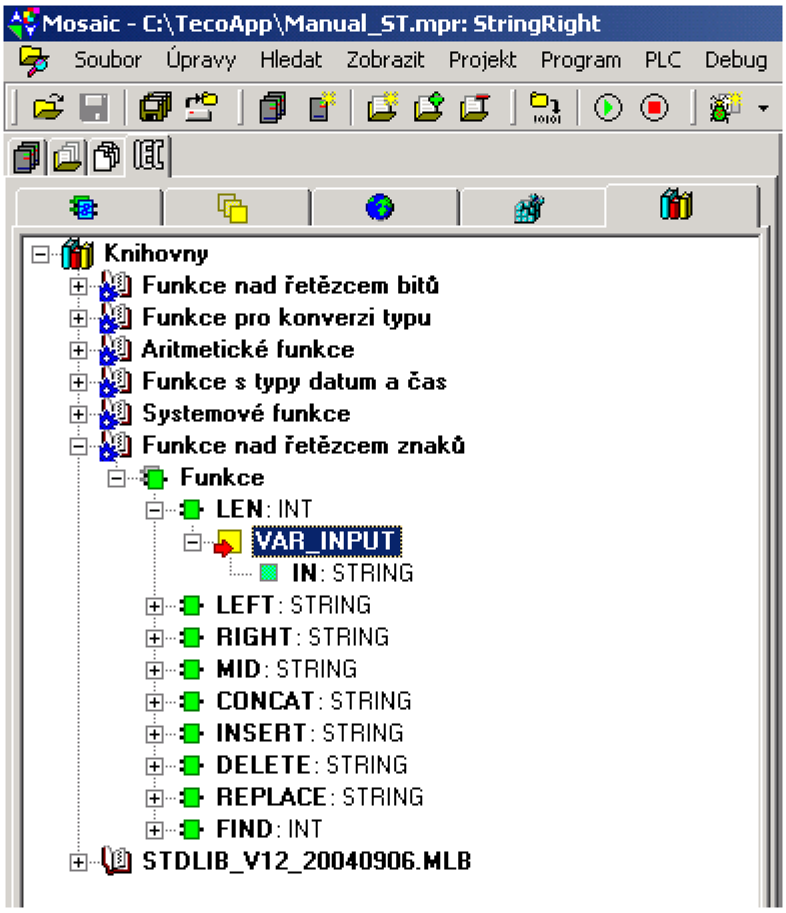
The CONCAT_DATE_TOD function sums up the input variable of the DATE type from the TIME_OF_DAY variable type. Return value of the function is of the DATE_AND_TIME type.

```
PROGRAM Example_CONCAT_DATE_TOD
  VAR
    varDate : DATE        := D#2005-02-28;
    varTOD  : TIME_OF_DAY := TOD#08:20:40.0;
    varDT   : DATE_AND_TIME;
  END_VAR

  varDT := CONCAT_DATE_TOD( varDate, varTOD);  // DT#2005-02-28-08:20:40.0
END_PROGRAM
```

3 FUNCTION ABOVE CHARACTER STRING

This library contains functions for work with character strings (**`STRING`** `data type`). It contains a function for determining the string length LEN, a function for selecting string parts LEFT, RIGHT and MID, a function for connecting strings CONCAT, a function for inserting strings into strings INSERT, a function for deleting a string part DELETE, a function for replacing a string parte REPLACE and finally a function for finding a string part within another string FIND. The following figure shows the implementation of functions above character string into the Mosaic environment library.

Implementation of the STRING data type in TECOMAT systems matches C language strings.



```
Knihovny
  Funkce nad řetězcem bitů
  Funkce pro konverzi typu
  Aritmetické funkce
  Funkce s typy datum a čas
  Systemové funkce
  Funkce nad řetězcem znaků
    Funkce
      LEN : INT
        VAR_INPUT
          IN : STRING
      LEFT : STRING
      RIGHT : STRING
      MID : STRING
      CONCAT : STRING
      INSERT : STRING
      DELETE : STRING
      REPLACE : STRING
      FIND : INT
  STDLIB_V12_20040906.MLB
```

3.1 LEN Function

The function will return the length of the input string IN. The length of the string matches the current number of characters in the string. The return value is of the INT type.

LEN : INT
VAR_INPUT
IN : STRING

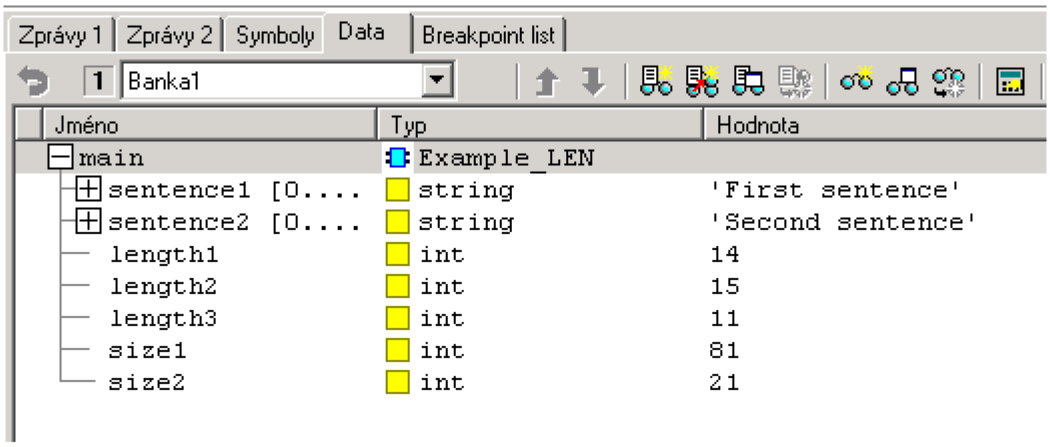
Input variables :

IN	character string

Return value : string length

The use of the LEN function is shown on the following example. In connection with the string length, it is good to relize, that the real string length, in most cases, does not coresspond to the size of the variable in which the string is located. The size of the STRING variable type is given by variable declaration and can be determined using the SIZEOF function while the size of the string in the variable is given by the number of ASCII characters in the string and is determined using the LEN function.

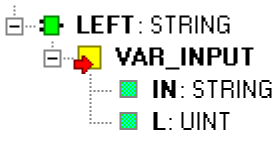
```
PROGRAM Example_LEN
  VAR
    sentence1      : STRING      := 'First sentence';
    sentence2      : STRING[20]  := 'Second sentence';
    length1,
    length2,
    length3 : INT;
    size1,
    size2   : INT;
  END_VAR

  // length of string
  length1 := LEN( sentence1);
  length2 := LEN( IN := sentence2);
  length3 := LEN( 'Hello world');

  // size of variable
  size1 := sizeof( sentence1);
  size2 := sizeof( sentence2);

END_PROGRAM
```



Jméno	Typ	Hodnota
main	Example_LEN	
sentence1 [0....	string	'First sentence'
sentence2 [0....	string	'Second sentence'
length1	int	14
length2	int	15
length3	int	11
size1	int	81
size2	int	21

3.2 LEFT function

The LEFT function returns L characters from the input string IN. Characters are returned from the left, i.e. from the beginning of the string.



```
LEFT : STRING
  VAR_INPUT
    IN : STRING
    L : UINT
```

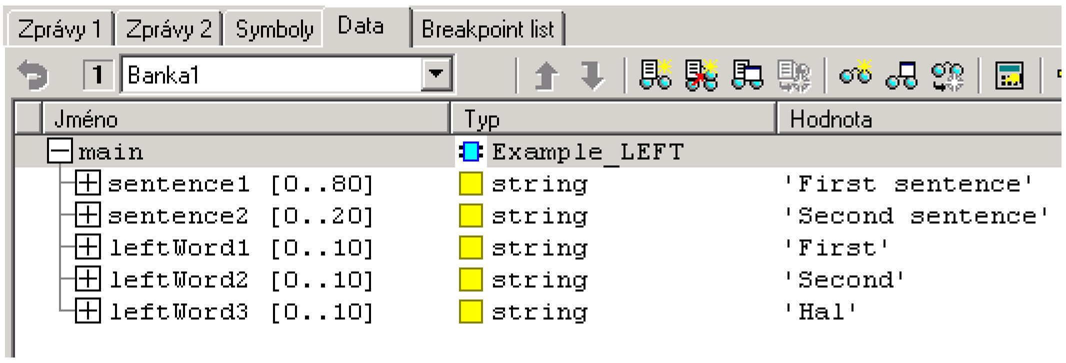
Input variables :

IN	input string
L	number of characters

Return value : string

The next example shows the use of the LEFT function.

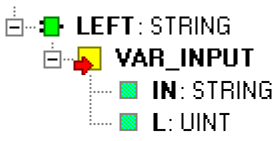
```
PROGRAM Example_LEFT
  VAR
    sentence1     : STRING     := 'First sentence';
    sentence2     : STRING[20] := 'Second sentence';
    leftWord1,
    leftWord2,
    leftWord3     : STRING[10];
  END_VAR

  leftWord1 := LEFT( sentence1, 5);
  leftWord2 := LEFT( IN := sentence2, L := 6);
  leftWord3 := LEFT( 'Hallo world', 3);

END_PROGRAM
```



Jméno	Typ	Hodnota
main	Example_LEFT	
sentence1 [0..80]	string	'First sentence'
sentence2 [0..20]	string	'Second sentence'
leftWord1 [0..10]	string	'First'
leftWord2 [0..10]	string	'Second'
leftWord3 [0..10]	string	'Hal'

3.3 RIGHT function

The RIGHT function returns L characters from the input string IN. Characters are returned from the right, i.e. from the end of the string.

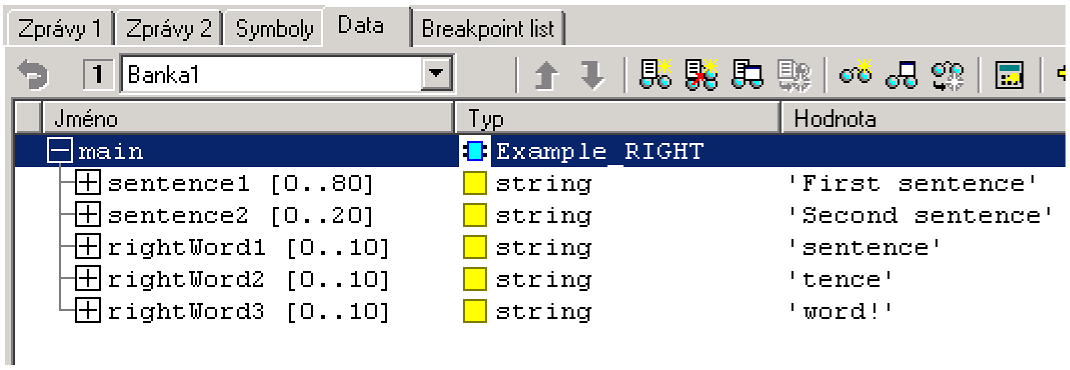


```
LEFT : STRING
  VAR_INPUT
    IN : STRING
    L : UINT
```

Input variables :	
IN	input string
L	number of characters
Return value : string	

The next example shows the use of the RIGHT function.

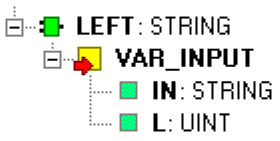
```
PROGRAM Example_RIGHT
  VAR
    sentence1      : STRING      := 'First sentence';
    sentence2      : STRING[20]  := 'Second sentence';
    rightWord1,
    rightWord2,
    rightWord3     : STRING[10];
  END_VAR

  rightWord1 := RIGHT( sentence1, 8);
  rightWord2 := RIGHT( IN := sentence2, L := 5);
  rightWord3 := RIGHT( 'Hello word!', 5);

END_PROGRAM
```



Jméno	Typ	Hodnota
main	Example_RIGHT	
sentence1 [0..80]	string	'First sentence'
sentence2 [0..20]	string	'Second sentence'
rightWord1 [0..10]	string	'sentence'
rightWord2 [0..10]	string	'tence'
rightWord3 [0..10]	string	'word!'

3.4 MID function

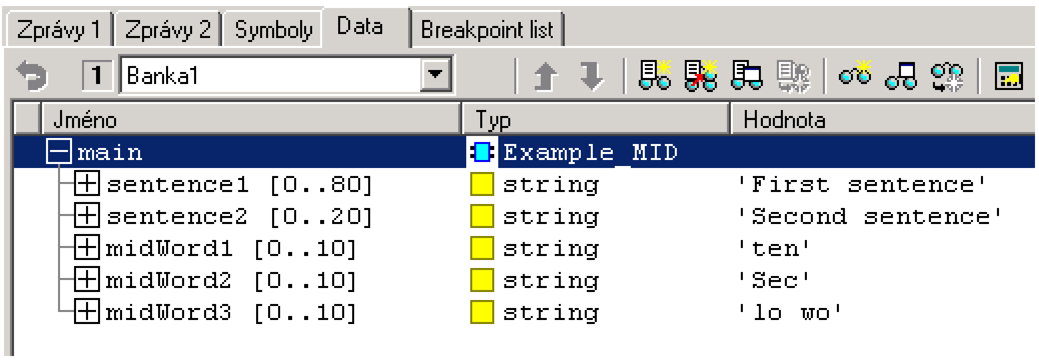
The MID function returns L characters from the input string IN. Characters are returned from the position P in the input string. The first string character has the position 1.



Input variables :	
IN	input string
L	number of characters
P	position in input string
Return value : string	

The next example shows the use of the MID function.

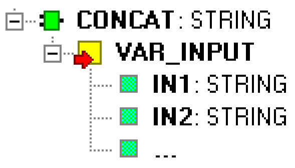
```
PROGRAM Example_MID
  VAR
    sentence1      : STRING := 'First sentence';
    sentence2      : STRING[20];
    midWord1,
    midWord2,
    midWord3       : STRING[10];
  END_VAR

  midWord1  := mid( sentence1, 3, 10);
  sentence2 := 'Second sentence';
  midWord2  := mid( IN := sentence2, L := 3, P := 1);
  midWord3  := mid( 'Hello world', 5, 4);

END_PROGRAM
```



Jméno	Typ	Hodnota
main	Example_MID	
sentence1 [0..80]	string	'First sentence'
sentence2 [0..20]	string	'Second sentence'
midWord1 [0..10]	string	'ten'
midWord2 [0..10]	string	'Sec'
midWord3 [0..10]	string	'lo wo'

3.5 CONCAT function

The CONCAT function merges several strings into the input string.



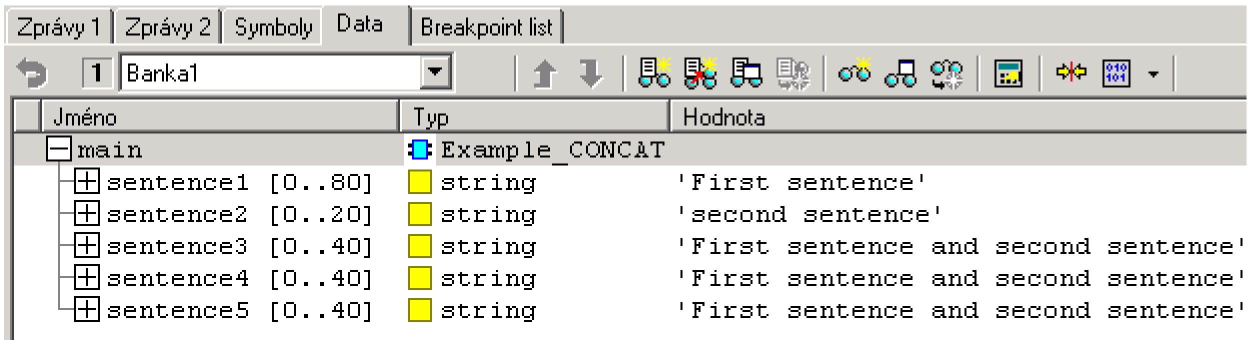
Input variables :

IN1	character string
IN2	character string

Return value : string

The use of the CONCAT function is shown on the following example. The function can be called in a classic manner using its name or using an overload operator "**+**". The CONCAT function is extendible, so it can have a various number of input strings.

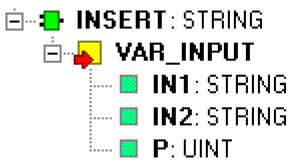
```
PROGRAM Example_CONCAT
  VAR
    sentence1      : STRING      := 'First sentence';
    sentence2      : STRING[20]  := 'second sentence';
    sentence3,
    sentence4,
    sentence5      : STRING[40];
  END_VAR

  sentence3 := CONCAT( sentence1, ' and ', sentence2);
  sentence4 := CONCAT( IN1 := sentence1, IN2 := ' and ', IN3 := sentence2);
  sentence5 := sentence1 + ' and ' + sentence2;
END_PROGRAM
```



```
Jméno                    Typ              Hodnota
main                     Example_CONCAT
  sentence1 [0..80]      string           'First sentence'
  sentence2 [0..20]      string           'second sentence'
  sentence3 [0..40]      string           'First sentence and second sentence'
  sentence4 [0..40]      string           'First sentence and second sentence'
  sentence5 [0..40]      string           'First sentence and second sentence'
```

3.6 INSERT function

The INSERT function inputs the IN2 string into the IN1 string into the P position. Such a merged string is returned as the function return value.


```
INSERT : STRING
  VAR_INPUT
    IN1 : STRING
    IN2 : STRING
    P : UINT
```

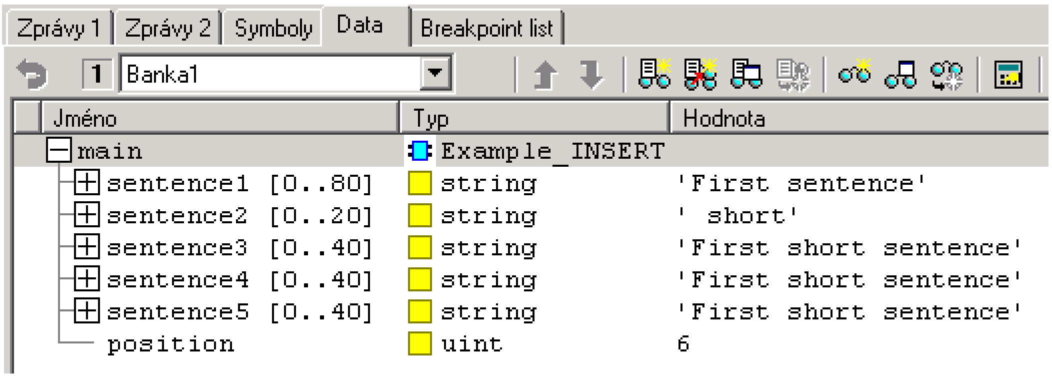
Input variables :

IN1	input string
IN2	string being input
P	position in input string

Return value : string

The next example shows the use of the INSERT function.

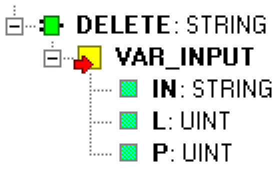
```
PROGRAM Example_INSERT
  VAR
    sentence1      : STRING      := 'First sentence';
    sentence2      : STRING[20]  := ' short';
    sentence3,
    sentence4,
    sentence5      : STRING[40];
    position       : UINT  := 6;
  END_VAR

  sentence3 := INSERT( sentence1, ' short', 6);
  sentence4 := INSERT( IN1 := sentence1, IN2 := ' short', P := 6);
  sentence5 := INSERT( sentence1, sentence2, position);
END_PROGRAM
```



```
Jméno                     Typ              Hodnota
main                      Example_INSERT
  sentence1 [0..80]       string           'First sentence'
  sentence2 [0..20]       string           ' short'
  sentence3 [0..40]       string           'First short sentence'
  sentence4 [0..40]       string           'First short sentence'
  sentence5 [0..40]       string           'First short sentence'
  position                uint             6
```

3.7 DELETE function

The DELETE function deletes a L number of characters from the IN string. The characters are deleted starting from the P position. Such a created string is returned as the function return value.



```
DELETE : STRING
  VAR_INPUT
    IN : STRING
    L : UINT
    P : UINT
```

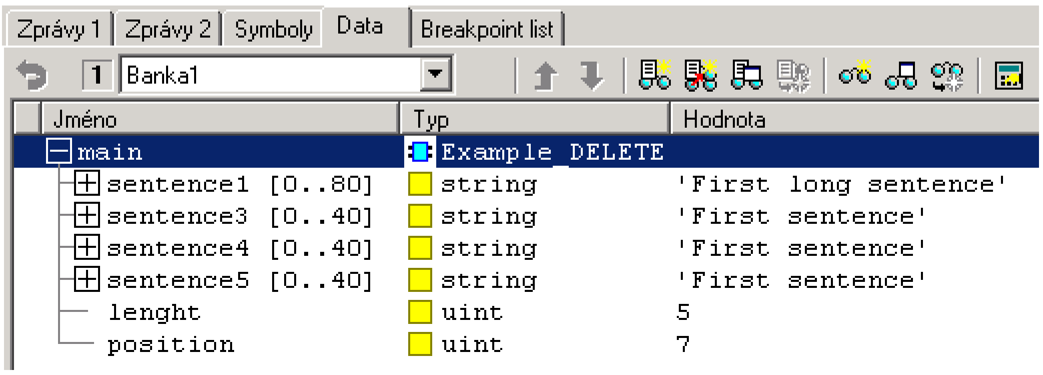
Input variables :

IN	input string
L	number of deleted characters
P	position from which the strings are deleted

Return value : string

The next example shows the use of the DELETE function.

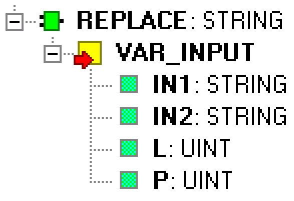
```
PROGRAM Example_DELETE
  VAR
    sentence1      : STRING     := 'First long sentence';
    sentence3,
    sentence4,
    sentence5      : STRING[40];
    lenght         : UINT  := 5;
    position       : UINT  := 7;
  END_VAR

  sentence3 := DELETE( sentence1, 5, 7);
  sentence4 := DELETE( IN := sentence1, L := lenght, P := 7);
  sentence5 := DELETE( sentence1, lenght, position);
END_PROGRAM
```



Jméno	Typ	Hodnota
main	Example_DELETE	
sentence1 [0..80]	string	'First long sentence'
sentence3 [0..40]	string	'First sentence'
sentence4 [0..40]	string	'First sentence'
sentence5 [0..40]	string	'First sentence'
lenght	uint	5
position	uint	7

3.8 REPLACE function

The REPLACE function deletes a L number of characters from the input IN string. The characters are deleted starting from the P position. Then a IN2 string is inserted into the same P position. Such a created string is returned as the function return value.



```
REPLACE : STRING
  VAR_INPUT
    IN1 : STRING
    IN2 : STRING
    L : UINT
    P : UINT
```

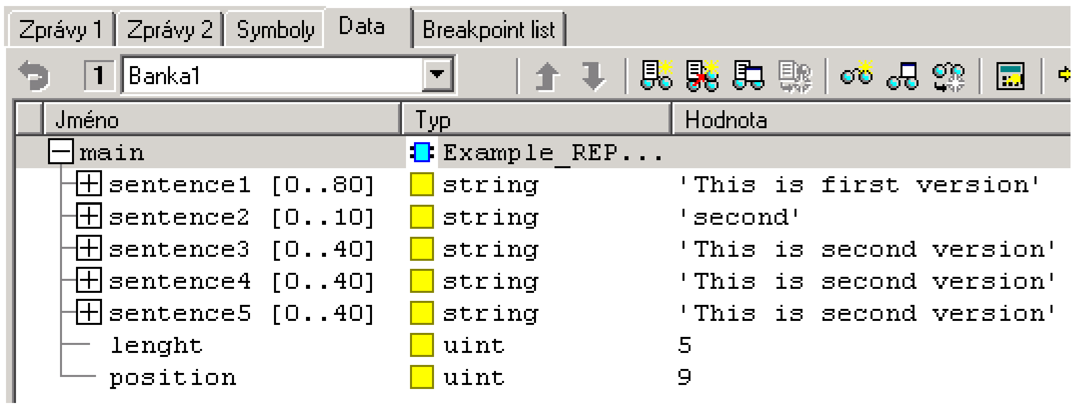
Input variables :

IN1	input string
IN2	string being input
L	number of deleted characters
P	position in input string

Return value : string

The next example shows the use of the REPLACE function.

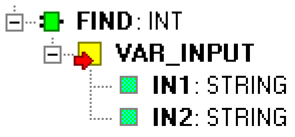
```
PROGRAM Example_REPLACE
  VAR
    sentence1      : STRING     := 'This is first version';
    sentence2      : STRING[10] := 'second';
    sentence3,
    sentence4,
    sentence5      : STRING[40];
    lenght         : UINT  := 5;
    position       : UINT  := 9;
  END_VAR

  sentence3 := REPLACE( sentence1, 'second', 5, 9);
  sentence4 := REPLACE( IN1 := sentence1, IN2 := sentence2, L := 5, P := 9);
  sentence5 := REPLACE( sentence1, sentence2, lenght, position);
END_PROGRAM
```



```
Zprávy 1 | Zprávy 2 | Symboly | Data | Breakpoint list
1 Banka1
Jméno                     Typ              Hodnota
main                      Example_REP...
  sentence1 [0..80]       string           'This is first version'
  sentence2 [0..10]       string           'second'
  sentence3 [0..40]       string           'This is second version'
  sentence4 [0..40]       string           'This is second version'
  sentence5 [0..40]       string           'This is second version'
  lenght                  uint             5
  position                uint             9
```

3.9 FIND function

The FIND function returns the IN2 string position in the IN1 input string. If the IN2 string is not contained in the IN1 string, the function returns a 0. If the IN2 string is contained in the IN1 string multiple times, the FIND function returns the position of the IN2 string's first position. The search differentiates between lowercase and capital letters. The function return value is of the INT type.



```
FIND : INT
  VAR_INPUT
    IN1 : STRING
    IN2 : STRING
```

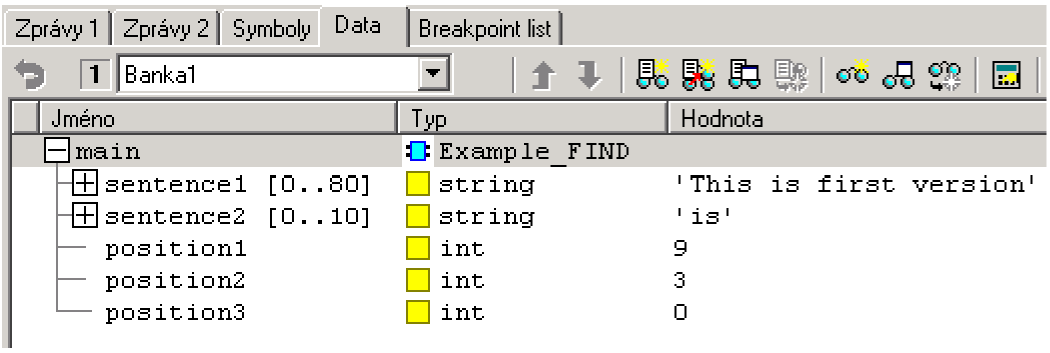
Input variables :

IN1	input string
IN2	searched for string

Return value : position IN2 in string IN1

The next example shows the use of the FIND function.

```
PROGRAM Example_FIND
  VAR
    sentence1      : STRING     := 'This is first version';
    sentence2      : STRING[10] := 'is';
    position1,
    position2,
    position3      : INT;
  END_VAR

  position1 := FIND( sentence1, 'first');
  position2 := FIND( IN1 := sentence1, IN2 := sentence2);
  position3 := FIND( sentence1, 'First');
END_PROGRAM
```



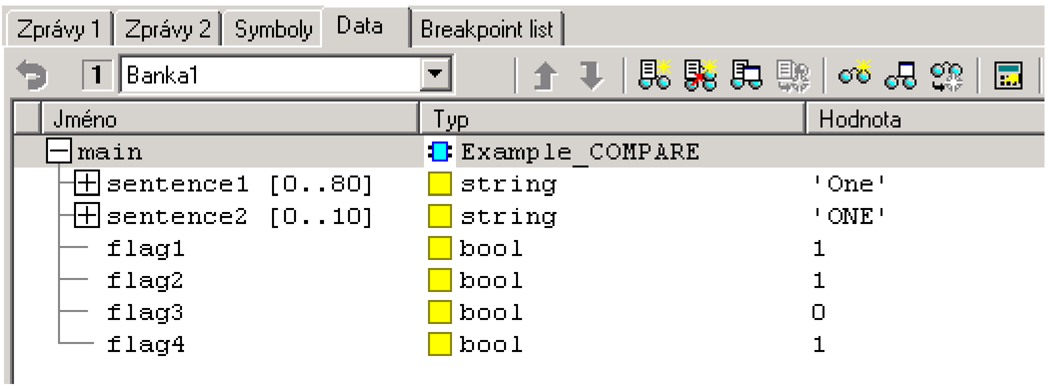
Jméno	Typ	Hodnota
main	Example_FIND	
sentence1 [0..80]	string	'This is first version'
sentence2 [0..10]	string	'is'
position1	int	9
position2	int	3
position3	int	0

3.10 String compare function

Standard function for comparing (larger, smaller, larger or equel,…) are overloaded even for the STRING type, i.e. strings may be compared between each other. Strings are compared character by character, lowercase and capital letters are differentiated. From the point of view of comparing, the lowercase letters have a “greater value” because the ASCII code of lowercase letters is greater than of capital letters.

The next example shows the use of the function for comparing strings.

```
PROGRAM Example_COMPARE
  VAR
    sentence1      : STRING      := 'One';
    sentence2      : STRING[10]  := 'ONE';
    flag1,
    flag2,
    flag3,
    flag4    : BOOL;
  END_VAR

  flag1 := sentence1 = 'One';
  flag2 := sentence1 <> sentence2;
  flag3 := sentence1 = sentence2;
  flag4 := sentence1 > sentence2;
END_PROGRAM
```



Jméno	Typ	Hodnota
main	Example_COMPARE	
sentence1 [0..80]	string	'One'
sentence2 [0..10]	string	'ONE'
flag1	bool	1
flag2	bool	1
flag3	bool	0
flag4	bool	1

4 TYPE CONVERSION FUNCTION

These functions convert values between elementary and data types.

4.1 ANY_TO_BOOL function

Conversion of a random variable of the elementary type into a BOOL type.

The following function belong into this group of functions:
SINT_TO_BOOL, INT_TO_BOOL, DINT_TO_BOOL
USINT_TO_BOOL, UINT_TO_BOOL, UDINT_TO_BOOL
REAL_TO_BOOL, LREAL_TO_BOOL
STRING_TO_BOOL
TIME_TO_BOOL, TOD_TO_BOOL, DATE_TO_BOOL, DT_TO_BOOL

The result of the conversion …_TO_BOOL is the value TRUE, if the input value of the function is a non-zero value. If a zero value is the input, the result of the conversion is the value FALSE.
In case of the STRING_TO_BOOL function, the result is a TRUE value, if the string input is “true” (size of letters is not important). In other cases the result of the conversion is the value FALSE.

Use of the …_TO_BOOL conversion is shown in the following example:

```
FUNCTION_BLOCK Example_ANY_TO_BOOL
  VAR_OUTPUT
    varBool      : ARRAY[1..20] OF BOOL;
  END_VAR

  varBool[1]   :=   SINT_TO_BOOL( 2);                     // 1
  varBool[2]   :=    INT_TO_BOOL( 0);                     // 0
  varBool[3]   :=   DINT_TO_BOOL( 1_235_678);             // 1
  varBool[4]   :=  USINT_TO_BOOL( 255);                   // 1
  varBool[5]   :=   UINT_TO_BOOL( UINT#16#FFFF);          // 1
  varBool[6]   :=  UDINT_TO_BOOL( 0);                     // 0
  varBool[7]   :=   REAL_TO_BOOL( -12.6);                 // 1
  varBool[8]   :=  LREAL_TO_BOOL( 123.4567);              // 1
  varBool[9]   :=   TIME_TO_BOOL( T#0:00:00.00);          // 0
  varBool[10]  :=   BYTE_TO_BOOL( BYTE#16#AF);            // 1
  varBool[11]  :=   WORD_TO_BOOL( 1122);                  // 1
  varBool[12]  :=  DWORD_TO_BOOL( 12345678);              // 1
  varBool[13]  := STRING_TO_BOOL( 'FALSE');               // 0
  varBool[14]  := STRING_TO_BOOL( 'True');                // 1
  varBool[15]  := STRING_TO_BOOL( '0');                   // 0
  varBool[16]  := STRING_TO_BOOL( '1');                   // 1
  varBool[17]  := STRING_TO_BOOL( 'Anything');            // 0

END_FUNCTION_BLOCK
```

4.2 *ANY_TO_SINT function*

Conversion of a random variable of the elementary type into SINT.

The following function belong into this group of functions:
BOOL_TO_SINT
INT_TO_SINT, DINT_TO_SINT
USINT_TO_SINT, UINT_TO_SINT, UDINT_TO_SINT
REAL_TO_SINT, LREAL_TO_SINT
STRING_TO_SINT
TIME_TO_SINT, TOD_TO_SINT, DATE_TO_SINT, DT_TO_SINT

The result of the conversion …_TO_SINT is a number in within the range <-128, +127>. If we convert from a bigger data type into a smaller one (e.g. DINT_TO_SINT) we are risking loss of information should the input value not be within the stated range

When converting the REAL and LREAL data types the input value is firstly rounded off to the nearest whole number and then the conversion is undertaken.

Use of the …_TO_SINT conversion is shown in the following example:

```
FUNCTION_BLOCK Example_ANY_TO_SINT
  VAR_OUTPUT
    varSint      : ARRAY[1..15] OF SINT;
  END_VAR

  varSint[1]  :=   BOOL_TO_SINT( BOOL#0);             // 0
  varSint[2]  :=   BOOL_TO_SINT( true);               // 1
  varSint[3]  := STRING_TO_SINT( '+127');             // 127
  varSint[4]  :=    INT_TO_SINT( INT#-128);           // -128
  varSint[5]  :=   DINT_TO_SINT( 1_235_678);          // -34
  varSint[6]  :=  USINT_TO_SINT( 255);                // -1
  varSint[7]  :=   UINT_TO_SINT( UINT#16#FFFF);       // -1
  varSint[8]  :=  UDINT_TO_SINT( 16#FFFF_FFFE);       // -2
  varSint[9]  :=   REAL_TO_SINT( -12.6);              // -13
  varSint[10] :=  LREAL_TO_SINT( 123.4567);           // 123
  varSint[11] :=   TIME_TO_SINT( T#0:00:00.100);      // 100
  varSint[12] :=   BYTE_TO_SINT( BYTE#16#AF);         // -81
  varSint[13] :=   WORD_TO_SINT( 1122);               //
  varSint[14] :=  DWORD_TO_SINT( 12345678);           // 78

END_FUNCTION_BLOCK
```

4.3 *ANY_TO_INT function*

Conversion of a random variable of the elementary type into **INT**.

The following function belong into this group of functions:
BOOL_TO_INT
SINT_TO_INT, DINT_TO_INT
USINT_TO_INT, UINT_TO_INT, UDINT_TO_INT
REAL_TO_INT, LREAL_TO_INT
STRING_TO_INT
TIME_TO_INT, TOD_TO_INT, DATE_TO_INT, DT_TO_INT

The result of the conversion …_TO_INT is a number in within the range <–32 768, +32 767>. If we convert from a bigger data type into a smaller one (e.g. DINT_TO_SINT) we are risking loss of information should the input value not be within the stated range.

When converting the REAL and LREAL data types the input value is firstly rounded off to the nearest whole number and then the conversion is undertaken.

Use of the …_TO_INT conversion is shown in the following example:

```
FUNCTION_BLOCK Example_ANY_TO_INT
  VAR_OUTPUT
    varInt     : ARRAY[1..10] OF INT;
  END_VAR

  varInt[1]  :=   BOOL_TO_INT( true);           // 1
  varInt[2]  :=   SINT_TO_INT( 99);             // 99
  varInt[3]  :=   REAL_TO_INT( 123.5678);       // 124
  varInt[4]  :=  LREAL_TO_INT( LREAL#5.21E3);   // 5210
  varInt[5]  := STRING_TO_INT( '-12.5');        // -12
  varInt[6]  := STRING_TO_INT( '4.47E+06');     // 4
  varInt[7]  :=   TIME_TO_INT( T#12s25ms);      // 12025
  varInt[8]  :=   BYTE_TO_INT( BYTE#16#AF);     // 175
  varInt[9]  :=   WORD_TO_INT( 1122);           // 1122
  varInt[10] :=  DWORD_TO_INT( 16#1234_5678);   // 16#5678

END_FUNCTION_BLOCK
```

4.4 ANY_TO_DINT function

Conversion of a random variable of the elementary type into **DINT**.

The following function belong into this group of functions:
BOOL_TO_DINT
SINT_TO_DINT, INT_TO_DINT
USINT_TO_DINT, UINT_TO_DINT, UDINT_TO_DINT
REAL_TO_DINT, LREAL_TO_DINT
STRING_TO_DINT
TIME_TO_DINT, TOD_TO_DINT, DATE_TO_DINT, DT_TO_DINT

The result of the conversion …_TO_DINT is a number in within the range <–2 147 483 648, +2 147 483 647>.

When converting the REAL and LREAL data types the input value is firstly rounded off to the nearest whole number and then the conversion is undertaken.

Use of the …_TO_DINT conversion is shown in the following example:

```
FUNCTION_BLOCK Example_ANY_TO_DINT
  VAR_OUTPUT
    varDint      : ARRAY[1..10] OF DINT;
  END_VAR

  varDint[1]  :=   BOOL_TO_DINT( true);              // 1
  varDint[2]  :=   SINT_TO_DINT( 99);                // 99
  varDint[3]  :=   REAL_TO_DINT( 123.5678);          // 124
  varDint[4]  :=  LREAL_TO_DINT( 5.21E3);            // 5210
  varDint[5]  := STRING_TO_DINT( '-12.5');           // -12
  varDint[6]  := STRING_TO_DINT( '4.47E+06');        // 4
  varDint[7]  :=   TIME_TO_DINT( T#12s25ms);         // 12025
  varDint[8]  :=   BYTE_TO_DINT( BYTE#16#AF);        // 175
  varDint[9]  :=   WORD_TO_DINT( 1122);              // 1122
  varDint[10] :=  DWORD_TO_DINT( 1234_5678);         // 1234_5678

END_FUNCTION_BLOCK
```

4.5 ANY_TO_USINT function

Conversion of a random variable of the elementary type into **USINT**.

The following function belong into this group of functions:
BOOL_TO_USINT
SINT_TO_USINT, INT_TO_USINT, DINT_TO_USINT
UINT_TO_USINT, UDINT_TO_USINT
REAL_TO_USINT, LREAL_TO_USINT
STRING_TO_USINT
TIME_TO_USINT, TOD_TO_USINT, DATE_TO_USINT, DT_TO_USINT

The result of the conversion …_TO_USINT is a number in within the range <0, 255>. If we convert from a bigger data type into a smaller one (e.g. UDINT_TO_USINT) we are risking loss of information should the input value not be within the stated range.

When converting the REAL and LREAL data types the input value is firstly rounded off to the nearest whole number and then the conversion is undertaken.

Use of the …_TO_USINT conversion is shown in the following example:

```
FUNCTION_BLOCK Example_ANY_TO_USINT
  VAR_OUTPUT
    varUsint     : ARRAY[1..15] OF USINT;
  END_VAR

  varUsint[1]  :=    BOOL_TO_USINT( BOOL#0);                // 0
  varUsint[2]  :=    BOOL_TO_USINT( true);                  // 1
  varUsint[3]  :=  STRING_TO_USINT( '+127');                // 127
  varUsint[4]  :=    SINT_TO_USINT( -1);                    // 255
  varUsint[5]  :=     INT_TO_USINT( INT#-128);              // 128
  varUsint[6]  :=    DINT_TO_USINT( DINT#1_345_678);        // 142
  varUsint[7]  :=    UINT_TO_USINT( UINT#16#FFFF);          // 255
  varUsint[8]  :=   UDINT_TO_USINT( 16#FFFF_FFFE);          // 254
  varUsint[9]  :=    REAL_TO_USINT( 12.6);                  // 13
  varUsint[10] :=   LREAL_TO_USINT( 123.4567);              // 123
  varUsint[11] :=    TIME_TO_USINT( T#0:00:00.100);         // 100
  varUsint[12] :=    BYTE_TO_USINT( BYTE#16#AF);            // 175
  varUsint[13] :=    WORD_TO_USINT( 1122);                  // 98
  varUsint[14] :=   DWORD_TO_USINT( 12345678);              // 78

END_FUNCTION_BLOCK
```

4.6 *ANY_TO_UINT function*

Conversion of a random variable of the elementary type into **UINT**.

The following function belong into this group of functions:
BOOL_TO_UINT
SINT_TO_UINT, INT_TO_UINT, DINT_TO_UINT
USINT_TO_UINT, UDINT_TO_UINT
REAL_TO_UINT, LREAL_TO_UINT
STRING_TO_UINT
TIME_TO_UINT, TOD_TO_UINT, DATE_TO_UINT, DT_TO_UINT

The result of the conversion …_TO_USINT is a number in within the range < 0, +65535>. If we convert from a bigger data type into a smaller one (e.g. UDINT_TO_UINT) we are risking loss of information should the input value not be within the stated range.

When converting the REAL and LREAL data types the input value is firstly rounded off to the nearest whole number and then the conversion is undertaken.

Use of the …_TO_UINT conversion is shown in the following example:

```
FUNCTION_BLOCK Example_ANY_TO_UINT
  VAR_OUTPUT
    varUint      : ARRAY[1..15] OF UINT;
  END_VAR

  varUint[1]  :=   BOOL_TO_UINT( BOOL#0);            // 0
  varUint[2]  :=   BOOL_TO_UINT( true);              // 1
  varUint[3]  := STRING_TO_UINT( '+127');            // 127
  varUint[4]  :=   SINT_TO_UINT( -1);                // 65535
  varUint[5]  :=    INT_TO_UINT( INT#-128);          // 65408
  varUint[6]  :=   DINT_TO_UINT( DINT#1_345_678);    // 34958
  varUint[7]  :=  USINT_TO_UINT( USINT#16#FF);       // 255
  varUint[8]  :=  UDINT_TO_UINT( 16#FFFF_FFFE);      // 65534
  varUint[9]  :=   REAL_TO_UINT( 12.6);              // 13
  varUint[10] :=  LREAL_TO_UINT( 123.4567);          // 123
  varUint[11] :=   TIME_TO_UINT( T#0:00:00.100);     // 100
  varUint[12] :=   BYTE_TO_UINT( BYTE#16#AF);        // 175
  varUint[13] :=   WORD_TO_UINT( 1122);              // 1122
  varUint[14] :=  DWORD_TO_UINT( 12345678);          // 24910

END_FUNCTION_BLOCK
```

4.7 ANY_TO_ UDINT function

Conversion of a random variable of the elementary type into **UDINT**.

The following function belong into this group of functions:
BOOL_TO_UDINT
SINT_TO_UDINT, INT_TO_UDINT, DINT_TO_UDINT
USINT_TO_UDINT, UINT_TO_UDINT
REAL_TO_UDINT, LREAL_TO_UDINT
STRING_TO_UDINT
TIME_TO_UDINT, TOD_TO_UDINT, DATE_TO_UDINT, DT_TO_UDINT

The result of the conversion …_TO_UDINT is a number in within the range <0, +4 294 967 295>. If the converted value is not within this range (e.g. REAL_TO_UDINT or in case of negative numbers) then the result of the conversion is not defined.

When converting the REAL and LREAL data types the input value is firstly rounded off to the nearest whole number and then the conversion is undertaken.

Use of the …_TO_UDINT conversion is shown in the following example:

```
FUNCTION_BLOCK Example_ANY_TO_UDINT
  VAR_OUTPUT
    varUdint      : ARRAY[1..15] OF UDINT;
  END_VAR

  varUdint[1]  :=   BOOL_TO_UDINT( BOOL#0);               // 0
  varUdint[2]  :=   BOOL_TO_UDINT( true);                 // 1
  varUdint[3]  := STRING_TO_UDINT( '+127');               // 127
  varUdint[4]  :=   SINT_TO_UDINT( -1);                   // 4 294 967 295
  varUdint[5]  :=    INT_TO_UDINT( INT#-128);             // 4 294 967 168
  varUdint[6]  :=   DINT_TO_UDINT( DINT#1_345_678);       // 1 345 678
  varUdint[7]  :=  USINT_TO_UDINT( USINT#16#FF);          // 255
  varUdint[8]  :=   UINT_TO_UDINT( UINT#16#FFFE);         // 65534
  varUdint[9]  :=   REAL_TO_UDINT( 12.6);                 // 13
  varUdint[10] :=  LREAL_TO_UDINT( 123.4567);             // 123
  varUdint[11] :=   TIME_TO_UDINT( T#0:00:00.100);        // 100
  varUdint[12] :=   BYTE_TO_UDINT( BYTE#16#AF);           // 175
  varUdint[13] :=   WORD_TO_UDINT( 1122);                 // 1122
  varUdint[14] :=  DWORD_TO_UDINT( 12345678);             // 12345678

END_FUNCTION_BLOCK
```

4.8 ANY_TO_ REAL function

Conversion of a random variable of the elementary type into **REAL**.

The following function belong into this group of functions:
BOOL_TO_REAL
SINT_TO_REAL, INT_TO_REAL, DINT_TO_REAL
USINT_TO_REAL, UINT_TO_REAL, UDINT_TO_REAL
LREAL_TO_REAL
STRING_TO_REAL
TIME_TO_REAL, TOD_TO_REAL, DATE_TO_REAL, DT_TO_REAL

Use of the …_TO_REAL conversion is shown in the following example:

```
FUNCTION_BLOCK Example_ANY_TO_REAL
  VAR_OUTPUT
    varReal      : ARRAY[1..20] OF REAL;
  END_VAR

  varReal[1]  :=   BOOL_TO_REAL( 0);             // 0.0
  varReal[2]  :=   BOOL_TO_REAL( true);          // 1.0
  varReal[3]  :=   SINT_TO_REAL( -99);           // -99.0
  varReal[4]  :=    INT_TO_REAL( -9900);         // -9900.0
  varReal[5]  :=   DINT_TO_REAL( -1_235_678);    // -1 235 678.0
  varReal[6]  :=  USINT_TO_REAL( 99);            // 99.0
  varReal[7]  :=   UINT_TO_REAL( 9900);          // 9900.0
  varReal[8]  :=  UDINT_TO_REAL( 1_235_678);     // 1 235 678.0
  varReal[9]  := STRING_TO_REAL( '-12.5');       // -12.5
  varReal[10] := STRING_TO_REAL( '4.47E6');      // 4470000.0
  varReal[11] := STRING_TO_REAL( '4.47-E6');     // 4.47
  varReal[12] :=   TIME_TO_REAL( T#12s25ms);     // 12025.0
  varReal[13] :=  BYTE_TO_REAL( 16#AF);          // 175.0
  varReal[14] :=  WORD_TO_REAL( 1122);           // 1122.0
  varReal[15] := DWORD_TO_REAL( 1234567);        // 1234567.0

END_FUNCTION_BLOCK
```

4.9 *ANY_TO_ LREAL function*

Conversion of a random variable of the elementary type into **LREAL**.

The following function belong into this group of functions:
BOOL_TO_LREAL
SINT_TO_LREAL, INT_TO_LREAL, DINT_TO_LREAL
USINT_TO_LREAL, UINT_TO_LREAL, UDINT_TO_LREAL
REAL_TO_LREAL
STRING_TO_LREAL
TIME_TO_LREAL, TOD_TO_LREAL, DATE_TO_LREAL, DT_TO_LREAL

Use of the …_TO_LREAL conversion is shown in the following example:

```
FUNCTION_BLOCK Example_ANY_TO_LREAL
  VAR_OUTPUT
    varLreal     : ARRAY[1..10] OF LREAL;
  END_VAR

  varLreal[1] :=   BOOL_TO_LREAL( false);            // 0.0
  varLreal[2] :=   BOOL_TO_LREAL( true);             // 1.0
  varLreal[3] :=   SINT_TO_LREAL( 99);               // 99.0
  varLreal[4] :=   DINT_TO_LREAL( 1_235_678);        // 1 235 678.0
  varLreal[5] := STRING_TO_LREAL( '-12.5');          // -12.5
  varLreal[6] := STRING_TO_LREAL( '4.47E+05');       // 447000.0
  varLreal[7] :=   TIME_TO_LREAL( T#12s25ms);        // 12025.0
  varLreal[8] :=    TOD_TO_LREAL( TOD#12:33:05.120);  // 45185120.0
               // 45185120.0 = 120 + 5*1000 + 33*1000*60 + 12*1000*60*60

END_FUNCTION_BLOCK
```

4.10 ANY_TO_ STRING function

Conversion of a random variable of the elementary type into **STRING**.

The following function belong into this group of functions:
BOOL_TO_STRING
SINT_TO_STRING, INT_TO_STRING, DINT_TO_STRING
USINT_TO_STRING, UINT_TO_STRING, UDINT_TO_STRING
REAL_TO_STRING, LREAL_TO_STRING
TIME_TO_STRING, TOD_TO_STRING, DATE_TO_STRING, DT_TO_STRING

The result of the conversion …_TO_STRING is a text string.

The result of the conversion of ANY_NUM is a STRING type containing a number in the decimal syste. The + sign is not used by positive values. A decimal point is used as a decimal separator.

By time and date conversion the resulting text string begins with a abbreviation of the converted data type (e.g. by converting TIME_TO_STRING, the resulting string will be '**T#**...'). Then the time information follows in a format that corresponds to the time literals during the inicialization of the time variables.

Use of the …_TO_STRING conversion is shown in the following example:

```
FUNCTION_BLOCK Example_ANY_TO_STRING
  VAR_OUTPUT
    varString    : ARRAY[1..20] OF STRING[30];
  END_VAR

  varString[1]  :=  BOOL_TO_STRING( false);           // '0'
  varString[2]  :=  BOOL_TO_STRING( true);            // '1'
  varString[3]  :=  SINT_TO_STRING( 99);              // '99'
  varString[4]  :=   INT_TO_STRING( -9_998);          // '-9998'
  varString[5]  :=  DINT_TO_STRING( 1_235_678);       // '1235678'
  varString[6]  := USINT_TO_STRING( 255);             // '255'
  varString[7]  :=  UINT_TO_STRING( 16#FFFF);         // '65535'
  varString[8]  := UDINT_TO_STRING( 16#FFFF_FFFF);    // '4294967295'
  varString[9]  :=  REAL_TO_STRING( -123E5);          // '-12300000.00000'
  varString[10] := LREAL_TO_STRING( 123.4567);        // '123.456700'
  varString[11] :=  TIME_TO_STRING( TIME#12s25ms);    // 'T#0:00:12.025'
  varString[12] :=   TOD_TO_STRING( TOD#12:33:05.120);// 'TOD#12:33:5.120'
  varString[13] :=  DATE_TO_STRING( DATE#2003-10-21); // 'D#2003-10-21'
  varString[14] :=    DT_TO_STRING( DT#2003-10-21-16:35:59);
                                         // 'DT#2003-10-21-16:35:59.000'
  varString[15] :=  BYTE_TO_STRING( 16#AF);           // '175'
  varString[16] :=  WORD_TO_STRING( 1122);            // '1122'
  varString[17] := DWORD_TO_STRING( 12345678);        // '12345678'

END_FUNCTION_BLOCK
```

4.11 ANY_TO_TIME function

Conversion of a random variable of the elementary type into **TIME**.

The following function belong into this group of functions:
BOOL_TO_TIME
SINT_TO_TIME, INT_TO_TIME, DINT_TO_TIME
USINT_TO_TIME, UINT_TO_TIME, UDINT_TO_TIME
REAL_TO_TIME, LREAL_TO_TIME
STRING_TO_TIME
TOD_TO_TIME, DATE_TO_TIME, DT_TO_TIME

The result of the conversion …_TO_TIME is a number that states the number of miliseconds.

When converting from the STRING type the input string must correspond to the TIME literal.

Use of the …_TO_TIME conversion is shown in the following example:

```
FUNCTION_BLOCK Example_ANY_TO_TIME
  VAR_OUTPUT
    varTime      : ARRAY[1..11] OF TIME;
  END_VAR

  varTime[1]  :=   BOOL_TO_TIME( BOOL#0);            // T#00h 00m 00.000s
  varTime[2]  :=   BOOL_TO_TIME( true);              // T#00h 00m 00.001s
  varTime[3]  :=    INT_TO_TIME( 1000);              // T#00h 00m 01.000s
  varTime[4]  :=   DINT_TO_TIME( 10*1000);           // T#00h 00m 10.000s
  varTime[5]  :=  UDINT_TO_TIME( 60*1000);           // T#00h 01m 00.000s
  varTime[6]  :=   REAL_TO_TIME( 60.*60.*1000.);     // T#01h 00m 00.000s
  varTime[7]  :=  LREAL_TO_TIME( 11.*60.*60.*1000.); // T#11h 00m 00.000s
  varTime[8]  := STRING_TO_TIME( 'T#06:30:15.250');  // T#06h 30m 15.250s
  varTime[9]  := STRING_TO_TIME( '06:30:15.250');    // T#00h 00m 00.000s
  varTime[10] := DT_TO_TIME( DT#1970-01-01-12:34:56);// T#12h 34m 56.000s

END_FUNCTION_BLOCK
```

4.12 ANY_TO_ TIME_OF_DAY function

Conversion of a random variable of the elementary type into **TIME_OF_DAY**. This data type can be also known as TOD.

The following function belong into this group of functions:
BOOL_TO_TIME_OF_DAY
SINT_TO_TIME_OF_DAY, INT_TO_TIME_OF_DAY, DINT_TO_TIME_OF_DAY
USINT_TO_TIME_OF_DAY, UINT_TO_TIME_OF_DAY, UDINT_TO_TIME_OF_DAY
REAL_TO_TIME_OF_DAY, LREAL_TO_TIME_OF_DAY
STRING_TO_TIME_OF_DAY
TIME_TO_TIME_OF_DAY, DATE_TO_TIME_OF_DAY, DT_TO_TIME_OF_DAY
or
BOOL_TO_TOD
SINT_TO_TOD, INT_TO_TOD, DINT_TO_TOD
USINT_TO_TOD, UINT_TO_TOD, UDINT_TO_TOD
REAL_TO_TOD, LREAL_TO_TOD
STRING_TO_TOD
TIME_TO_TOD, DATE_TO_TOD, DT_TO_TOD

The result of the conversion …_TO_TIME_OF_DAY is a number that states the number of miliseconds.

When converting from the STRING type the input string must correspond to the TOD literal.

Use of the …_TO_TIME_OF_DAY conversion is shown in the following example:

```
FUNCTION_BLOCK Example_ANY_TO_TIME_OF_DAY
  VAR_OUTPUT
    varTod      : ARRAY[1..11] OF TOD;
  END_VAR

  varTod[1]  :=   BOOL_TO_TOD( BOOL#0);              // TOD#00:00:00.000
  varTod[2]  :=   BOOL_TO_TOD( true);                // TOD#00:00:00.001
  varTod[3]  :=    INT_TO_TOD( 1000);                // TOD#00:00:01.000
  varTod[4]  :=   DINT_TO_TOD( 10*1000);             // TOD#00:00:10.000
  varTod[5]  :=  UDINT_TO_TOD( 60*1000);             // TOD#00:01:00.000
  varTod[6]  :=   REAL_TO_TOD( 60.*60.*1000.);       // TOD#01:00:00.000
  varTod[7]  :=  LREAL_TO_TOD( 11.*60.*60.*1000.);   // TOD#11:00:00.000
  varTod[8]  := STRING_TO_TOD( 'TOD#06:30:15.250');  // TOD#06:30:15.250
  varTod[9]  := STRING_TO_TOD( '06:30:15.250');      // TOD#00:00:00.000
  varTod[10] :=  DT_TO_TOD( DT#1970-01-01-12:34:56); // TOD#12:34:56.000

END_FUNCTION_BLOCK
```

4.13 ANY_TO_DATE function

Conversion of a random variable of the elementary type into **DATE**.

The following function belong into this group of functions:
BOOL_TO_DATE
SINT_TO_DATE, INT_TO_DATE, DINT_TO_DATE
USINT_TO_DATE, UINT_TO_DATE, UDINT_TO_DATE
REAL_TO_DATE, LREAL_TO_DATE
STRING_TO_DATE
TIME_TO_DATE, TOD_TO_DATE, DT_TO_DATE

The result of the conversion **..._TO_DATE** is a number whose whole part determines the numbers of seconds from 00:00:00 1.1.1970.

When converting from the STRING type the input string must correspond to the DATE literal.

Use of the …_TO_DATE conversion is shown in the following example:

```
FUNCTION_BLOCK Example_ANY_TO_DATE
  VAR_OUTPUT
    varDate      : ARRAY[1..11] OF DATE;
  END_VAR

  varDate[1]  :=   BOOL_TO_DATE( BOOL#0);                  // D#1970-01-01
  varDate[2]  :=   BOOL_TO_DATE( true);                    // D#1970-01-01
  varDate[3]  :=   DINT_TO_DATE( 24*60*60);                // D#1970-01-02
  varDate[4]  :=  UDINT_TO_DATE( 11*24*60*60);             // D#1970-01-12
  varDate[5]  :=   REAL_TO_DATE( 365.*24.*60.*60.);        // D#1971-01-01
  varDate[6]  :=  LREAL_TO_DATE( 10.*365.*24.*60.*60.);    // D#1979-12-30
  varDate[7]  :=   TIME_TO_DATE( T#25:00:00.0);            // D#1970-01-02
  varDate[8]  := STRING_TO_DATE( 'D#2003-12-24');          // D#2003-12-24
  varDate[9]  := STRING_TO_DATE( '2003-12-24');            // D#1970-01-01
  varDate[10] :=     DT_TO_DATE( DT#2002-11-25-00:00:00); // D#2002-11-25
  varDate[11] :=     DT_TO_DATE( DT#2002-11-25-12:22:33); // D#2002-11-26

END_FUNCTION_BLOCK
```

4.14 ANY_TO_ DATE_AND_TIME function

Conversion of a random variable of the elementary type into **`DATE_AND_TIME`**.

The following function belong into this group of functions:
BOOL_TO_DATE_AND_TIME
SINT_TO_DATE_AND_TIME, INT_TO_DATE_AND_TIME,
DINT_TO_DATE_AND_TIME, USINT_TO_DATE_AND_TIME
UINT_TO_DATE_AND_TIME, UDINT_TO_DATE_AND_TIME
REAL_TO_DATE_AND_TIME, LREAL_TO_DATE_AND_TIME
STRING_TO_DATE_AND_TIME
TIME_TO_DATE_AND_TIME, TOD_TO_DATE_AND_TIME
DATE_TO_DATE_AND_TIME
or
BOOL_TO_DT
SINT_TO_DT, INT_TO_DT, DINT_TO_DT
USINT_TO_DT, UINT_TO_DT, UDINT_TO_DT
REAL_TO_DT, LREAL_TO_DT
STRING_TO_DT
TIME_TO_DT, TOD_TO_DT
DATE_TO_DT

The result of the conversion …_TO_DATE_AND_TIME is a number whose whole part determines the numbers of seconds from 00:00:00 1.1.1970. The decimal part of the number represents milliseconds.

When converting from the STRING type the input string must correspond to the DATE_AND_TIME literal.

Use of the …_TO_ DATE_AND_TIME conversion is shown in the following example:

```
FUNCTION_BLOCK Example_ANY_TO_DT
  VAR_OUTPUT
    varDt     : ARRAY[1..11] OF DATE_AND_TIME;
  END_VAR

  varDt[1]  :=  BOOL_TO_DT( BOOL#0);              // DT#1970-01-01-00:00:00
  varDt[2]  :=  BOOL_TO_DT( true);                // DT#1970-01-01-00:00:01
  varDt[3]  :=  DINT_TO_DT( 24*60*60);            // DT#1970-01-02-00:00:00
  varDt[4]  := UDINT_TO_DT( 11*24*60*60 + 35);    // DT#1970-01-12-00:00:35
  varDt[5]  :=  REAL_TO_DT( 365.*24.*60.*60.);    // DT#1971-01-01-00:00:00
  varDt[6]  := LREAL_TO_DT( 10.*365.*24.*60.*60.);//DT#1979-12-30-00:00:00
  varDt[7]  :=   TIME_TO_DT( T#25:00:00.0);       // DT#1970-01-02-01:00:00
  varDt[8]  := STRING_TO_DT( 'DT#2003-12-24-18:10:20');
                                                  // DT#2003-12-24-18:10:20
  varDt[9]  := STRING_TO_DT( '2003-12-24');       // DT#1970-01-01-00:00:00
  varDt[10] :=   DATE_TO_DT( DATE#2002-11-25);    // DT#2002-11-25-00:00:00

END_FUNCTION_BLOCK
```

5 ARITHMETIC FUNCTIONS

5.1 ABS absolute value function

The ABS function returns the absolute value of the input value. The input value may be of the ANY_NUM value.

```
PROGRAM ExampleABS
  VAR
    varA         : INT   := -22;
    varR         : REAL  := -12.5;
    absA         : INT;
    absR         : REAL;
    absI,absL    : LREAL;
  END_VAR

  absA  := ABS( varA);                          // 22
  absR  := ABS( varR);                          // 12.5
  absI  := ABS( INT_TO_LREAL(-123));            // 123
  absL  := ABS( LREAL#-345.678);                // 345.678
END_PROGRAM
```

5.2 SQRT square root function

The SQRT function returns the square root of the input parameter. The input parameter may not be a negative number.

```
PROGRAM ExampleSQRT
  VAR
    varR         : REAL := 144;
    sqrtR        : REAL;
    sqrtI,sqrtL  : LREAL;
  END_VAR

  sqrtR  := SQRT( varR);                          // 12
  sqrtI  := SQRT( INT_TO_LREAL(-123));            // NaN
  sqrtL  := SQRT( 1024.0);                        // 32.0
END_PROGRAM
```

5.3 LN natural logarithm function

The LN function returns the natural logarithm of an input parameter.

```
PROGRAM ExampleLN
  VAR
    varA         : INT   := 1;
    varR         : REAL := 2.718282;
    lnA, lnR     : REAL;
    lnI, lnL     : LREAL;
  END_VAR

  lnA := LN( INT_TO_REAL(varA));        // 0
  lnR := LN( varR);                     // 1.00000
  lnI := LN( INT_TO_LREAL(-123));       // NaN
  lnL := LN( 22026.3);                  // 9.99999
END_PROGRAM
```

5.4 LOG decimal logarithm function

The LOG function returns a decimal logarithm of the input parameter.

```
PROGRAM ExampleLOG
  VAR
    varA         : INT   := 1;
    varR         : REAL := 100.0;
    logA, logR   : REAL;
    logI, logL   : LREAL;
  END_VAR

  logA  := LOG( INT_TO_REAL(varA));       // 0
  logR  := LOG( varR);                    // 2.0
  logI  := LOG( INT_TO_LREAL(-123));      // NaN
  logL  := LOG( LREAL#1_000.0);           // 3.0
END_PROGRAM
```

5.5 EXP natural exponential function

The EXP function returns the value e^x, where x is the input parameter.

```
PROGRAM ExampleEXP
  VAR
    exp1, exp2   : REAL;
    exp3         : LREAL;
  END_VAR

  exp1 := EXP( REAL#2.0);                    //  7.3890
  exp2 := EXP( 0.0);                         //  1.0
  exp3 := EXP( 1.0);                         //  2.7182
END_PROGRAM
```

5.6 SIN Sine of input angle function

The SIN function returns a sine of the input parameter entered in radians. The input parameter must be within the range < -π/2, π/2 >.

```
PROGRAM ExampleSIN
  VAR CONSTANT
    PI           : LREAL := LREAL#3.14159265358979323846;
  END_VAR
  VAR
    sin1, sin2   : REAL;
    sin3, sin4   : LREAL;
  END_VAR

  sin1 := SIN( REAL#-3.14159 / 2.0);         // -1.0
  sin2 := SIN( 0.0);                         //  0.0
  sin3 := SIN( PI / 2.);                     //  1.0
  sin4 := SIN( 2.0 * PI);                    //  0.0
END_PROGRAM
```

5.7 COS Cosine of input angle

The COS function returns a cosine of the input parameter entered in radians. The input parameter must be within the range < -π/2, π/2 >.

```
PROGRAM ExampleCOS
  VAR CONSTANT
    PI           : LREAL := LREAL#3.14159265358979323846;
  END_VAR
  VAR
    cos1, cos2   : REAL;
    cos3, cos4   : LREAL;
  END_VAR

  cos1  := COS( REAL#-3.1415 / 2.0);        //  0.0
  cos2  := COS( 0.0);                       //  1.0
  cos3  := COS( PI / 2.);                   //  0.0
  cos4  := COS( 2.0 * PI);                  //  1.0
END_PROGRAM
```

5.8 TAN tangent of input angle

The TAN function returns a tangent of the input parameter entered in radians. The input parameter must be within the range < -π/2, π/2 >.

```
PROGRAM ExampleTAN
  VAR CONSTANT
    PI           : LREAL := LREAL#3.14159265358979323846;
  END_VAR
  VAR
    tan1, tan2   : REAL;
    tan3, tan4   : LREAL;
  END_VAR

  tan1  := TAN( REAL#-3.14159 / 4.0);       //  -1.0
  tan2  := TAN( 0.0);                       //   0.0
  tan3  := TAN( PI / 2.);                   //  +INF
  tan4  := TAN( -PI);                       //  -INF
END_PROGRAM
```

5.9 *ASIN arc sine function*

The ASIN function returns an arc sine of the input parameter. The input parameter must be within the range < -1, 1 >.

```
PROGRAM ExampleASIN
  VAR
    asin1, asin2  : REAL;
    asin3, asin4  : LREAL;
  END_VAR

  asin1 := ASIN( REAL#-1.0);                // -1.570796 (-PI/2)
  asin2 := ASIN( 0.0);                      //   0.0
  asin3 := ASIN( 0.0);                      //   0.0
  asin4 := ASIN( 1.0);                      //   1.570796 (PI/2)
END_PROGRAM
```

5.10 *ACOS Arc cosine function*

The ACOS function returns an arc cosine of the input parameter. The input parameter must be within the range < -1, 1 >.

```
PROGRAM ExampleACOS
  VAR
    acos1, acos2  : REAL;
    acos3         : LREAL;
  END_VAR

  acos1 := ACOS( REAL#-1.0);                //   3.14159 (PI)
  acos2 := ACOS( 0.0);                      //   1.15079 (PI/2)
  acos3 := ACOS( 1.0);                      //   0.0
END_PROGRAM
```

5.11 ATAN arc tangent function

The ATAN function returns an arc tangent of the input parameter.

```
PROGRAM ExampleATAN
  VAR CONSTANT
    PI           : LREAL := LREAL#3.14159265358979323846;
  END_VAR
  VAR
    atan1, atan2   : REAL;
    atan3          : LREAL;
  END_VAR

  atan1 := ATAN( REAL#-3.14159 / 2.0);       //  -1.0
  atan2 := ATAN( 0.0);                       //   0.0
  atan3 := ATAN( PI / 2.);                   //  +1.0
END_PROGRAM
```

CONTENTS