JUMO diraTRON/diraVIEW 104/108/116/132

Compact controller/digital indicator

ST Editor Manual

70211000T96Z001K000

V2.00/EN/00688820

Contents

Contents

1.1 Safety information

Note symbols

NOTE!

This symbol refers to **important information** about the product, its handling, or additional benefits.

REFERENCE!

This symbol refers to **additional information** in other sections, chapters, or other manuals.

1.2 Intended use

The device is designed for use in an industrial environment as specified in the technical data. Other uses beyond those defined are not viewed as intended uses.

The device has been manufactured in compliance with applicable standards and directives as well as the applicable safety regulations. Nevertheless, improper use may lead to personal injury or material damage.

To avoid danger, only use the device:

- For the intended use
- When in good order and condition
- When taking the technical documentation provided into account

Risks resulting from the application may arise, e.g. as the result of missing safety provisions or wrong settings, even when the device is used properly and as intended.

1.3 Qualification of personnel

This document contains the necessary information for the intended use of the device to which it relates.

It is intended for staff with technical qualifications who have been specially trained and have the appropriate knowledge in the field of automation technology.

The appropriate level of knowledge and the technically fault-free implementation of the safety information and warnings contained in the technical documentation provided are prerequisites for risk-free mounting, installation, and startup as well as for ensuring safety when operating the described modules. Only qualified personnel have the required specialist knowledge to correctly interpret and implement the safety information and warnings contained in this document in specific situations.

1.4 Content of this document

NOTE!

This document applies to devices from the 70211x series (compact controllers) as well as to devices from the 70151x series (digital indicators).

This document describes the application of the ST editor with which users can create their own applications in the PLC programming language "Structured Text" (ST) for the device. The document is intended for users with relevant programming knowledge.

The PLC programming language "Structured Text" (ST) is described in the standard DIN IEC 61131-3. Detailed information about programming can be found in this standard. The ST module of the respective device supports only a subset of the programming language described in the standard.

1 Introduction

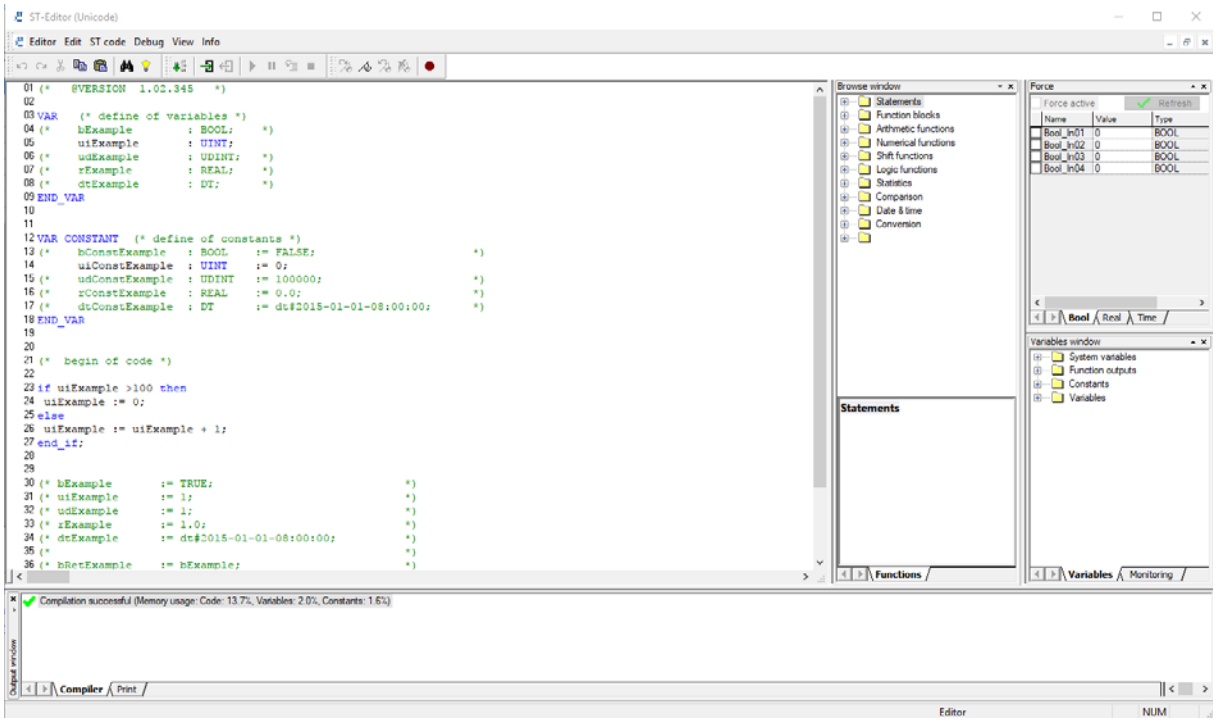
In addition to this document, the operating manual of the respective device series must be observed:

- Types 701510, 701511, 701512, 701513, 701514 (digital indicators):
  Document 70151000T90Z...K...
- Types 702110, 702111, 702112, 702113, 702114 (compact controllers):
  Document 70211000T90Z...K...

The ST editor is part of the setup program and is started by clicking on the corresponding button in the "ST code" window (see operating manual).

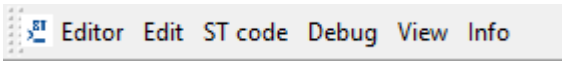
The finished application is transmitted to the device as ST code and continuously processed in the integrated ST module.

Overview



The user interface consists of several toolbars and windows, which are briefly described in the following sections.

Menu bar



The individual menus contain functions for editing and compiling a program, for troubleshooting, for displaying and hiding toolbars and windows of the user interface, as well as ST editor version information.

Further information about the functions of the menu bar is displayed in the status bar when the mouse pointer hovers over the individual function in the respective menu.

2 User interface

Toolbar

Some functions of the menu bar are also available in the toolbar and can be selected by a simple mouse click. The meaning of the symbols is briefly described by a tool tip function (hover over the respective symbol with the mouse pointer). In addition, further information about the relevant function is then also displayed in the status bar.

Symbol	Meaning	Description
	Undo (Ctrl+Z)	Undo previous action
	Restore	Restore previously undone action
	Cut (Ctrl+X)	Remove selected data and transfer to clipboard
	Copy (Ctrl+C)	Copy selected data and transfer to clipboard
	Paste (Ctrl+V)	Insert data from the clipboard
	Find (Ctrl+F)	Search for entered text
	Function text (F4)	Insert function text from the browser window See also section "Browse window ", page 11.
	Compile (F7)	Compile ST code
	Start debugging	Start debugging (cold start) After starting, a program that is already on the device is terminated. Instead, the current program is loaded onto the device. See also section "Debugging ", page 13.
	Terminate debugging	Terminate debugging After terminating, the current program remains on the device and is run. See also section "End ST editor ", page 13.
	Start/continue (F5)	Start program or continue running after interruption
	Interrupt program	Interrupt program
	Single step (F11)	Run program in a single step
	Terminate	Terminate program
	Next bookmark (F2)	Move insertion mark to the next bookmark
	Toggle bookmark (Ctrl+F2)	Toggle bookmark for the current line
	Previous bookmark (Shift+F2)	Move insertion mark to the previous bookmark
	Delete bookmark (Shift+Ctrl+F2)	Delete all bookmarks

Symbol	Meaning	Description
	Toggle breakpoint (F9)	Toggle breakpoint for the current line

Edit window

```
(*   @VERSION  1.02.345   *)

VAR   (* define of variables *)
(*    bExample        : BOOL;    *)
      uiExample       : UINT;
(*    udExample       : UDINT;   *)
(*    rExample        : REAL;    *)
(*    dtExample       : DT;      *)
END_VAR


VAR CONSTANT  (* define of constants *)
(*    bConstExample   : BOOL    := FALSE;                      *)
      uiConstExample  : UINT    := 0;
(*    udConstExample  : UDINT   := 100000;                     *)
(*    rConstExample   : REAL    := 0.0;                        *)
(*    dtConstExample  : DT      := dt#2015-01-01-08:00:00;     *)
END_VAR


(*  begin of code *)

if uiExample >100 then
 uiExample := 0;
else
 uiExample := uiExample + 1;
end_if;

```

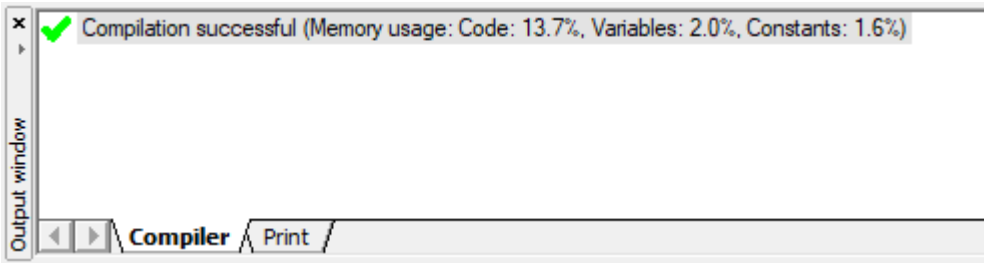
The program is created in the edit window by declaring variables and constants, creating program code and using commentary texts.

In order to assign a version name to the program, the keyword @VERSION followed by the version name must be used in a comment line. If the program contains several comment lines with this keyword, only the first comment line is evaluated. After the ST code has been permanently transferred to the device, the version name can be displayed on the device (Device info > Versions > ST code version).

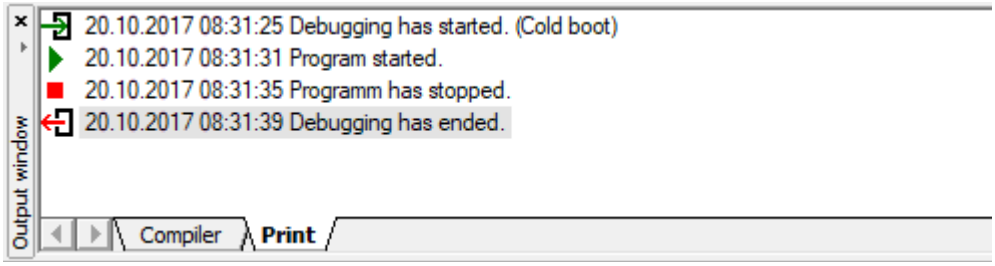
It is possible to change the font size using Ctrl+Mouse wheel.

Output window

The output window consists of two parts (tabs).

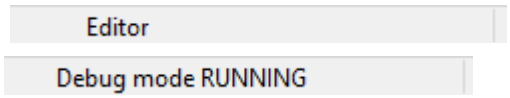


The "Compiler" tab shows messages concerning the compilation of the program code.



The "Print" tab contains messages concerning the debug mode.

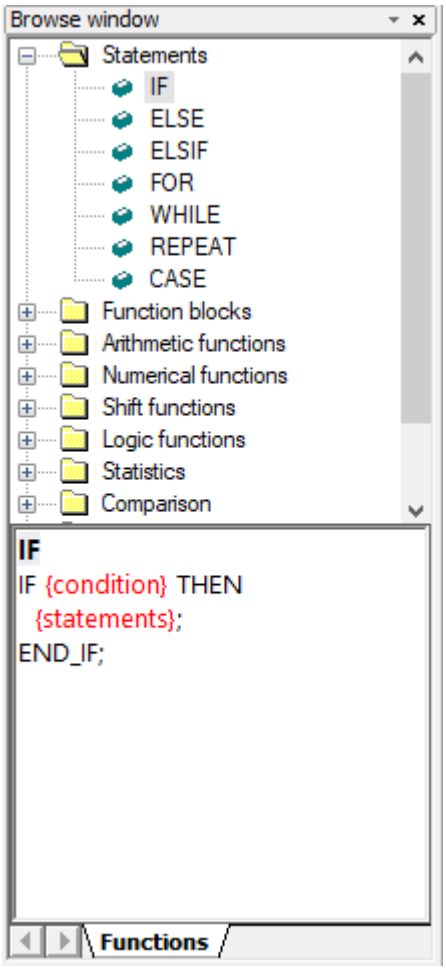
Status bar



The status bar is located at the bottom of the program interface and indicates whether the editor or debug mode is active. In the case of debug mode, the individual states (STOP, RUN, PAUSE) are displayed.

Further information about the functions of the menu bar and the toolbar is also displayed in the status bar (hover over the individual function in the respective menu or over the symbol in the toolbar with the mouse pointer).

Browse window

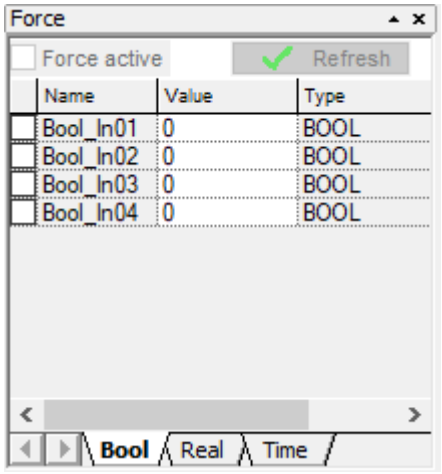


The browse window lists all the instructions and functions that can be used in the editor. The use of the instruction or function is shown in the bottom part of the window.

The function text displayed in the bottom part of the browse window can be inserted into the ST code in the desired position by dragging and dropping (select text beforehand), by clicking on the "Function text" icon, or using the F4 function key in the edit window (where the cursor is or instead of the selected text). After inserting, the next placeholder is selected by clicking on the symbol again or pressing F4.

Force window

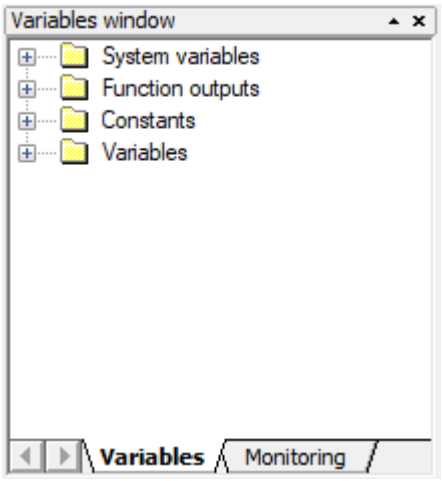
The force window consists of three parts (tabs). Only the "Bool" tab is shown here.



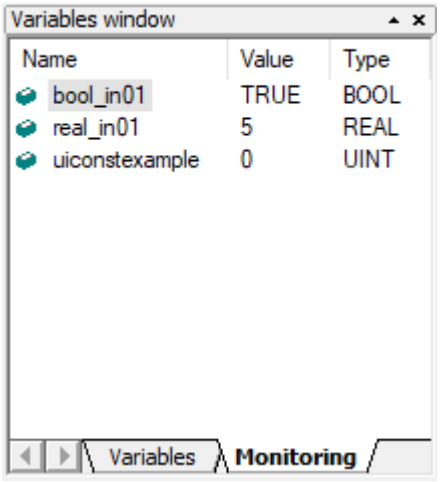
The "Bool" and "Real" input variables as well as date and time can be set by the user in the force window in order to test the program code.

Variables window

The variables window consists of two parts (tabs).



The "Variables" tab lists all system variables and function outputs (outputs of the function blocks) as well as the constants and variables (markers) declared by the user. They can be copied from the variable window by dragging and dropping and pasted into the ST code at the desired position in the edit window.



Variables and constants can be displayed and observed during debugging in the "Monitoring" tab. To do so, they must be copied from the edit window or the "Variables" tab by dragging and dropping and pasted into the "Monitoring" tab.

Pasting and removing is only possible if the program is not running. Different functions are then also available via the context menu (right mouse button).

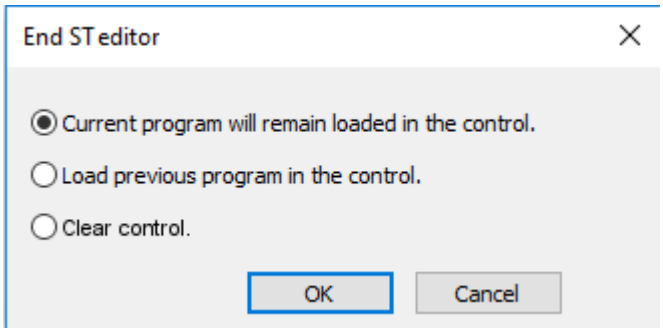
Debugging

The following variants are available with the *Debug > Start debugging* (menu bar) function:

- Cold start: A program that is already running on the device is terminated and the current program from the ST editor is loaded onto the device. All variables (retain and non-retain variables) are reset to their default values.
- Warm start: The retain variables are not reset (70211x and 70151x device types do not support retain variables).
- Service start: The program currently running on the device is used in the ST editor and can be analyzed. No variables are reset here. A prerequisite for the service start is that the code compiled in the ST editor corresponds to the code on the device.

End ST editor

When ending the ST editor with *Editor > Save and end (finish)* (menu bar), there are the following options (if debugging mode was active):



- *Current program will remain loaded on the control.*
  The program last loaded onto the device in debugging mode must be used.
- *Load previous program onto the control.*
  The program last loaded onto the device in debugging mode must not be used. Instead, the previous (continuously loaded onto the device) program is used, if available.
- *Clear control.*
  Neither the program last loaded onto the device in debugging mode nor the previous program must be used.

NOTE!

In debugging mode, the program is only temporarily loaded onto the device (no backup in case of a power failure). In order to make changes permanently effective on the device, the setup file must be transferred to the device after exiting the ST editor (close ST code window by pressing "OK", data transfer to the device).

NOTE!

The source code created in the ST editor is transferred into the device in compiled form. It is therefore necessary to save the source code separately in a setup file.

The system variables include the input and output variables as well as the internal variables.

The input and output variables can be used to exchange values of different data types between the device and the ST module integrated in the device. The internal variables are only relevant within the ST module and can be used to implement certain functions.

3.1 Input variables

The input variables provide device values for use in the ST module.

The values are assigned to the input variables in the setup program by selecting the respective signals from the selectors (analog selector or digital selector). In addition, there are some variables that are permanently assigned in the device and therefore do not occur in the selectors.

bool_in

Designation	Designation in the setup program	Description
bool_in01 to bool_in04	bool_in01 to bool_in04	Boolean input variables; flexible assignment in the setup program (digital selector)

real_in

Designation	Designation in the setup program	Description
real_in01 to real_in06	real_in01 to real_in06	Real input variables; flexible assignment in the setup program (analog selector)

dword_in

Designation	Designation in the setup program	Description
dword_in01 dword_in02	(no designation, no fixed assignment)	(These double-word input variables are not used.)

rtc.cdt

Designation	Designation in the setup program	Description
rtc.cdt	(no designation, fixed assignment)	This type DT input variable returns the time since the device was switched on (date and time start each time the device is switched on from 01.01.1970 00:00:00:00).

3 System variables

3.2 Output variables

The output variables are used to transfer the values generated in the ST module to the device.

The output variables are available for configuration in the setup program and in the selectors (analog selector or digital selector) on the device, and can be used individually. In addition, there are some variables that have certain functions on the device (e.g. text display) and therefore cannot be selected using the selectors.

For certain output variables there are associated variables in the variable window with the designations "...conf" and "...sec". These variables can be used to implement a defined behavior in the ST code in the event of an error.

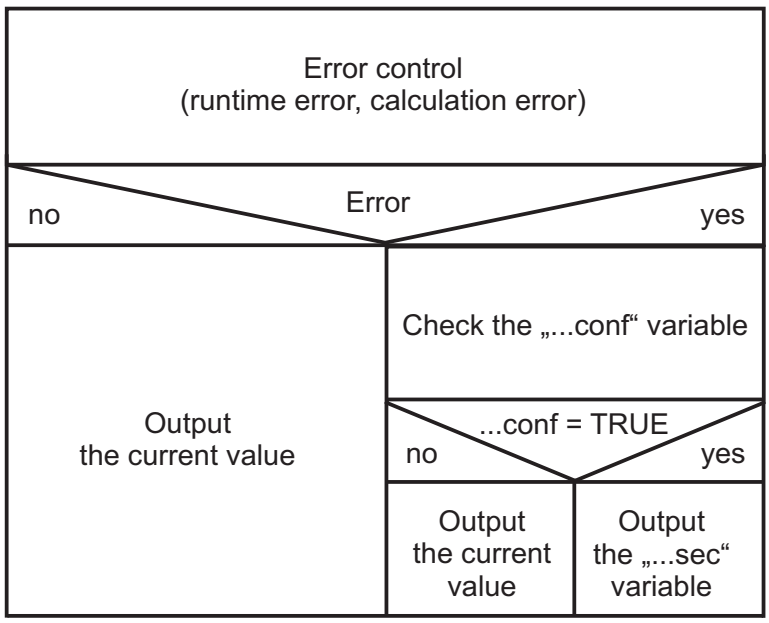
- The **variable "...conf"** can be set in the ST code to decide which value the corresponding output variable will accept in case of an error: FALSE = current value; TRUE = replacement value.

  Default setting: FALSE
- The **variable "...sec"** contains the replacement value; it can be set to a certain value or to the default value in the ST code.

  In case of REAL variables, the replacement value is checked for compliance with the scaling limits. If it is outside the limits, the default value is used.

  The SET_DEFAULT function is available for setting the default value.

  ⇨ 8.9 "Other functions", page 42



bool_out

Designation	Designation in the setup program and on the device	Description
bool_out01 to bool_out04	1. ST digital output to 4. ST digital output	Boolean output variables; flexible use in the setup program and on the device (digital selector) A description text can be assigned to each variable in the setup program.

real_out

Designation	Designation in the setup program and on the device	Description
real_out01 to real_out06	1. ST analog output to 6. ST analog output	Real output variables; flexible use in the setup program and on the device (analog selector) A description text can be assigned to each variable in the setup program. It also requires settings that affect the variable (e.g. scaling).

alarm

Designation	Designation in the setup program and on the device	Description
alarm	ST alarm	Boolean output variable; flexible use in the setup program and on the device (digital selector) Unlike the bool_out variables, this variable has a special designation. However, it can be freely used in the ST code.

error_out

Designation	Designation in the setup program and on the device	Description
error_out	ST error	Boolean output variable; flexible use in the setup program and on the device (digital selector) This variable is set to TRUE if the ST code cannot be processed further, such as in the following cases: • Division by 0 • Incorrect array access • Runtime exceeded The variable is set to FALSE again if the ST code has been completely run through once after the error occurred.

dword_out

Designation	Designation in the setup program and on the device	Description
dword_out01, dword_out02	(no designation, fixed assignment)	Double-word output variables of the application; fixed assignment: text display on the device display The texts must be entered in the "Display/operation" dialog under "Display texts" in the setup program.

Variable	Value	Function
dword_out01	0	No text display
	1 to 10	Devices in formats 108H, 108Q, 104: The respective text (1 to 10) is displayed in line 3. Devices in formats 132, 116: The respective text (1 to 10) is displayed in line 1.

Variable	Value	Function
dword_out02	0	No text display
	1 to 10	Devices in formats 108H, 108Q, 104: The respective text (1 to 10) is displayed in line 4. Devices in formats 132, 116: The respective text (1 to 10) is displayed in line 2.

3.3 Internal variables

The internal variables can be used in the ST code to implement certain functions.

ext_error

Designation	Description
ext_error	Integer variable The variable is 0 if the function Set_CP (set controller parameter) was executed without errors. If an error occurs, the variable returns a value > 0.

reset_flag

Designation	Description
reset_flag	Boolean variable The variable is set to TRUE after power on. If the ST code is completely run through once without an error after power on, the variable is set to FALSE. Otherwise, it stays as TRUE. Debug mode: The first time the program is started in debug mode or after a program stop, the variable is TRUE. After a second run through, it is set to FALSE.

sys_error

Designation	Description
sys_error	Real variable The variable returns an error value if an error occurs during program execution: 1.00E+37 = general error value 1.0E+37 = underrange (underflow) 2.0E+37 = overrange (overflow) 3.0E+37 = invalid input value 4.0E+37 = division by zero 5.0E+37 = incorrect math value 7.0E+37 = invalid value (general)

week_day

Designation	Description
week_day	Integer variable The variable includes the current day: 0 = Sunday, 1 = Monday, 2 = Tuesday, 3 = Wednesday, 4 = Thursday, 5 = Friday, 6 = Saturday The day is derived from the input variable rtc.cdt.

3 System variables

The ST module supports the following data types:

- Boolean value (BOOL)
- Integer without prefix sign, 2 bytes (UINT)
- Double integer without prefix sign, 4 bytes (UDINT)
- Floating-point number (REAL)
- Date and time (DT or DATE_AND_TIME)

NOTE!

No range monitoring of the data types takes place during operations.
Exceptions: square root function (SRQT), reciprocal (1/x)

NOTE!

The ST module does not support variables that are stored via power off (retain variables).

Boolean value

Keyword: BOOL

Value range: TRUE (1) or FALSE (0)

Declaration of a variable (example):

```
VAR
    bExample : BOOL;
END_VAR
```

Declaration of a constant (example):

```
VAR CONSTANT
    bConstExample : BOOL := FALSE;
END_VAR
```

Assignment (example):

```
bExample := TRUE;
```

Integer (2 bytes)

Keyword: UINT

Value range: 0 to 65535 (0 to 2^{16}-1)

Declaration of a variable (example):

```
VAR
    uiExample : UINT;
END_VAR
```

Declaration of a constant (example):

```
VAR CONSTANT
    uiConstExample : UINT := 0;
END_VAR
```

Assignment (example):

```
uiExample := 1;
```

Double integer (4 bytes)

Keyword: UDINT

Value range: 0 to 4294967295 (0 to 2^{32}-1)

Declaration of a variable (example):

```
VAR
    udExample : UDINT;
END_VAR
```

Declaration of a constant (example):

```
VAR CONSTANT
    udConstExample : UDINT := 100000;
END_VAR
```

Assignment (example):

```
udExample := 200000;
```

Floating-point number

Keyword: REAL

Value range: -1.5E-45 to 1.0E37
(Values ≥ 1.0E37 are interpreted as error values.)

Declaration of a variable (example):

```
VAR
    rExample : REAL;
END_VAR
```

Declaration of a constant (example):

```
VAR CONSTANT
    rConstExample : REAL := 0.0;
END_VAR
```

Assignment (example):

```
rExample := 1.0;
```

Date and time

Keywords: DT or DATE_AND_TIME

Value range: yyyy-mm-dd-hh:mm:ss

Declaration of a variable (example):

```
VAR
    dtExample : DT;
END_VAR
```

Declaration of a constant (example):

```
VAR CONSTANT
    dtConstExample : DT := dt#2017-12-31-23:59:59;
END_VAR
```

Assignment (example):

```
dtExample := dt#2017-01-01-08:00:00;
```

4.1 Fields

All data types can also be declared as a one-dimensional or two-dimensional field (array) (see standard DIN IEC 61131-3). The following sections use the data type REAL as an example.

One-dimensional field

Declaration of variables (example):

```
VAR
    rArrayExample : ARRAY [0..2] OF REAL; (* field with 3 variables *)
END_VAR
```

Declaration of constants (example):

```
VAR CONSTANT
    rArrayConstExample : ARRAY [0..2] OF REAL := [0.0, 0.1, 0.2]; (* field with 3 constants *)
END_VAR
```

Assignment (example):

```
rArrayExample[0] := 0.0;
rArrayExample[1] := 0.1;
rArrayExample[2] := 0.2;
```

Two-dimensional field

Declaration of variables (example):

```
VAR
    rArrayExample : ARRAY [0..2, 0..1] OF REAL; (* field with 3 x 2 variables *)
END_VAR
```

Declaration of constants (example):

```
VAR CONSTANT
    rArrayConstExample : ARRAY [0..2, 0..1] OF REAL :=
    [0.0, 0.1, 0.2,
    1.0, 1.1, 1.2]; (* field with 3 x 2 constants *)
END_VAR
```

Assignment (example):

```
rArrayExample[0,0] := 0.0;
rArrayExample[1,0] := 0.1;
rArrayExample[2,0] := 0.2;
rArrayExample[0,1] := 1.0;
rArrayExample[1,1] := 1.1;
rArrayExample[2,1] := 1.2;
```

4 Data types

All operators supported by the ST module are shown in the following table. The order in the table depends on the ranking of the operators, starting with the highest rank.

Operation	Symbol	Admissible data types	Example
Brackets	(expression)		a := 3.0 * (b - 1.0);
Function	Identifier (argument list)		i := MIN (3, j);
Negation	-	REAL	a := -a;
Complement	NOT	BOOL, UINT, UDINT[a]	a := NOT b;
Multiplication	*	UINT, UDINT, REAL	i := 5 * j;
Division	/		a := 5.0 / b;
Modulo	MOD	UINT, UDINT	j := i MOD 10;
Addition	+	UINT, UDINT, REAL	i := 5 + j;
Subtraction	-		a := b - 5.0E20;
Comparison	<, >, <=, >=	UINT, UDINT, REAL, DATE_AND_TIME	bExample := 5 <= j;
Equality	=	BOOL, UINT, UDINT, REAL, DATE_AND_TIME	bExample := 5 = i;
Inequality	<>		bExample := 5 <> j;
AND	& or AND	BOOL, UINT, UDINT[a]	bExample := x AND y;
Exclusive OR	XOR	BOOL, UINT, UDINT[a]	bExample := x XOR y;
OR	OR	BOOL, UINT, UDINT[a]	bExample := x OR y;

[a] Boolean variables are logically linked, integer variables are linked bit-by-bit.

5 Operators

A selection instruction executes an instruction or group of instructions based on a specified condition.

IF instruction

The IF instruction specifies that a group of instructions is only executed if the corresponding Boolean expression returns the value TRUE. If the condition is incorrect, either no instruction may be executed or the instruction group following the keyword ELSE (or the keyword ELSIF, if its associated Boolean condition is true) must be executed.

Keywords:

IF, THEN, ELSIF, ELSE, END_IF

Example:

```
VAR
    i : UINT;
END_VAR

IF reset_flag THEN
    bool_out01 := false; (* after the reset, the outputs bool_out01 and *)
    bool_out02 := false; (* bool_out02 are reset first and *)
    i := 1; (* i set to 1 *)
END_IF;

if i>0 and i<=100 then
    bool_out01 := true; (* in run through 1 to 100, the output bool_out01 is set ... *)
    bool_out02 := false; (* and bool_out02 reset *)
elsif i>100 and i<=200 then
    bool_out01 := false; (* in run through 101 to 200, the output bool_out01 is reset ... *)
    bool_out02 := false; (* and bool_out02 set *)
else
    i := 1; (* in run through 201, i is set to 1 and ... *)
    bool_out01 := false; (* the outputs bool_out01 and ... *)
    bool_out02 := false; (* bool_out02 are reset; it starts again from the beginning *)
end_if;

i := i + 1; (* i is increased by 1 *)
```

6 Selection instructions

CASE instruction

The CASE instruction combines several conditional instructions. It consists of an expression that contains an integer variable and a list of instruction groups. Each group is identified by a mark consisting of one or more integers (separated by commas). The value of the variable determines the mark and thus the instruction group to be executed. The keyword ELSE can also be used as an option. If no value applies, the instruction group following the keyword ELSE is executed.

Keywords:

CASE, OF, ELSE, END_CASE

Example:

```
CASE i OF (* variable i must be UINT or UDINT *)
    1: b1 := TRUE;
    2, 3, 4: b2 := TRUE;
ELSE
    b1 := FALSE;
    b2 := FALSE;
END_CASE;
```

With repeat instructions (iterations), instructions and groups of instructions are executed repeatedly.

There are three types of repeat instructions:

- FOR
- WHILE
- REPEAT UNTIL

EXIT

With EXIT, a repeat instruction is terminated before the end condition of the iteration is fulfilled. If EXIT is executed within a nested repetition construct, the innermost loop is exited within the EXIT.

NOTE!

For repeat instructions, you must ensure that there are no endless loops or very long running loops. The runtime is monitored.

FOR instruction

The FOR instruction is used if the number of repeats is fixed.

Keywords:

FOR, TO, BY, DO, END_FOR

Examples:

```
FOR i := 10 TO 100 BY 10 DO (* control variable must be UINT *)
    j := j + i; (* i runs from 10 to 100 in steps of 10 (10, 20, 30, …, 100)  *)
END_FOR;

FOR i := 1 TO 5 DO (* without specification of "BY x", the ... *)
    a := a + b; (* control variable increases by 1 in each run through (1, 2, 3, 4, 5) *)
    a := a * 3.0;
END_FOR;
```

WHILE instruction

The WHILE instruction causes a group of instructions to be repeatedly executed until the associated Boolean expression is incorrect (FALSE, untrue). If the Boolean expression is incorrect from the beginning, the group of instructions is not executed at all.

Keywords:

WHILE, DO, END_WHILE

Example:

```
WHILE j < 100 DO
    j := j + 2;
END_WHILE;
```

7 Repeat instructions

REPEAT instruction

The REPEAT instruction causes a group of instructions up to the keyword UNTIL to be repeatedly executed until the associated Boolean condition is true (TRUE).

Keywords:

REPEAT, UNTIL, END_REPEAT

Example:

```
REPEAT
    i := i - 2;
    a := a + 2.0;
UNTIL i = 0 END_REPEAT;
```

The ST module supports the following functions:

- Type conversion
- Arithmetic functions
- Numerical functions
- Bit sequence functions (shift functions)
- Logical functions
- Selection (statistics)
- Comparison
- Date and time
- Specific functions (not part of the standard/norm)

8.1 Type conversion

Admissible data types

Argument: UINT, UDINT

Result: BOOL, UINT, UDINT, REAL

INT_TO_REAL

Converts an INTEGER into a REAL number.

Example:

a := INT_TO_REAL(10); (* a := 10.0 *)

INT_TO_DINT

Converts an INTEGER into a DOUBLE INTEGER.

Example:

a := INT_TO_DINT(10); (* a := 10 *)

INT_TO_BOOL

Converts an INTEGER into a BOOL value.

The result is FALSE if the argument is 0. In all other cases, the result is TRUE.

Examples:

a := INT_TO_BOOL(0); (* a = FALSE *)

b := INT_TO_BOOL(1); (* b = TRUE *)

c := INT_TO_BOOL(8); (* c = TRUE *)

DINT_TO_REAL

Converts a DOUBLE INTEGER into a REAL number.

Examples:

a := DINT_TO_REAL(100000); (* a = 100000.0 *)

8 Functions

DINT_TO_INT

Converts a DOUBLE INTEGER into an INTEGER.

Only the lower two bytes are evaluated (bit 0 to bit 15).

Examples:

a := DINT_TO_INT(1); (* a = 1; 1 = 0000 0000 0000 0001 = 0x0001 *)

b := DINT_TO_INT(65535); (* b = 65535; 65535 = 1111 1111 1111 1111 = 0xFFFF *)

c := DINT_TO_INT(65536); (* c = 0; 65536 = 0001 0000 0000 0000 0000 = 0x10000 *)

d := DINT_TO_INT(65537); (* d = 1, 65537 = 0001 0000 0000 0000 0001 =0x10001 *)

DINT_TO_BOOL

Converts a DOUBLE INTEGER into a BOOL value.

The result is FALSE if the argument is 0. In all other cases, the result is TRUE.

Examples:

a := DINT_TO_BOOL(0); (* a = FALSE: *)

b := DINT_TO_BOOL(1); (* b = TRUE *)

c := DINT_TO_BOOL(100000); (* c = TRUE *)

REAL_TO_INT

Converts a REAL number with rounding into an INTEGER.

Examples:

a := REAL_TO_INT(1.378); (* a = 1 *)

b := REAL_TO_INT(1.897); (* b = 2 *)

REAL_TO_DINT

Converts a REAL number with rounding into a DOUBLE INTEGER.

Examples:

a := REAL_TO_DINT(100000.378); (* a = 100000 *)

b := REAL_TO_DINT(100000.897); (* b = 100001 *)

8.2 Arithmetic functions

Admissible data types

Argument: UINT, UDINT, REAL (only REAL for negation, only UINT or UDINT for modulo)

Result: UINT, UDINT, REAL

Addition

Addition is an expandable function. The sum of the arguments is returned as the result.

Example:

OUT := IN1 + IN2 + ... INn;

Subtraction

The second argument is subtracted from the first argument as a result.

Example:

OUT := IN1 - IN2;

Multiplication

Multiplication is an expandable function. The result is obtained by multiplying the arguments.

Example:

OUT := IN1 * IN2 * ...INn;

Division

The quotient of the two arguments is returned as the result.

The result of the division of integers (UINT or UDINT) is an integer with truncation of the decimal places (example: 7/3 = 2).

Example:

OUT := IN1 / IN2;

Negation

REAL numbers can be negated.

Example:

OUT := -IN;

Modulo

The arguments of the modulo function (%) must be integers (UINT or UDINT). The result of the modulo function is the same as processing the following instruction:

```
IF (IN2 = 0) THEN OUT := 0;
ELSE OUT := IN1 - (IN1 / IN2) * IN2;
END_IF
```

The result of the division of integers (UINT or UDINT) in the ELSE branch of the above IF selection instruction is an integer with truncation of the decimal places.

Example:

```
IN1 := 17;
IN2 := 3;
OUT := IN1 % IN2; (* OUT := 2 *)
```

8.3 Numerical functions

Admissible data types

Argument: REAL

Result: REAL

ABS (IN)

Returns the absolute value of IN.

Example:

```
IN := -2.3;
OUT := ABS(IN); (* OUT = 2.3 *)
```

SQRT (IN)

Returns the square root of IN.

If IN is negative, the error value 5.0E+37 is returned.

Example:

```
IN := 4.0;
OUT := SQRT(IN); (* OUT = 2.0 *)
```

LN (IN)

Returns the natural logarithm of IN.

Example:

```
IN := 2.718282; (* number e (rounded) *)
OUT := LN(IN); (* OUT = 1.0 (rounded) *)
```

LOG (IN)

Returns the base 10 logarithm of IN.

Example:

```
IN := 100.0;
OUT := LOG(IN); (* OUT = 2.0 *)
```

EXP (IN)

Returns the natural exponential function of IN.

Example:

```
IN := 2.0;
OUT := EXP(IN); (* OUT = 7.389056 *)
```

SIN (IN)

Returns the sine of IN (radian).

Example:

```
IN := 1.5708; (* corresponds to 90° *)
OUT := SIN(IN); (* OUT = 1.0 *)
```

COS (IN)

Returns the cosine of IN (radian).

Example:

```
IN := 0.0; (* corresponds to 0° *)
OUT := COS(IN); (* OUT = 1.0 *)
```

TAN (IN)

Returns the tangent of IN (radian).

Example:

```
IN := 0.7854; (* corresponds to 45° *)
OUT := TAN(IN); (* OUT = 1.0 *)
```

ASIN (IN)

Returns the arc sine of IN (radian).

Example:

```
IN := 1.0;
OUT := ASIN(IN); (* OUT = 1.5708; corresponds to 90° *)
```

ACOS (IN)

Returns the arc cosine of IN (radian).

Example:

```
IN := 1.0;
OUT := ASIN(IN); (* OUT = 0.0; corresponds to 0° *)
```

ATAN (IN)

Returns the arc tangent of IN (radian).

Example:

```
IN := 1.0;
OUT := ATAN(IN); (* OUT = 0.7854; corresponds to 45° *)
```

8.4 Bit sequence functions

Admissible data types

Argument: UINT, UDINT

Result: UINT, UDINT

SHL (IN, n)

Moves the bit sequence of the IN argument to the left by n bits. Subsequent digits are filled with 0 from the right.

Example:

```
IN := 255; (* bit sequence: 0000 0000 1111 1111 *)
OUT := SHL(IN, 4); (* OUT = 4080; bit sequence: 0000 1111 1111 0000 *)
```

SHR (IN, n)

Moves the bit sequence of the IN argument to the right by n bits. Subsequent digits are filled with 0 from the left.

Example:

```
IN := 255; (* bit sequence: 0000 0000 1111 1111 *)
OUT := SHR(IN, 4); (* OUT = 15; bit sequence: 0000 0000 0000 1111 *)
```

ROL (IN, n)

Rotates the bit sequence of the IN argument to the left by n bits in a circle.

Example:

```
IN := 43690; (* bit sequence: 1010 1010 1010 1010 *)
OUT := ROL(IN, 1); (* OUT = 21845; bit sequence: 0101 0101 0101 0101 *)
```

ROR (IN, n)

Rotates the bit sequence of the IN argument to the right by n bits in a circle.

Example:

```
IN := 21845; (* bit sequence: 0101 0101 0101 0101 *)
OUT := ROR(IN, 1); (* OUT = 43690; bit sequence: 1010 1010 1010 1010 *)
```

8.5 Logical functions

Boolean variables are linked logically, UINT and UDINT variables are linked bit-by-bit.

If there are more than two parameters, each pair is always linked from left to right.

Admissible data types

Operand: BOOL, UINT, UDINT

Result: BOOL, UINT, UDINT

AND

AND link

Examples:

Logical link:

```
x := TRUE;
y := FALSE;
z := x AND y; (* z = FALSE *)
```

Bit-by-bit link:

```
a := 5;. . . . . 0101 = 5
b := 6;. . . . . .0110 = 6
      . . . . . ._____
c := a AND b; (* c = 4 *) . . . . . 0100 = 4
```

OR

OR link

Examples:

Logical link:

```
x := TRUE;
y := FALSE;
z := x OR y; (* z = TRUE *)
```

Bit-by-bit link:

```
a := 5;. . . . . 0101 = 5
b := 6;. . . . . .0110 = 6
      . . . . . ._____
c := a OR b; (* c = 7 *) . . . . . .0111 = 7
```

XOR

Exclusive OR link

Examples:

Logical link:

```
x := TRUE;
y := FALSE;
z := x XOR y; (* z = TRUE *)
```

Bit-by-bit link:

```
a := 5;. . . . . 0101 = 5
b := 6;. . . . . .0110 = 6
      . . . . . ._____
c := a XOR b; (* c = 3 *). . . . . .0011 = 3
```

NOT

Inversion (form complement)

Examples:

```
IN := 1;
OUT := NOT IN; (* OUT = 0 *)
IN := 43690; (* IN = 1010 1010 1010 1010 *)
OUT := NOT IN; (* OUT = 21845; OUT = 0101 0101 0101 0101 *)
```

8.6 Selection

Admissible data types

Argument: UINT, UDINT, REAL (with SEL, additionally BOOL for the selector)

Result: UINT, UDINT, REAL

MAX

Returns the biggest argument as a result.

Example:

OUT := MAX(8, 9, 10, 11, 12); (* OUT := 12 *)

MIN

Returns the smallest argument as a result.

Example:

OUT := MIN(8, 9, 10, 11, 12); (* OUT := 8 *)

LIMIT

Checks whether a value (IN) is within a certain range (MIN, MAX). If the value is within the range, the value itself is returned. If the value is below the range, the lower range limit is returned. If the value is above the range, the upper range limit is returned.

Examples:

OUT := LIMIT(IN, MIN, MAX);

OUT := LIMIT(15, 10, 20); (* OUT = 15 *)

OUT := LIMIT(5, 10, 20); (* OUT = 10 *)

OUT := LIMIT(25, 10, 20); (* OUT = 20 *)

SEL

Selects one of two values (IN0, IN1) depending on a Boolean variable (G). The data type of the values IN0 and IN1 must be identical.

Examples:

OUT := SEL(G, IN0, IN1);

OUT := SEL(G, 10, 20); (* G = FALSE: OUT = 10 *)

OUT := SEL(G, 10, 20); (* G = TRUE: OUT = 20 *)

8.7 Comparison

Admissible data types

Argument with LT, LE, GT, GE: UINT, UDINT, REAL, DT

Argument with EQ and NE: BOOL, UINT, UDINT, REAL, DT

Result: BOOL

< (LT)

Compares arguments on whether one is smaller than another.

Examples:

bOUT := IN1 < IN2; (* bOUT := TRUE, if IN1 is smaller than IN2 *)

bOUT := (IN1 < IN2) & (IN2 > IN3) & ... & (INn-1 > INn);

<= (LE)

Compares arguments on whether one is smaller than or equal to another.

Examples:

bOUT := IN1 <= IN2; (* bOUT := TRUE, if IN1 is smaller than or equal to IN2 *)

bOUT := (IN1 <= IN2) & (IN2 <= IN3) & ... & (INn-1 <= INn);

> (GT)

Compares arguments on whether one is larger than another.

Examples:

bOUT := IN1 > IN2; (* bOUT := TRUE, if IN1 is larger than IN2 *)

bOUT := (IN1 > IN2) & (IN2 > IN3) & ... & (INn-1 > INn);

>= (GE)

Compares arguments on whether one is larger than or equal to another.

Example:

bOUT := IN1 >= IN2; (* bOUT := TRUE, if IN1 is larger than or equal to IN2 *)

= (EQ)

Compares arguments on whether they are equal.

Example:

bOUT := IN1 = IN2; (* bOUT := TRUE, if IN1 is equal to IN2 *)

<> (NE)

Compares arguments on whether they are unequal.

Example:

bOUT := IN1 <> IN2; (* bOUT := TRUE, if IN1 is unequal to IN2 *)

8.8 Date and time

These are specific functions that are not part of the DIN IEC 61131-3 standard.

For example, these functions can be used to query the input variable rtc.cdt, which returns the time since the device was switched on (date and time start each time the device is switched on from 01.01.1970 00:00:00:00).

Admissible data types

Argument: DT (or DATE_AND_TIME)

Result: UINT

GET_YEAR

Returns the year (with century).

Examples:

iOUT := GET_YEAR(dt#2017-03-20-17:45:12); (* iOUT = 2017 *)

GET_MONTH

Returns the month.

Example:

iOUT := GET_MONTH(dt#2017-03-20-17:45:12); (* iOUT = 03 *)

GET_DAY

Returns the day.

Example:

iOUT := GET_DAY(dt#2017-03-20-17:45:12); (* iOUT = 20 *)

GET_HOUR

Returns the hour.

Example:

iOUT := GET_HOUR(dt#2017-03-20-17:45:12); (* iOUT = 17 *)

GET_MINUTE

Returns the minute.

Example:

iOUT := GET_MINUTE(dt#2017-03-20-17:45:12); (* iOUT = 45 *)

GET_SECOND

Returns the second.

Example:

iOUT := GET_SECOND(dt#2017-03-20-17:45:12); (* iOUT = 12 *)

iOUT := GET_SECOND(rtc_cdt); (* iOUT = 59, if rtc_cdt = 1970-01-01 00:00:59 *)

8.9 Other functions

These are specific functions that are not part of the DIN IEC 61131-3 standard.

IS_VALID

Checks the validity of a REAL number. The result is a Boolean value (valid = TRUE, invalid = FALSE).

Values ≥ 1.00E+37 (general error value) are considered as invalid, as are the values "Infinity" (INF) and "No number" (NaN).

Examples:

```
bOUT := IS_VALID(1.2); (* bOUT =  TRUE *)
```

SET_DEFAULT

Sets a REAL or a BOOL number to the default value.

Default value REAL: 5.0E+37 (math error value)
Default value BOOL: FALSE

Example:

```
SET_DEFAULT(bOUT); (* bOUT =  FALSE *)
SET_DEFAULT(rOUT); (* rOUT = 5.0E+37 *)
```

Instance

A function module – called a function block in the ST editor – is a program organizational unit that returns one or more values when executed (see also standard DIN IEC 61131-3). A function module can exist in several instances (copies). Each of these instances has a corresponding identifier. The instances of a function module are independent of each other.

If a function module called TON exists, for example in 4 instances, the individual instances are addressed with TON01, TON02, TON03, TON04.

The ST module includes the following function modules:

Identifier	Instances	Function
CTUD	4	Software up/down counter
TP	4	Impulse generator
TON	4	Switch-on delay
TOF	4	Switch-off delay
R_TRIG	4	Rising edge detection (0 -> 1)
F_TRIG	4	Falling edge detection (1 -> 0)
SR	4	Bistable function module (set priority)
RS	4	Bistable function module (reset priority)
SET_CP	1	Specify controller parameter

Parameters

One or more parameters (inputs) are passed to the function modules. The parameters follow in round brackets after the specification of the instance. The sequence of the parameters is fixed. Like any instruction, the call for a function module is completed with a ";".

Example:

TON01(TRUE,5,1);

Inputs/outputs

The statuses of the inputs and outputs of all instances of the function modules remain constant until the next call of the instance of the function module. The outputs can be queried and used in any operations.

Access to outputs

Outputs (structure elements) of function modules are accessed as usual in the programming language Pascal.

Example:

Queries of the output ET of the function module TON with instance 01:

OUT := TON01.ET;

9 Function modules

9.1 Software up/down counter

The counter provides the following functions:

- Counting up
- Counting down
- Reset the counter to 0
- Set an upper counter value
- Query in case the upper counter value is exceeded
- Query of counter status 0
- Query counter value

The counter counts the positive edges at the inputs CU (counting upwards) and CD (counting downwards).

Call

CTUD<instance> (CU, CD, R, LD, PV);

Inputs

The parameters R, LD, CU and CD are sorted in descending order in the table according to their priority.

Parameter	Data type	Description
<Instance>		01 to 04 (specify instance with two digits)
R	BOOL	TRUE = set counter status to 0
LD	BOOL	TRUE = set upper counter value
CU	BOOL	Positive edge = count up
CD	BOOL	Positive edge = count down
PV	UINT	Upper counter value

Outputs

Parameter	Data type	Query; (* description *)
CV	UINT	OUT := CTUD<instance>.CV; (* OUT = counter status *)
QU	BOOL	OUT := CTUD<instance>.QU; (* OUT = TRUE, if counter status > upper counter value *)
QD	BOOL	OUT := CTUD<instance>.QD; (* OUT = TRUE, if counter status <= 0 *)

Comment

To set the upper counter value with LD, R must not be TRUE.

The values of the inputs/outputs are not retained via power off.

Examples

CTUD01 (IN, FALSE, FALSE, FALSE, 100); (* upwards counting of the positive edges of IN *)

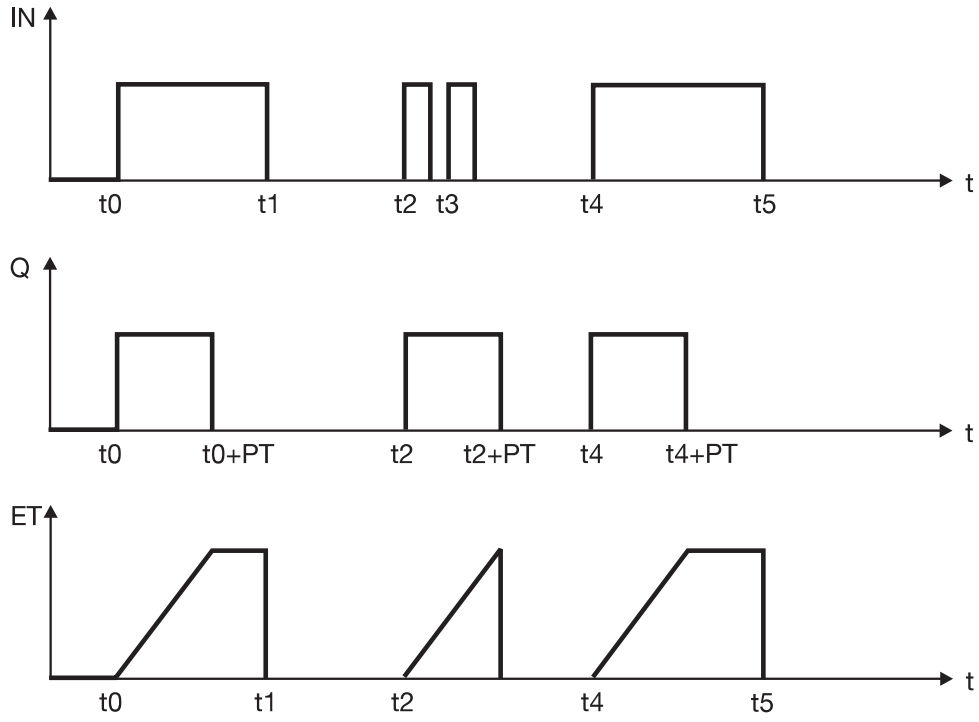
CTUD01 (FALSE, IN, FALSE, FALSE, 100); (* downwards counting of the positive edges of IN *)

CTUD01 (IN, FALSE, TRUE, FALSE, 100); (* set counter status to 0 *)

CTUD01 (IN, FALSE, FALSE, TRUE, 100); (* set upper counter value: 100 *)

9.2 Impulse generator

The following diagram illustrates the operating principle of the impulse generator:



Call

TP<instance> (IN, PT, TimeBase);

Inputs

Parameter	Data type	Description
<Instance>		01 to 04 (specify instance with two digits)
IN	BOOL	Positive edge at IN sets Q for the time PT to TRUE
PT	UINT	Time (unit: TimeBase), for Q = TRUE, IN = TRUE
TimeBase	UINT	Unit of PT: 1 = ms 2 = s 3 = min

Outputs

Parameter	Data type	Query; (* description *)
Q	BOOL	OUT := TP<instance>.Q; (* OUT = TRUE for the time PT, if IN := 0 -> 1 *)
ET	UINT	OUT := TP<instance>.ET; (* OUT = time passed since IN = TRUE until Q = FALSE (time passed since the start of the active impulse phase) *)

Comment

The TimeBase parameter is an extension to DIN IEC 61131-3. This extension does not constitute a limitation to the standard. According to the standard, the time limit is "implementation-dependent".

The values of the inputs/outputs are not retained when the power is turned off.

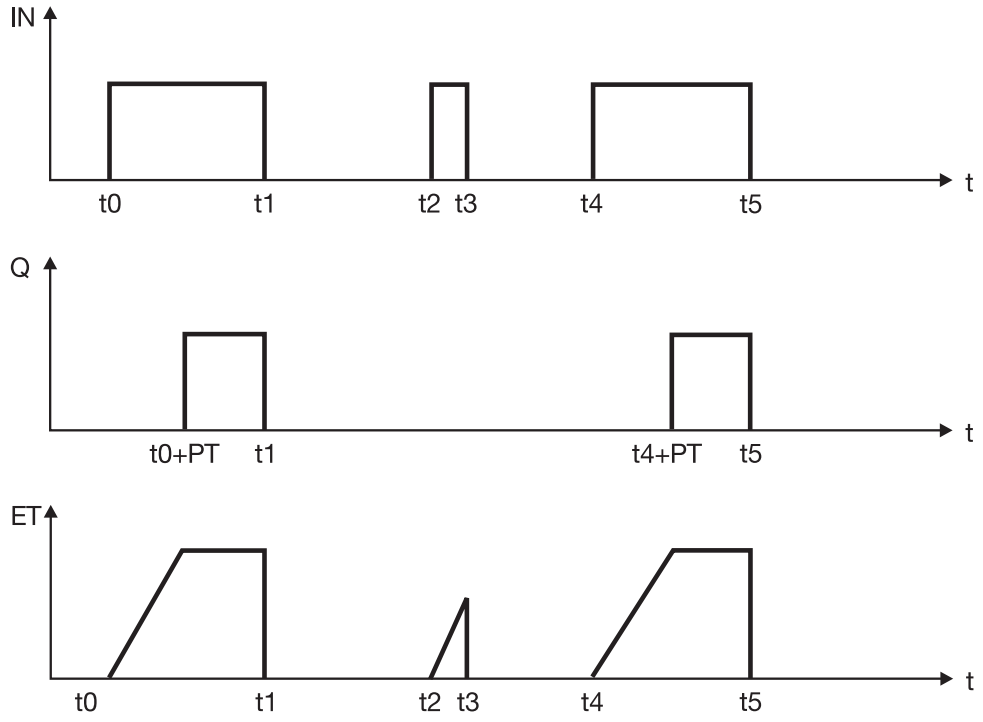
The time resolution of the impulse length is 150 ms (device cycle time).

Example

```
TP01(IN, 5, 3);
(* Positive edge at IN sets TP01.Q for approx. 5 minutes to TRUE (note time resolution).
TP01.ET liefert die vergangene Zeit in Minuten seit Beginn der aktiven Pulsphase. *)
```

9.3 Switch-on delay

The following diagram illustrates the operating principle of the switch-on delay:



Call

TON<instance> (IN, PT, TimeBase);

Inputs

Parameter	Data type	Description
<Instance>		01 to 04 (specify instance with two digits)
IN	BOOL	IN = TRUE sets Q = TRUE, **after** the time PT is expired
PT	UINT	Delay time (unit: TimeBase)
TimeBase	UINT	Unit of PT: 1 = ms 2 = s 3 = min

Outputs

Parameter	Data type	Query; (* description *)
Q	BOOL	OUT := TON<instance>.Q; (* OUT = TRUE, if IN = TRUE and PT expired *)
ET	UINT	OUT := TON<instance>.ET; (* OUT = time passed since IN = TRUE until Q = TRUE or until IN = FALSE *)

Comment

The TimeBase parameter is an extension to DIN IEC 61131-3. This extension does not constitute a limitation to the standard. According to the standard, the time limit is "implementation-dependent".

The values of the inputs/outputs are not retained when the power is turned off.

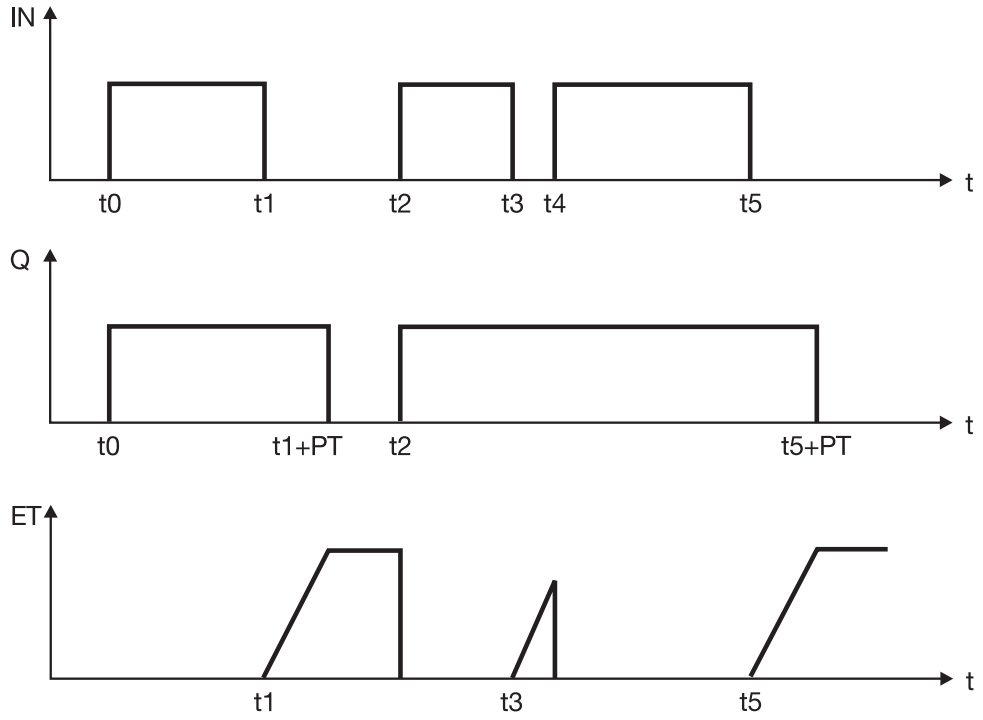
The time resolution of the delay time is 150 ms (device cycle time).

Example

```
TON01(IN, 6, 2);
(* IN = TRUE sets TON01.Q to TRUE after 6 seconds.
TON01.Q bleibt solange TRUE, wie IN = TRUE ist.
TON01.ET liefert die vergangene Zeit in Sekunden von IN = TRUE bis TON01.Q = TRUE (max. 6 Sekunden) oder bis IN = FALSE. *)
```

9.4 Switch-off delay

The following diagram illustrates the operating principle of the switch-off delay:



Call

TOF<instance> (IN, PT, TimeBase);

Inputs

Parameter	Data type	Description
<Instance>		01 to 04 (specify instance with two digits)
IN	BOOL	IN = TRUE sets Q = TRUE IN = FALSE sets Q around the time PT delayed to FALSE
PT	UINT	Delay time (unit: TimeBase)
TimeBase	UINT	Unit of PT: 1 = ms 2 = s 3 = min

Outputs

Parameter	Data type	Query; (* description *)
Q	BOOL	OUT := TOF<instance>.Q; (* OUT = TRUE, if IN = TRUE or if IN = FALSE and PT has not yet expired *)
ET	UINT	OUT := TOF<instance>.ET; (* OUT = time passed since IN = FALSE until Q = FALSE or until IN = TRUE *)

Comment

The TimeBase parameter is an extension to DIN IEC 61131-3. This extension does not constitute a limitation to the standard. According to the standard, the time limit is "implementation-dependent".

The values of the inputs/outputs are not retained when the power is turned off.

The time resolution of the delay time is 150 ms (device cycle time).

Example

```
TOF01(IN, 450, 1);
(* IN = TRUE sets TOF01.Q to TRUE.
After IN = FALSE, TOF01.Q remains TRUE until the time PT = 450  ms has expired.
TOF01.ET returns the time passed in milliseconds from IN = FALSE until TOF01.Q = FALSE (max. 450 ms) or until IN = TRUE. *)
```

9.5 Rising edge detection

This function module detects a rising edge (transition 0 -> 1) .

Call

R_TRIG<instance> (CLK);

Input

Parameter	Data type	Description
<Instance>		01 to 04 (specify instance with two digits)
CLK	BOOL	Due to a rising edge (0 -> 1) at CLK, Q = TRUE

Output

Parameter	Data type	Query
Q	BOOL	OUT := R_TRIG<instance>.Q;

Comment

The operating principle of the function module corresponds to the following program code:

VAR CLK : BOOL; END_VAR

VAR Q : BOOL; END_VAR

VAR M : BOOL := FALSE; END_VAR

Q := CLK AND NOT M;

M := CLK;

The Q output remains at its Boolean value from one call until the next. It follows the transition of the input signal (CLK) from "0" to "1" and returns to "0" the next time it is called.

Example

```
IF reset_flag THEN
bool_out01 := FALSE;
END_IF;
R_TRIG01 (bool_in01); (* detection of a rising edge at input bool_in01 *)
bool_out01 := R_TRIG01.Q; (* short impulse at output bool_out01 for rising edge at input *)
```

9 Function modules

9.6 Falling edge detection

This function module detects a falling edge (transition 1 -> 0) .

Call

F_TRIG<instance> (CLK);

Input

Parameter	Data type	Description
<Instance>		01 to 04 (specify instance with two digits)
CLK	BOOL	Due to a falling edge (1 -> 0) at CLK, Q = TRUE

Output

Parameter	Data type	Query
Q	BOOL	OUT := F_TRIG<instance>.Q;

Comment

The operating principle of the function module corresponds to the following program code:

VAR CLK : BOOL; END_VAR

VAR Q : BOOL; END_VAR

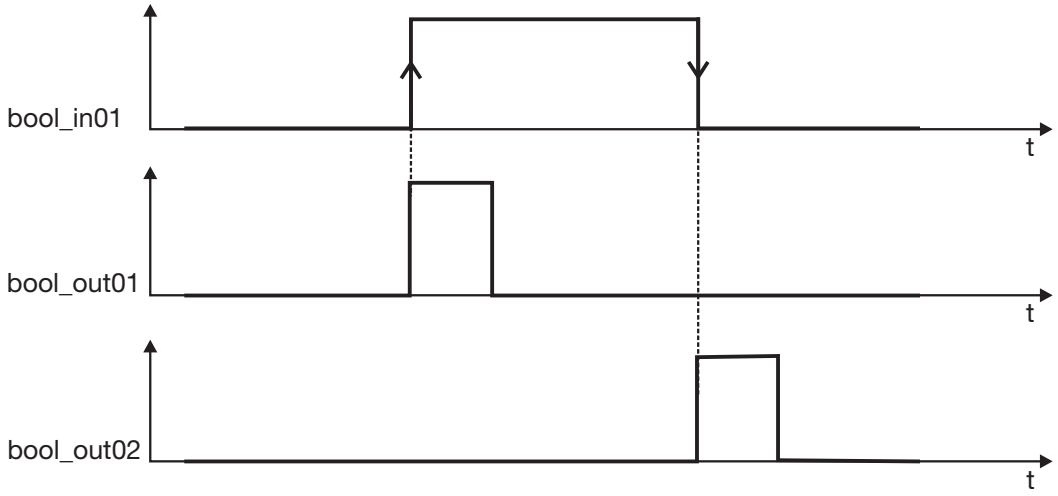
VAR M : BOOL := TRUE; END_VAR

Q := NOT CLK AND NOT M;

M := NOT CLK;

The Q output remains at its Boolean value from one call until the next. It follows the transition of the input signal (CLK) from "1" to "0" and returns to "0" the next time it is called.

Example with R_TRIG and F_TRIG




```
IF reset_flag THEN
bool_out01 := FALSE;
bool_out02 := FALSE;
END IF;

R_TRIG01 (bool_in01); (* detection of a rising edge at input bool_in01 *)
bool_out01 := R_TRIG01.Q; (* short impulse at output bool_out01 for rising edge at input *)

F_TRIG01 (bool_in01); (* detection of a falling edge at input bool_in01 *)
bool_out02 := F_TRIG01.Q; (* short impulse at output bool_out02 for falling edge at input *)
```

9.7 Bistable function SR

Bistable function (set priority) with lock

Call

SR<instance> (S1, R);

Inputs

Parameter	Data type	Description
<Instance>		01 to 04 (specify instance with two digits)
S1	BOOL	S1 = TRUE sets Q1 = TRUE
R	BOOL	R = TRUE sets Q1 = FALSE, if S1 = FALSE

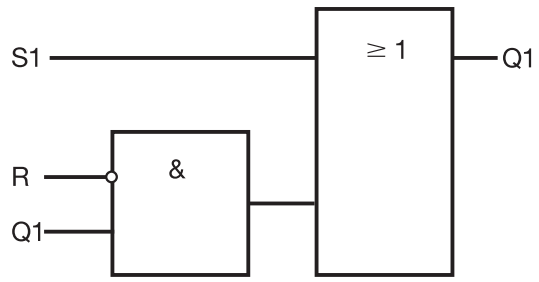
Outputs

Parameter	Data type	Query
Q1	BOOL	OUT := SR<instance>.Q1;

Comment

Output Q1 remains TRUE after being set by S1 until it is reset with R.
If S1 and R are set simultaneously (TRUE), output Q1 is also set (TRUE).

Logic function



Example

SR1 (bool_in01, bool_in02); (* set with input bool_in01, reset with input bool_in02 *)
bool_out01 := SR1.Q1; (* the status is output via output bool_out01. *)

9.8 Bistable function RS

Bistable function (reset priority) with lock

Call

RS<instance> (S, R1);

Inputs

Parameter	Data type	Description
<Instance>		01 to 04 (specify instance with two digits)
R1	BOOL	R1 = TRUE sets Q1 = FALSE
S	BOOL	S = TRUE sets Q1 = TRUE, if R1 = FALSE

Outputs

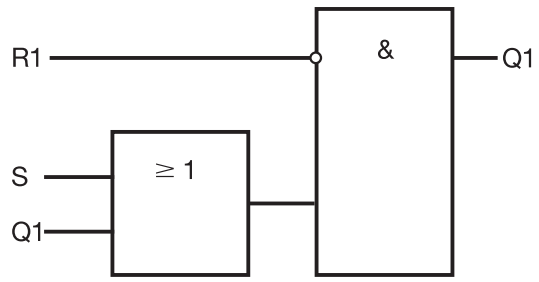
Parameter	Data type	Query
Q1	BOOL	OUT := RS<instance>.Q1;

Comment

After resetting with R1, output Q1 remains FALSE until it is set with S.

If R1 and S are set simultaneously (TRUE), output Q1 is reset (FALSE).

Logic function



Example

RS1 (bool_in01, bool_in02); (* reset with input bool_in02, set with input bool_in01 *)

bool_out02 := RS1.Q1; (* the status is output via output bool_out02. *)

9.9 Set controller parameter

The controller parameters of the **second** parameter set can be changed with this function module (specific function, not part of the standard DIN IEC 61131-3).

Call

Set_CP (Set, Key, Value);

Inputs

Parameter	Data type	Description
Controller	UINT	Controller no.: 0 = controller 1; 1 = controller 2; ... Set = 0, if the device only has one controller (controller channel).
Parameter	UINT	Parameter: 0 = Xp1 (proportional band 1) 1 = Xp2 (proportional band 2) 2 = Tv1 (derivative time 1) 3 = Tv2 (derivative time 2) 4 = Tn1 (reset time 1) 5 = Tn2 (reset time 2) 6 = Cy1 (cycle time 1) 7 = Cy2 (cycle time 2) 8 = Y1 (max. output value limit) 9 = Y2 (min. output value limit)
Value	REAL	Value of the parameter concerned

Outputs

This function module has no outputs.

Comment

The values of the parameters are checked before being adopted in the controller. If the value is outside the admissible range of values, the adoption is refused and an error value is returned in the variable ext_error (can be evaluated in the ST code). If the value is valid, the new value is used from this point on.

The changed values are not stored in the controller configuration. After a reset (power on) the stored values are used again as controller parameters.

To change several parameters, the function module must be called more than once (see example: two calls).

Example

```
VAR
    controller : UINT; (* controller no. *)
    para1 : UINT; (* Parameter 1 *)
    value1 : REAL; (* value of parameter 1 *)
    para2 : UINT; (* Parameter 2 *)
    value2 : REAL; (* value of parameter 2 *)
END_VAR
SET_CP (controller, para1, value1);
SET_CP (controller, para2, value2);
```

JUMO GmbH & Co. KG
Street address:
Moritz-Juchheim-Straße 1
36039 Fulda, Germany
Delivery address:
Mackenrodtstraße 14
36039 Fulda, Germany
Postal address:
36035 Fulda, Germany
Phone: +49 661 6003-0
Fax: +49 661 6003-607
Email: mail@jumo.net
Internet: www.jumo.net

JUMO Instrument Co. Ltd.
JUMO House
Temple Bank, Riverway
Harlow, Essex, CM20 2DY, UK
Phone: +44 1279 63 55 33
Fax: +44 1279 62 50 29
Email: sales@jumo.co.uk
Internet: www.jumo.co.uk

JUMO Process Control, Inc.
6733 Myers Road
East Syracuse, NY 13057, USA
Phone: +1 315 437 5866
Fax: +1 315 437 5860
Email: info.us@jumo.net
Internet: www.jumousa.com