FAQ COMBIVIS studio 6

Structured Text Editor FAQ No.0008

Part	Version	Revision	Date	Status
en	6.2.3.0	001	2019-01-01	Released

Content

KEB Automation KG
Südstraße 38
32683 Barntrup, Germany
E-mail: info@keb.de
Tel. +49 5263 401-0
Fax +49 5263 401-116
www.keb.de

District Court Lemgo HRA 5649
DUNS-No. 314108728
VAT-No. DE309087075
Bank Details: Sparkasse Paderborn-Detmold
IBAN DE 19 4765 0130 0000 0060 07
BIC WELADE3L

General Partner:
Vittorio Tavella
KEB Verwaltungs-GmbH, Barntrup
District Court: Lemgo HRB 8965
Directors: Curt Bauer CMO,
Ralf Lutter COO, Vittorio Tavella CFO,
Wolfgang Wiele CTO

Introduction

This document gives some basic information about the structured text (ST) programming language from IEC 61131-3 and the ST-Editor in COMBIVIS studio 6.

Variable declaration

In ST-language you can define variables in different ways to make them available in just one program or function or make them global so you can use them in all parts of the project.
VAR: Local variable in one POU.
VAR_GLOBAL: Global variable available in complete project.

CONSTANT: Variable with constant value.
RETAIN: Variable is stored in the EEPROM.
PERSISTENT: Value of variable remains on new program download.

VAR_INPUT: Input variable given to a POU.
VAR_OUTPUT: Output variable given from a POU.
VAR_IN_OUT: Input and output variable given as a pointer to a POU.

Important Data types

The table below shows the most important data types.

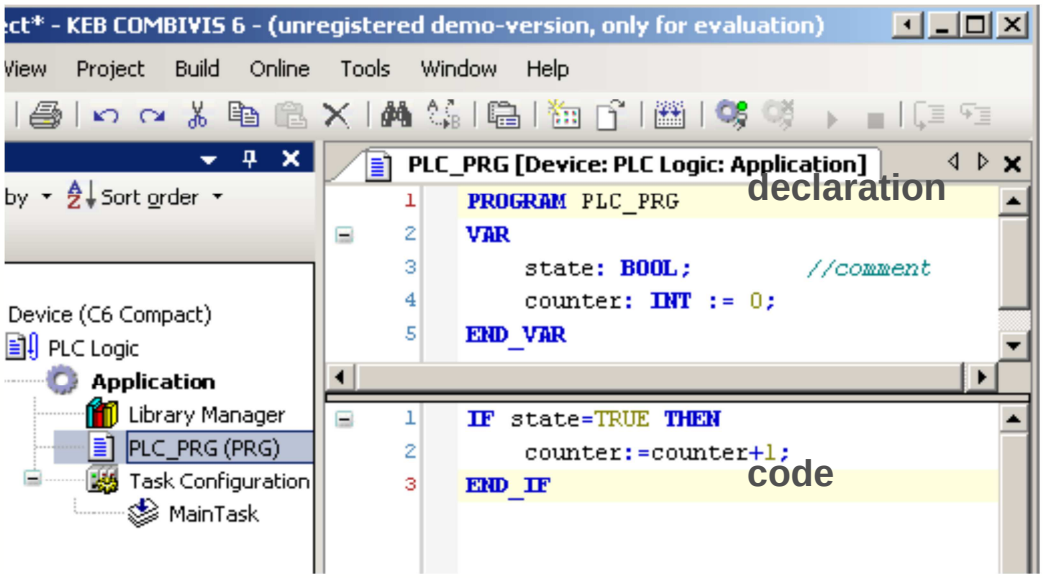
type	lower limit	upper limit	memory usage
BOOL	FALSE=0	TRUE=1	8 Bit (1 Byte)
BYTE	0	255	8 Bit (1 Byte)
WORD	0	65535	16 Bit (2 Byte)
INT	- 32768	32767	16 Bit (2 Byte)
DWORD	0	4294967295	32 Bit (4 Byte)
DINT	-2147483648	2147483647	32 Bit (4 Byte)
REAL	1.175494351e-38	3.402823466e+38	32 Bit (4 Byte)
STRING	Text *str:STRING(35):=,This is a STRING';*		1 Byte + 1Byte* number of chars
TIME	Time constant *Bsp.: tTimer:TIME:=T#100S12ms;*		32 Bit (4 Byte)
ARRAY	1 to 9 Dimensional field *fields: ARRAY[1..13, 1..4] OF INT;*		variable

Structured text

The structured text is a powerful language you can use to build loops and conditional statements very easy. The basic syntax is nearly the same as in other high-level languages like turbo pascal,C/C++ or Basic.
In the following some basic functions and statements in structured text will be shown.

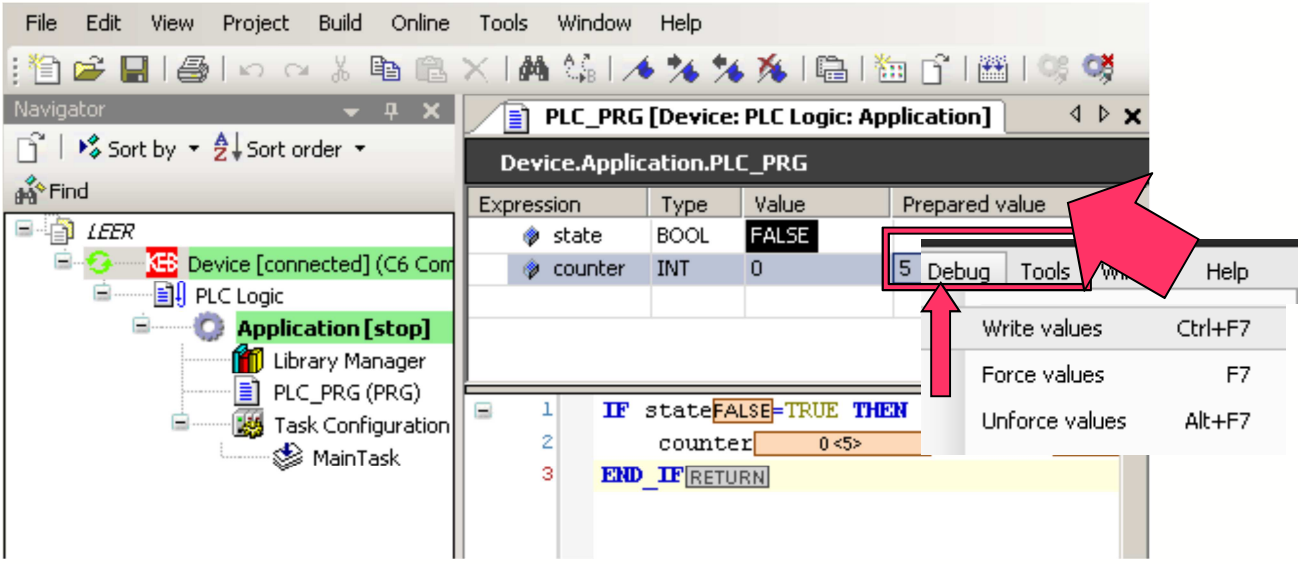
The ST-Editor

The main window is split up in two parts. The upper part contains the declaration part of the POU and the lower part contains the main program code of the POU.
The colors in the editor are making it much easier to read and understand the program code.



The ST-Editor in online mode

In online mode the Structured Text Editor (ST-Editor) provides views for monitoring and for writing and forcing the variables and expressions on the controller. Debugging functionality (breakpoints, stepping etc.) is available.



Expressions

An expression is a construction which after its evaluation returns a value. This value is used in instructions.
Expressions are composed of operators, operands and/or assignments. An operand can be a constant, a variable, a function call or another expression.

Examples:

```
33                          (* constant *)
ivar                        (* variable *)
fct(a,b,c)                  (* function call *)
a AND b                     (* expression *)
(x*y) / z                   (* expression *)
real_var2 := int_var;       (* assignment *)
```

Evaluation of expressions

The evaluation of expression takes place by means of processing the operators according to certain binding rules. The operator with the strongest binding is processed first, then the operator with the next strongest binding, etc., until all operators have been processed.
Operators with equal binding strength are processed from left to right.
Below you find a table of the ST operators in the order of their binding strength:

Operation	Symbol	Binding strength
Put in parentheses	(expression)	Strongest binding
Function call	Function name (parameter list)	
Exponentiation	EXPT	
Negate	-	
Building of complements	NOT	
Multiply	*	
Divide	/	
Modulo	MOD	
Add	+	
Subtract	-	
Compare	<,>,<=,>=	
Equal to	=	
Not equal to	<>	
Boolean AND	AND	
Boolean XOR	XOR	
Boolean OR	OR	Weakest binding

Comments

To comment only one row just use // (two slashes). To comment more than one row use a (* at the beginning and a *) at the end of your comment. Comments are an easy way to exclude parts of your program without deleting them or just use them to describe what you are doing so other programmers can understand your code even faster.

Data Type conversions

Its is forbidden to implicitly convert from a "larger" type to a "smaller" type (for example from INT to BYTE or from DINT to WORD). Special type conversions are required if one wants to do this. One can basically convert from any elementary type to any other elementary type.
Syntax:
<elem.Typ1>_TO_<elem.Typ2>
samples:

```
wValue := REAL_TO_WORD(drive1.Angle);
rValue := WORD_TO_REAL(wSum1) + 1.55;
dw := TIME_TO_DWORD(T#5m); (* Result is 300000 *)
bv := STRING_TO_BYTE('500'); (* Result is 244 *)
```

Conditional statements

If....Else...
The if-condition is an easy to use statement to decide what happens when a condition comes true.

```
IF state=TRUE THEN
        //Do this...
ELSE
        //Do that...
END_IF
```

To make more complex conditions you can combine different statements. It is also possible to use another condition in the ELSE part of the if statement.

```
IF state=TRUE AND NOT error=TRUE THEN
        //Do this...
ELSIF error=TRUE THEN
        //Do that...
ELSE
        //Do something different...
END_IF
```

CASE...OF

To check many values of one variable you can use the CASE instruction.

```
CASE number OF
0:        //do this...
1,6:      //do that...
2..5:     //do this...
DEFAULT
          //do this when no value above was found.
END_CASE
```

Loops

FOR loop

With the FOR loop one can program repeated processes.

Syntax:

```
INT_Var :INT;
FOR <INT_Var> := <INIT_VALUE> TO <END_VALUE> {BY <Step size>} DO
          <Instructions>
END_FOR;
```

The part in braces {} is optional.
The <Instructions> are executed as long as the counter <INT_Var> is not greater than the <END_VALUE>. This is checked before executing the <Instructions> so that the <instructions> are never executed if <INIT_VALUE> is greater than <END_VALUE>.
When <Instructions> are executed, <INT_Var> is increased by <Step size>. The step size can have any integer value. If it is missing, then it is set to 1. The loop must also end since <INT_Var> only becomes greater.

Example:

```
FOR Counter:=1 TO 5 BY 1 DO
          Var1:=Var1*2;
END_FOR;
```

WHILE loop

The WHILE loop can be used like the FOR loop with the difference that the break-off condition can be any Boolean expression. This means you indicate a condition which, when it is fulfilled, the loop will be executed.

Syntax:

```
WHILE <Boolean expression> DO
        <Instructions>
END_WHILE;
```

The <Instructions> are repeatedly executed as long as the <Boolean_expression> returns TRUE. If the <Boolean_expression> is already FALSE at the first evaluation, then the <Instructions> are never executed. If <Boolean_expression> never assumes the value FALSE, then the <Instructions> are repeated endlessly which causes a relative time delay.

Example:

```
WHILE Counter<>0 DO
        Var1 := Var1*2;
        Counter := Counter-1;
END_WHILE
```

REPEAT loop

The REPEAT loop is different from the WHILE loop because the break-off condition is checked only after the loop has been executed. This means that the loop will run through at least once, regardless of the wording of the break-off condition.

Syntax:

```
REPEAT
        <Instructions>
UNTIL <Boolean expression>
END_REPEAT;
```

The <Instructions> are carried out until the <Boolean expression> returns TRUE.
If <Boolean expression> is produced already at the first TRUE evaluation, then <Instructions> are executed only once. If <Boolean_expression> never assumes the value TRUE, then the <Instructions> are repeated endlessly which causes a relative time delay.

Example:

```
REPEAT
        Var1 := Var1*2;
        Counter := Counter-1;
UNTIL
        Counter=0
END_REPEAT;
```

CONTINUE instruction

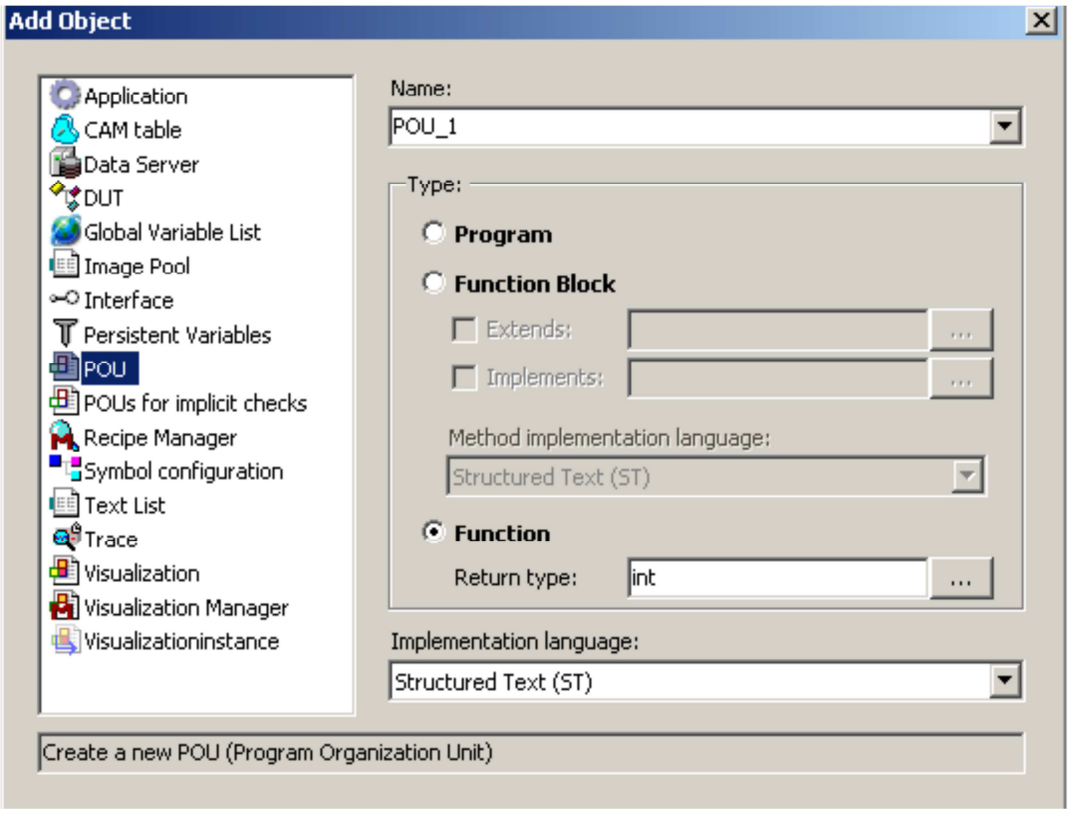
As an extension to the IEC 61131-3 standard (ExST) the CONTINUE instruction is supported within FOR, WHILE and REPEAT-loops.
CONTINUE makes the execution proceed with the next loop-cycle.

Example:

```
FOR Counter:=1 TO 5 BY 1 DO
        INT1:= INT1/2;
        IF INT1=0 THEN
                CONTINUE; (* to avoid division by zero *)
        END_IF
        Var1:=Var1/INT1; (* only executed, if INT1 is not "0" *)
END_FOR;
Erg:=Var1;
```

EXIT instruction
If the EXIT instruction appears in a FOR, WHILE, or REPEAT loop, then the innermost loop is ended, regardless of the break-off condition.

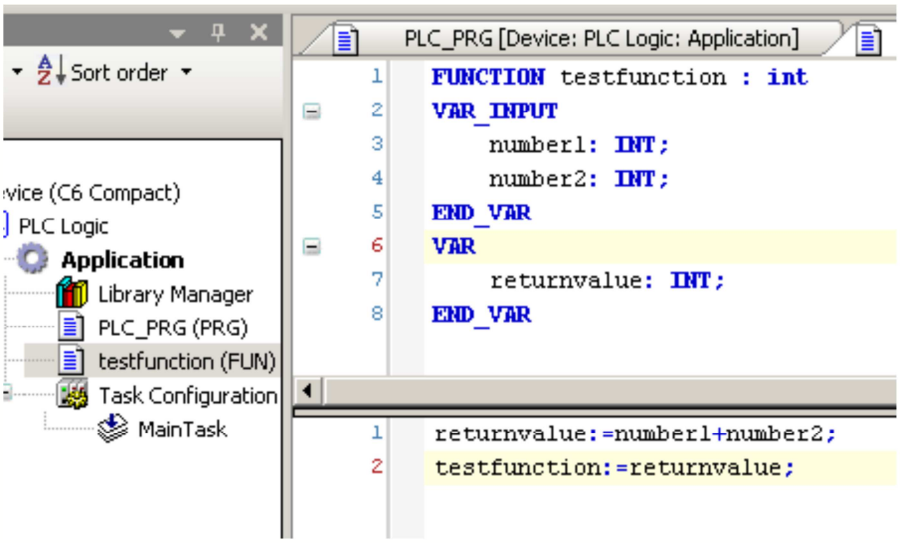
Functions and execution of other POUs



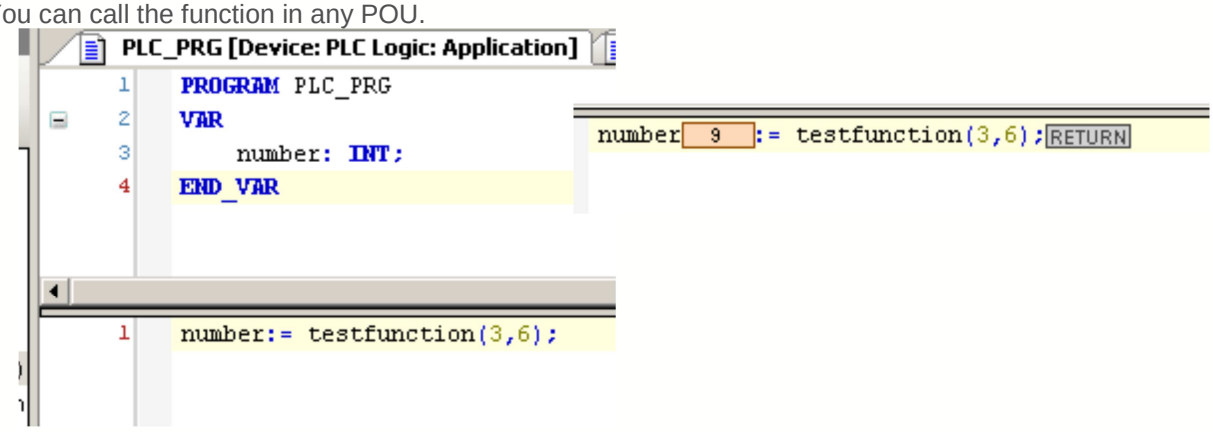
You can add a function or a new POU with the object manager. To create a function which returns a value select function and type in the return type. The name of the POU is the name of the function.

Sample function

The function should add number1 and number2 and return it to the main Program.



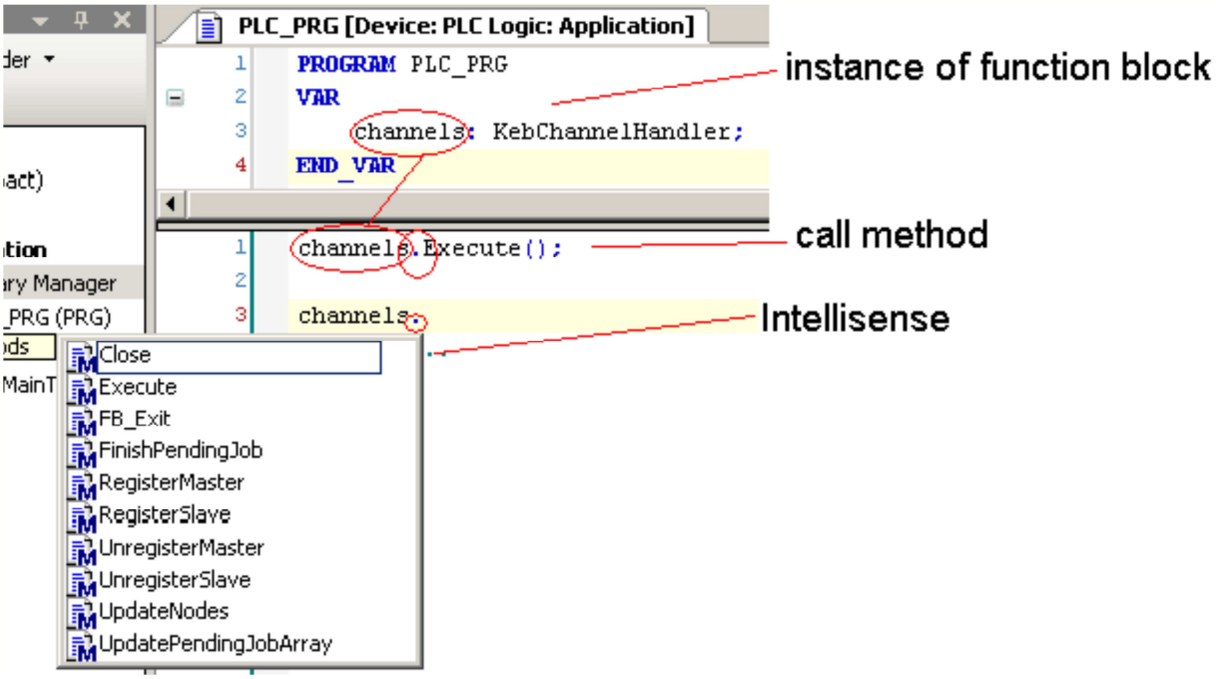
You can call the function in any POU.



How to use methods

Supporting object oriented programming methods can be used to describe a sequence of instructions. Like a function a method is not an independent POU, but must be assigned to a function block. It can be regarded as a function which contains an instance of the respective function block.

To use such methods you have to declare an instance of a function block (like the KebChannelHandler). Then you can execute the methods within the instance with a pointer to that method you want to call. Just separate the name of the instance and the name of the method with a dot.
The ST-Editor brings an auto-complete function (Intellisense) to help you using the available methods of an object. When typing the dot it opens a list of the methods you have access to.



Disclaimer

KEB Automation KG reserves the right to change/adapt specifications and technical data without prior notification. The safety and warning reference specified in this manual is not exhaustive. Although the manual and the information contained in it is made with care, KEB does not accept responsibility for misprint or other errors or resulting damages. The marks and product names are trademarks or registered trademarks of the respective title owners.

The information contained in the technical documentation, as well as any user-specific advice in verbal or in written form are made to the best of our knowledge and information about the application. However, they are considered for information only without responsibility. This also applies to any violation of industrial property rights of a third-party.

Inspection of our units in view of their suitability for the intended use must be done generally by the user. Inspections are particular necessary, if changes are executed, which serve for the further development or adaption of our products to the applications (hardware, software or download lists). Inspections must be repeated completely, even if only parts of hardware, software or download lists are modified.

Application and use of our units in the target products is outside of our control and therefore lies exclusively in the area of responsibility of the user.

KEB Automation KG
Südstraße 38 • D-32683 Barntrup
fon: +49 5263 401-0 • fax: +49 5263 401-116
net: www.keb.de • mail: info@keb.de