INOVANCE

2023

Autoshop

LiteST Programming

INOVANCE TECHNOLOGY EUROPE GMBH

V2.0

TABLE OF CONTENTS

1 GENERAL DATA

Date:	09.04.2023
Hardware:	H5U & Easy PLC
Software:	AutoShop v4.8.1.0
Info:	Autoshop LiteST programming

2 PURPOSE OF THIS DOCUMENT

The purpose of this document is to explain Structure Language (ST) programming on Easy series PLCs and H5U with AutoShop development software.

AutoShop v4.8.1.0 and above supports three languages ST, LD and SFC. ST and LD language can be mixed in the same project.

NOTE : Structured language programming is supported by H5U series firmware version **5.14.0.0** and above, and EASY series firmware version **5.67.0.0** and above. The AutoShop programming software must be at least version **v4.8.1.0**.

3 REVISION HISTORY

Revision	Date	Author	Description
1.0	17 January 23	RsR	First release
2.0	9 April 23	RsR	Add 10 IEC-61131-3 compatibility

4 INTRODUCTION

LiteST is a high-level text programming language for automation systems, and its grammatical structure is similar to PASCAL. Simple standard structures ensure fast and efficient programming.

LiteST uses many traditional features of high-level languages, including: variables, operators, and control flow statements. The LiteST language has a free text editing method.

Compared with the IL programming method, it has no fixed format restrictions. By adding additional placeholders, the entire program has a hierarchical structure in terms of appearance and structure, which is easy to read and understand.

Graphical programming methods such as LD, and full text are also easy to translate and reuse.

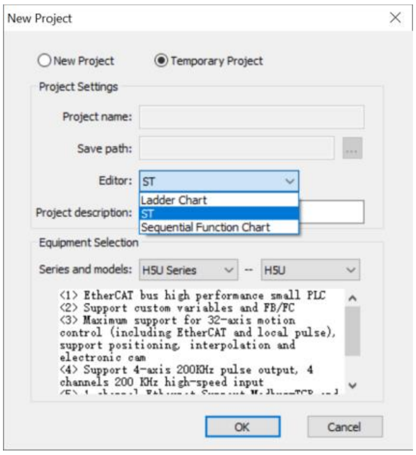
Example:

```
IF A>0 THEN
    X:=10;
ELSIF
    X:=0;
END_IF;
```

5 CREATE NEW PROJECT

When creating a new project in Autoshop we can select the language of the main program. We have three possibilities ladder language (LD), Structured Language (ST), Sequential Function Chart (SFC).

In the project you can switch between ST and LD using the FB, FC and the subprograms.



New Project

○ New Project ● Temporary Project

Project Settings

Project name:

Save path:

Editor: ST

Ladder Chart
ST
Sequential Function Chart

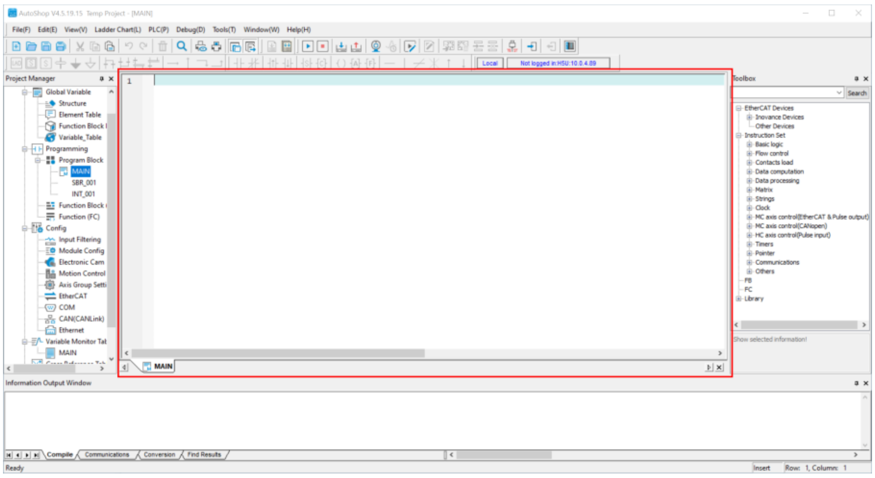
Project description:

Equipment Selection

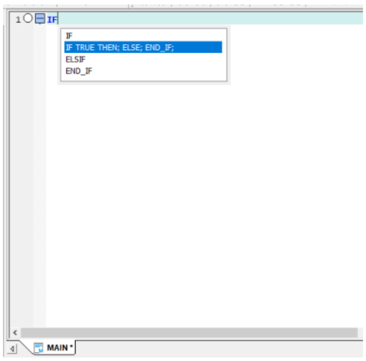
Series and models: H5U Series -- H5U

```
<1> EtherCAT bus high performance small PLC
<2> Support custom variables and FB/FC
<3> Maximum support for 32-axis motion
control (including EtherCAT and local pulse),
support positioning, interpolation and
electronic cam
<4> Support 4-axis 200KHz pulse output, 4
channels 200 KHz high-speed input
```

When creating a new project in ST language, the main program MAIN appears where you can write the program in ST language.

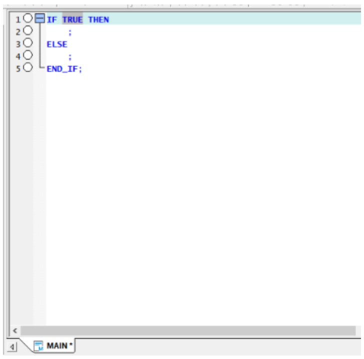


The ST language editor is a text editor. When we start writing in the editor, a help window appears to facilitate the introduction of ST language instructions. n the window you can select the relevant instruction and by pressing the enter key the editor completes the instruction.



```
IF
```

IF
IF TRUE THEN; ELSE; END_IF;
ELSIF
END_IF



```
IF TRUE THEN
    ;
ELSE
    ;
END_IF;
```

6 PROGRAMMING LANGUAGE (LITEST)

6.1 EXPRESIONS AND OPERATORS

Like in the LD programming environment, block diagrams with different functions constitute the basic elements of LD programming. In LiteST, expressions are the building blocks of the LiteST language. Expressions consist of operators and operands. The operand can be a constant, variable, function call, or other expression. E.g:

1) Constants, for example: 20, 1.43, 16#10
2) Variables, for example: iVar, xEnable,..
3) Function call, the value is the return value of the call, for example: Fun1(1,2,4)
4) Other expressions: 10+3, var1 OR var2, (x+y)/z, iVar1:=iVar2+22

The evaluation of the expression evaluates the operands by operator in the order defined by the particular operator precedence. The operator with the highest precedence in an expression shall be evaluated first, followed by the operator with the next lower precedence, and so on, from highest to lowest.

Operators with equal precedence shall be performed in the left-to-right order as written in the expression.

For example: if A, B, C and D are of type INT and have values 1, 2, 3, 4 respectively, then A+BC*ABS(D) shall equal -9, and (A+BC)*ABS(D) should be equal to 0.

When an operator has two operands, the leftmost operand will be evaluated first. For example, in the expression SIN(X)*COS(Y), the expression SIN(X) should be evaluated first, COS(Y) should be evaluated next, and the product should be evaluated last.

Operation type	Symbol	Example	Priority
Brackets	(expression)	(A+B/C),(A+B)/C,A/(B+C)	9 (highest)
Function call	Function name (parameter list, separated by commas)	LN(A),MAX(X,Y)	8
Negate	-	-A	7
Positive	+	+B	7
Complement	NOT	NOT C	7
Multiplication	*	A*B	6
Division	/	A/B	6
Division remainder	MOD	A MOD B	6
Add	+	A+B	5
Subtraction	-	A-B	5
Compare	<,>,<=,>=	A<B	4
Equal	=	A=B	4
Not equal to	<>	A<>B	4
Logic AND	AND	A AND B	3
Logic XOR	XOR	A XOR B	2
Logic OR	OR	A OR B	1 (minimum)

6.2 VARIABLES

In the H5U programming system, in addition to programming directly using direct addresses, such as X, Y, M, D, R, you can also program with "variables" without specific storage addresses. This improves the readability of writing code.

Type	Capacity	Data type	Description
Pointer	4096 points (32 bits)	BOOL/ INT/ DINT/ REAL array	Pointer variable Power failure does not save
BOOL	2MB	INT/DINT/REAL variables INT/DINT/REAL array INT/ DINT/ REAL structs	256KB power-down save
INT			
DINT			
REAL			

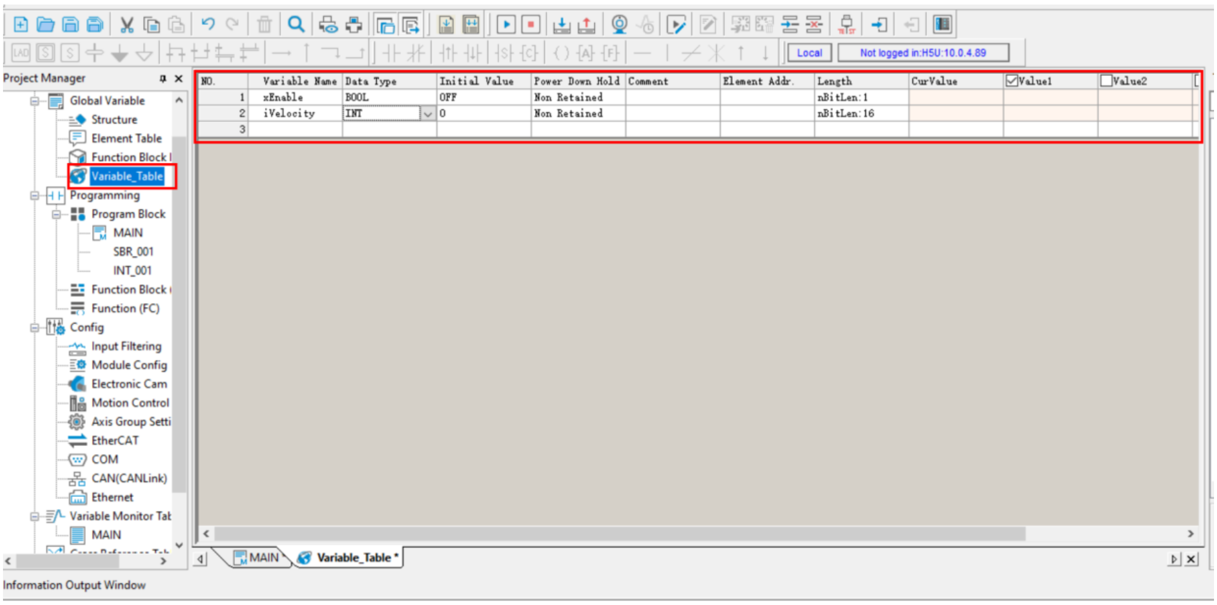
NOTE : The **REAL** type complies with the standard **IEEE-754** Floating Point **32-bits single precision** representation. For example, a single precision floating point number has a maximum of 7 decimal significant digits. If the floating-point number 1234567.89 is transferred to the D0 register, the value of D0 is 1234567.9.

The definition of variables must follow the following rules:

1. It can only consist of ***_, letters, numbers, Chinese characters*** and cannot start with ***_, numbers***
2. Cannot have the same name as ***device form, constant, standard data type, statement***
3. They cannot be keywords like ***ARRAY, TRUE, FALSE, ON, OFF, NULL***.
4. Variables are not case sensitive

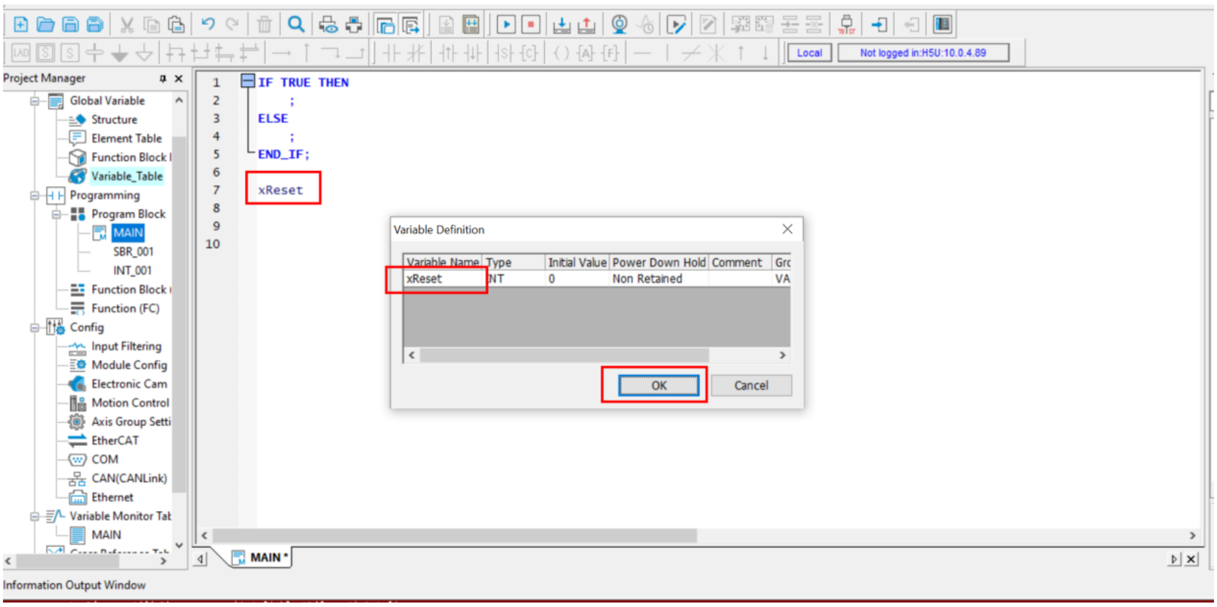
There are two ways to declare variables. In the variable tables or directly in the ST code editor.

To access the table of variables, you can click on the name of the table in the project tree:



NO.	Variable Name	Data Type	Initial Value	Power Down Hold	Comment	Element Addr.	Length	CurValue	☑Value1	□Value2
1	xEnable	BOOL	OFF	Non Retained			nBitLen:1			
2	iVelocity	INT	0	Non Retained			nBitLen:16			
3										

When write directly in the program editor, and then press the ENTER key or click the area outside the basic block of the program the editor interprets that it is a name of a variable and opens the dialog box that allows to declare the variable that will be added to the general table of variables.



```
IF TRUE THEN
    ;
ELSE
    ;
END_IF;

xReset
```

Variable Definition

Variable Name	Type	Initial Value	Power Down Hold	Comment	Grc
xReset	INT	0	Non Retained		VA

6.3 CONSTANTS

There are several ways to represent constants in the ST programs:

1) Default decimal value such as

   A:=100; (Integer value)

   A:=12.20; (Floating point value)

2) Separated with underline means

   A:=10#100_10; (Decimal value)

   A:=16#FF_AE_12; (Hexadecimal value)

   A:=2#1100_1111_11_10; (Binary value)

3) Boolean values

   xEnable:=TRUE;

   xEnable:=FALSE;

4) LiteST also supports the constant expression of LD language. That is, K100 represents the constant 100, H data represents the hexadecimal number, and E represents the floating point data.

6.4 COMMENTS

There are two ways of writing comments in structured text.

1) Single-line comments: start with "//" . For example: "// This is a comment."
2) Multi-line comments: start with "(*" and end with "*)". For example: "(*This is a comment.*)"

Comments can be anywhere in the declaration or implementation section of the LiteST editor.

Nesting annotations: annotations can be placed inside other annotations:

```
(*
a:=inst.out; (*to be checked*)
b:=b+1;
* )
```

6.5 FB, FC, SUBROUTINE, INTERRUPT

The function (FC) and the function block (FB) are like a subroutine. Users can create a piece of repeating logic in this block and call the FC or FB every time that logic needs to be executed.

NOTE : FB, FC nested calls cannot exceed 8 levels.

NOTE : The number of nested levels of SBR cannot exceed 6 levels.

Function block (FB)

A function block (FB) can abstractly encapsulate a part of the repeated code used in the program in a general program block, which can be called repeatedly in the program. Using the encapsulated function blocks in programming can improve the efficiency of program development, reduce programming errors, and improve program quality.

A function block can generate one or more values during execution. The function block keeps its own special internal variables. The controller execution system allocates memory for the function block state variables. These internal variables constitute their own state characteristics. Modifying the values of the input parameters of the FB we obtain different results in the execution or in the output parameters.

FBs must be instantiated before being used in a program. In other words, a variable of the FB type must be defined in order to be able to use the FB in the program.

NOTE : Motion function blocks do not need to be instantiated. They are called directly from the program using the name of the FB.

Function (FC)

Unlike a function block (FB), a function (FC) is a logic block without memory. This block can be created by the user and called as many times as necessary. Like the FB, the functions also have input parameters that allow to alter the execution and its results (output parameters).

Functions do not need to be instantiated, they can be called directly from code to execute their internal code.

Subroutine

The subroutine is called as a function, but without parameters.

Interrupt

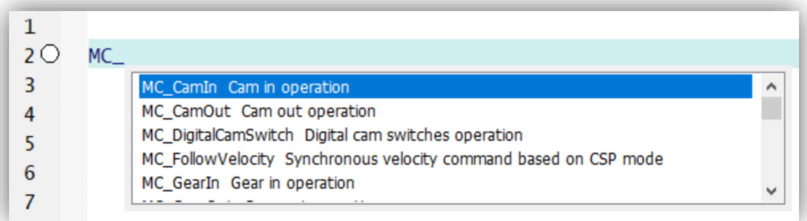
Subroutines can be called through an interrupt. It does not need to be called manually as a subroutine, it must call *EI();* before turning it on. It needs to call *DI();* to disable subroutine interrupt trigger.

6.6 SMART TYPING AND HINTS

The ST code editor offers some functionality to make application development easier.

Quick input

The "Quick input" tool makes it easy to enter commands in the ST editor. When you begin to write the name of a command, a window appears that shows the different possibilities depending on the written command.

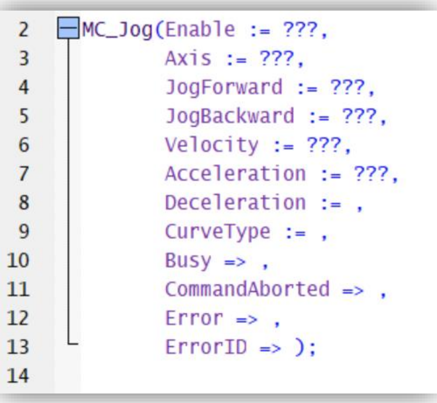


```
MC_
```

- MC_CamIn Cam in operation
- MC_CamOut Cam out operation
- MC_DigitalCamSwitch Digital cam switches operation
- MC_FollowVelocity Synchronous velocity command based on CSP mode
- MC_GearIn Gear in operation

You can select the desired command and press ENTER for the editor to finish typing the full name of the command.

After entering the name of the FB or FC command, press the TAB key, and all the inputs output of that function will appear. If the default parameter after the parameter is "???", it indicates that the input parameter is required. Otherwise, you can decide whether or not to put the parameter according to the needs of the application.

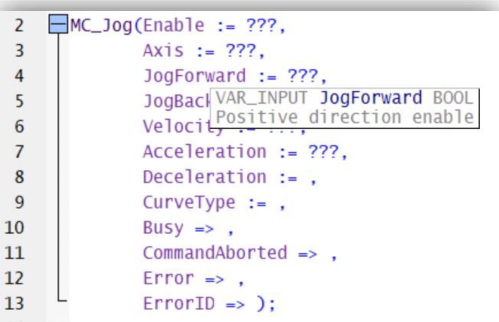


```
MC_Jog(Enable := ???,
       Axis := ???,
       JogForward := ???,
       JogBackward := ???,
       Velocity := ???,
       Acceleration := ???,
       Deceleration := ,
       CurveType := ,
       Busy => ,
       CommandAborted => ,
       Error => ,
       ErrorID => );
```

Mouse tool tips

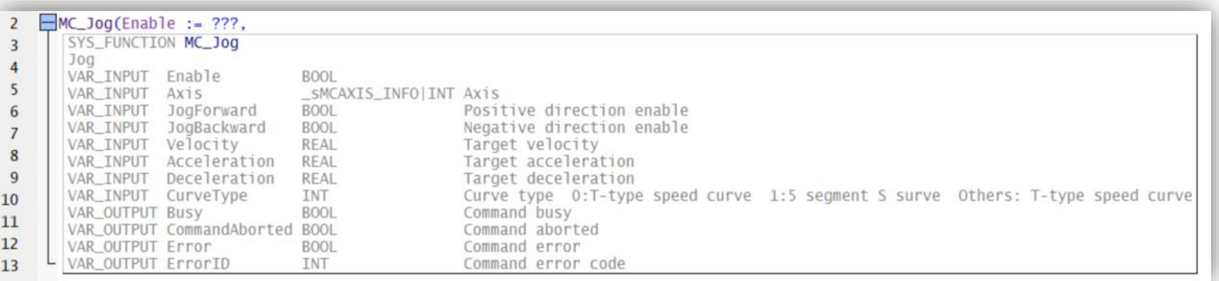
When moving the mouse cursor over the editor commands or variables, a window appears with information about the specific element.

The following image shows the information of the input parameter JogForward of the FB MC_Jog:



```
MC_Jog(Enable := ???,
        Axis := ???,
        JogForward := ???,
        JogBack VAR_INPUT  JogForward BOOL
        Velocit Positive direction enable
        Acceleration := ???,
        Deceleration := ,
        CurveType := ,
        Busy => ,
        CommandAborted => ,
        Error => ,
        ErrorID => );
```

If the cursor is over an FB the information window shows all the inputs/outputs of the FB with the respective information:



```
MC_Jog(Enable := ???,
 SYS_FUNCTION MC_Jog
 Jog
 VAR_INPUT  Enable         BOOL
 VAR_INPUT  Axis           _sMCAXIS_INFO|INT Axis
 VAR_INPUT  JogForward     BOOL              Positive direction enable
 VAR_INPUT  JogBackward    BOOL              Negative direction enable
 VAR_INPUT  Velocity       REAL              Target velocity
 VAR_INPUT  Acceleration   REAL              Target acceleration
 VAR_INPUT  Deceleration   REAL              Target deceleration
 VAR_INPUT  CurveType      INT               Curve type  0:T-type speed curve  1:5 segment S surve  Others: T-type speed curve
 VAR_OUTPUT Busy           BOOL              Command busy
 VAR_OUTPUT CommandAborted BOOL              Command aborted
 VAR_OUTPUT Error          BOOL              Command error
 VAR_OUTPUT ErrorID        INT               Command error code
```

6.7 SYNTAX INSTRUCTIONS

6.7.1 ASSIGNMENT INSTRUCTION

An assignment statement replaces the current value of one or more element variables with the result of evaluating an expression. An assignment statement should consist of: a variable reference on the left, followed by the assignment operator ":=", followed by an expression requiring a value.

Example:

```
A:=B*10;
```

After finishing execution, the value of A is 10 times that of B.

6.7.2 FUNCTION BLOCK CALL

Functions and Instantiated function blocks can be called with the input/output parameters in the same call or the input/output parameters can be assigned or retrieved on a different line than the call:

Syntax (Parameters in the FC/FB call):

Syntax 1:

```
FB instance name(FB input variable  := value, FB output variable  => value, …);
```

Syntax 2:

```
FB instance name( FB input variable  := value,
                  FB output variable  => value,
                  …);
```

Syntax (Parameters outside the FC/FB call):

```
FB instance name.FB input variable  := value;
Result:=FB instance name.FB output variable;
FB instance name();
```

Example:

Call an instance of the function block (MAXFB) for calculating the maximum value, and input the input parameters D0, D1 and output parameter D2. After the function is executed, the result can be assigned to the variable maxVar.

```
MYFB(VAR1 := D0,VAR2 := D1,RESULT => D2);
maxVar := MYFB.RESULT;
```

NOTE : **MYFB** is the function block instance of MAXFB.

6.7.3 INSTRUCTION IF

Through the ***IF*** keyword, you can add an execution condition and execute the corresponding code according to the condition.

Syntax:

```
IF <boolean_expression1> THEN
<IF_Command
{ELSIF <boolean expression 2> THEN
<ELSIF_command 1>
ELSIF <boolean expression n> THEN
END_IF;
```

6.7.4 INSTRUCTION CASE

Using the **CASE** instruction, you can list and process corresponding commands according to a condition variable and its corresponding multiple values. Condition variables can only be integers.

Syntax:

```
CASE <Var1> OF
       <value1>: <Instruction 1>
       <value2>: <Instruction 2>
       <value3, value4, value5>: <Instruction 3>
       <value n>: <Instruction n>
ELSE
       <ELSE Instruction>
END_CASE;
```

6.7.5 INSTRUCTION WHILE

The **WHILE** loop can be used as a loop processing like the **FOR** loop, but unlike the **FOR** loop, the loop condition can be any Boolean expression.

Once the loop condition is met, the loop executes, otherwise it exits the loop.

Syntax:

```
WHILE <boolean expression> DO
        <instructions>
END_WHILE;
```

When the value of <Boolean_expression> is **TRUE**, the <Instructions> instruction will be executed until the value of <Boolean_expression> is **FALSE**. If <Boolean_expression> evaluates to **FALSE** for the first time, <Instructions> will never be executed. If <Boolean_expression> is always **TRUE**, the repeated execution of <Instructions> will not stop, and it will enter into an endless loop state.

NOTE: When programming, be sure not to have an infinite loop.

6.7.6 INSTRUCTION REPEAT

A **REPEAT** loop is different from a **WHILE** loop because the loop condition is checked after the loop instruction is executed, which means that the loop is executed at least once regardless of the loop condition value.

Syntax:

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

<Instructions> has been executed until the value of <Boolean expression> is **TRUE**. If <Boolean expression> evaluates to **TRUE** for the first time, then <Instructions> are executed only once. If the value of <Boolean_expression> is always **FALSE**, then <Instructions> will be executed forever, resulting in an infinite loop.

NOTE: When programming, be sure not to have an infinite loop.

6.7.7 INSTRUCTION FOR

Through the FOR loop, it is possible to program repeated processing logic.

Syntax:

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

Parts inside {} are optional.

INT_Var is a counter, which is an integer type. As long as the counter <INT_Var> is not greater than <END_VALUE>, <Instructions> will be executed. exist

Check the condition first before executing <Instructions>, if <INIT_VALUE> is greater than <END_VALUE>, <instructions> will not be executed.

When <Instructions> is executed once, <INT_Var> will automatically increase <Step size>. <Step size> can be any integer value, if you do not set this parameter, the default value is 1. When <INT_Var> is greater than <END_VALUE>, the loop stops.

> **NOTE** : When programming, be sure not to have an infinite loop.

6.7.8 INSTRUCTION EXIT

The **EXIT** instruction is used to exit a **FOR**, **WHILE**, or **REPEAT** loop. Interrupts the repeated processing of the loop **FOR**, **WHILE**, or **REPEAT** instruction, and simultaneously executes the next step of the repeated processing.

Syntax:

```
EXIT;
```

6.7.9 INSTRUCTION CONTINUE

The **CONTINUE** command is used in **FOR**, **WHILE**, and **REPEAT** loops to end the current loop early and restart the next loop. Interrupting the loop is different from exiting the loop. To interrupt the loop means to ignore the current loop. That is, the statement of the current loop is not executed, but the execution continues in the next cycle of the loop.

Example:

```
FOR Counter:=1 TO 5 BY DO
    INT1:=INT1/2;
    IF INT1=0 THEN
        CONTINUE;
    END_IF
    Var2:=Var1/UBT1L
END_FOR;
```

6.7.10 INSTRUCTION RETURN

The **RETURN** instruction means to leave this POU when the precondition is **TRUE**.

Syntax:

```
RETURN;
```

Example:

```
IF b=TRUE THEN
       RETURN;
END_IF;
a:=a+1;
```

If b is **TRUE**, the statement "a:=a+1;" will not be executed and the POU will be returned immediately.

> **NOTE** : Be careful when using the **RETURN** statement because it modifies the flow of the program and it can be difficult to follow the thread of execution..

7 SUPPORTED INSTRUCTIONS

All the instructions supported by PLC are summarized in the instruction quick reference table and classified according to the corresponding functional categories.

Command category	Name	Description	Supported Language
Contact command	LD	Load normally open contact	LD
	LDI	Load normally closed contact	LD
	AND	Normally open contacts in series	LD
	ANI	Series normally closed contacts	LD
	OR	Parallel normally open contacts	LD
	ORI	Parallel normally closed contacts	LD
	LDP	Take the rising edge of the pulse	LD
	LDF	Take pulse falling edge	LD
	ANDP	Serial connection with pulse rising edge detection	LD
	ANDF	Serial connection with pulse falling edge detection	LD
	ORP	Or pulse rising edge detection connected in parallel	LD
	ORF	Or pulse falling edge detection connected in parallel	LD
	MEP	Rising edge pulse of operation result	LD
	MEF	The falling edge of the operation result is pulsed	LD
Output control command	OUT	Drive coil	LD
	SET	Set Action Save Coil Instruction	LD
	RST	Contact or register clear	LD
	ZSET	All data set	LD, LiteST
	ZRST	All data reset	LD, LiteST
	PLS	Pulse rising edge detection coil command	LD
	PLF	Pulse (F) falling edge detection coil command	LD
	ALT	Alternate output	LD
	R_TRIG	Rising edge detection instruction	LD, LiteST
	F_TRIG	Falling edge detection instruction	LD, LiteST
Power flow control command	INV	Negate the result of the operation	LD
Flow Control Instructions	CJ	Conditional jump instruction	LD
	LBL	Label instruction	LD
	CALL	Subroutine call instruction	LD
	SSRET	Subroutine waits for conditional return	LD
	EI	Discontinued license	LD, LiteST
	DI	Interrupt disabled	LD, LiteST
	WDT	Watchdog Timer Reset Instruction	LD
	FOR	Loop range start command	LD
	NEXT	Loop range end instruction	LD

Command category	Name	Description	Supported Language
SFC instruction	STL	Program jumps to sub-bus	LD
	RET	Program returns to main bus	LD
	OUTSTL	The output program jumps to the auxiliary bus	LD
	SETSTL	Set program jumps to auxiliary bus	LD
	RSTSTL	Clear program jump to secondary bus	LD
Contact comparison	LD＝	LD contacts compare equal to	LD
	LD>	LD contacts are larger than	LD
	LD<	LD contacts are less than	LD
	LD<>	LD contacts compare not equal to	LD
	LD>＝	LD contacts compare greater than or equal to	LD
	LD<＝	LD contacts compare less than or equal to	LD
	AND＝	AND contacts compare equal to	LD
	AND>	AND contacts are greater than	LD
	AND<	AND contacts compare less than	LD
	AND<	AND contacts compare not equal to	LD
	AND>＝	AND contacts compare greater than or equal to	LD
	AND<＝	AND contacts compare less than or equal to	LD
	OR＝	OR contacts compare equal to	LD
	OR>	OR contacts are greater than	LD
	OR<	OR contacts are less than	LD
	OR<>	OR contacts compare not equal to	LD
	OR>＝	OR contacts compare greater than or equal to	LD
	OR<＝	OR contacts compare less than or equal to	LD
	LD&	LD logic and operation	LD
	LD|	LD logic or operation	LD
	LD^	LD logical exclusive OR operation	LD
	AND&	AND logic and operation	LD
	AND|	AND logical or operation	LD
	AND^	AND logic XOR operation	LD
	OR&	OR logic and operation	LD
	OR|	OR logical or operation	LD
	OR^	OR logical exclusive-or operation	LD
	FLDD >	Floating-point number > compared status contact S1 > S2 is turned on	LD
	FLDD >=	Floating-point number >= comparative status contact S1 ≧ S2 is turned on	LD
	FLDD <	Floating-point number <comparison status contact S1 < S2 conducts	LD
	FLDD <=	Floating-point number <= comparison status contact S1 ≦ S2 is turned on	LD

Command category	Name	Description	Supported Language
	FLDD =	Floating point number = state of comparison Contact S1 = S2 conduction	LD
	FLDD <>	The state contact S1 ≠ S2 of the floating-point number <> comparison is turned on	LD
	FANDD>	Floating-point number > comparison and status contact S1 > S2 is turned on	LD
	FANDD>=	When the floating point number >= compared with the status contact S1 ≧ S2, it is turned on	LD
	FANDD<	Floating-point number <compared with the state contact S1 < S2 conduction	LD
	FANDD<=	When the floating-point number <= compared with the status contact S1 ≦ S2, it is turned on	LD
	FANDD=	Floating point number = compared with status contact S1 = ON when S2	LD
	FANDD<>	Floating-point number <> compared with the status contact S1 ≠ S2 conduction	LD
	FORD>	Floating-point number > comparative or status contact S1 > S2 conducts	LD
	FORD>=	Floating-point number >= compared or state contact S1 ≧ S2 is turned on	LD
	FORD<	Floating-point number < compared or state contact S1 < S2 conducts	LD
	FORD<=	Floating-point number <= compared or state contact S1 ≦ S2 is turned on	LD
	FORD=	Float = compare or status contact S1 = ON when S2	LD
	FORD<>	Floating-point number <> compared or status contact S1 ≠ S2 conducts	LD
	LDZ>	Absolute value > comparative state contacts |S1 - S2| > |S3|	LD
	LDZ>=	Absolute value >= comparative status contact |S1 - S2|≧|S3|	LD
	LDZ<	The state contact |S1-S2|<|S3| is turned on when the absolute value<comparison	LD
	LDZ<=	Absolute value <= comparative state contact |S1 - S2|≦|S3|	LD
	LDZ=	Absolute value = compared state contact |S1 - S2| = |S3|	LD
	LDZ<>	Absolute value<>compared status contacts |S1 - S2|≠|S3|	LD
	ANDZ>	Absolute value > compared with state contact |S1 - S2| > | S3 conduction	LD

Command category	Name	Description	Supported Language
	ANDZ>=	Absolute value >= comparative and state contact |S1 - S2|≧|S3|	LD
	ANDZ<	Absolute value < compared with state contact |S1 - S2|<| S3 conduction	LD
	ANDZ<=	Absolute value <= compared with state contact |S1 - S2|≦|S3|	LD
	ANDZ=	Absolute value = compared with state contact |S1 - S2|=|S3 conduction	LD
	ANDZ<>	Absolute value<>compared with state contact |S1 - S2|≠|S3|	LD
	ORZ>	Absolute value > comparative OR status contacts |S1 - S2| > | S3 conduct	LD
	ORZ>=	Absolute value >= comparative OR state contact |S1 - S2|≧|S3|	LD
	ORZ<	Absolute value < comparative OR state contact |S1 - S2|<| S3 conducts	LD
	ORZ<=	Absolute value <= comparative OR state contact |S1 - S2|≦|S3|	LD
	ORZ=	Absolute value = comparative OR state contact |S1 - S2 | = | S3 conducts	LD
	ORZ<>	Absolute value<>Comparative OR status contacts |S1 - S2|≠|S3|	LD
Arithmetic	ADD	Binary Data Addition	LD
	SUB	Binary Data Subtraction	LD
	MUL	Binary data multiplication	LD
	DIV	Binary data division	LD
	MOD	Binary division remainder	LD, LiteST
	EADD	Binary floating point addition	LD
	ESUB	Binary floating point subtraction	LD
	EMUL	Binary floating point multiplication	LD
	EDIV	Binary floating point division	LD
	INC	Binary data plus one	LD
	DEC	Binary data minus one	LD
Data logic operation	WAND	Logical AND of binary data	LD
	WOR	Logical OR of binary data	LD
	WXOR	Logical XOR of binary data	LD
	NEG	Binary data complement	LD
	ENEG	Invert the sign of a binary floating-point number	LD
Word bit operations	BLD	Word or double word bit data contact instruction	LD
	BLDI	Word or double word bit data inversion instruction	LD
	BAND	Word or double word bit data and contact instruction	LD
	BANDI	Word or double word bit data and non-contact instructions	LD

Command category	Name	Description	Supported Language
	BOR	Word or double word bit data or contact instruction	LD
	BORI	Word or double word bit data or non-contact instruction	LD
	BOUT	Word or double word bit data output instruction	LD
	BSET	Word or double word bit data set instruction	LD
	BRST	Word or double word bit data reset instruction	LD
Trigonometric functions	SIN	Floating-point SIN operation instruction	LD
	COS	Floating-point COS operation instruction	LD
	TAN	Floating-point TAN operation instruction	LD
	ASIN	Binary floating-point number ARCSIN operation	LD
	ACOS	Binary floating point number ARCCOS operation	LD
	ATAN	Binary floating-point number ARCTAN operation	LD
	RAD	Binary floating-point number conversion from angle to radian	LD
	DEG	Binary floating-point number radian → angle conversion	LD
	SINH	Binary floating point number SINH operation	LD
	COSH	Binary floating point number COSH operation	LD
	TANH	Binary floating point number TANH operation	LD
Table operation	WSUM	Calculate the total value of the data	LD
	MEAN	average calculation	LD
	LIMIT	Upper and lower limit control	LD
	BZAND	dead zone control	LD
	ZONE	area control	LD
	SCL	Fixed coordinates (coordinate data of different points)	LD
	SCL2	Fixed coordinate 2 (X/Y coordinate data)	LD
Exponential operation	EXP	Binary Floating Point Exponentiation	LD
	LOGE	Binary floating-point natural logarithm operation	LD
	LOG	Base 10 logarithm operation of binary floating point numbers	LD
	ESQR	Binary floating-point square root	LD
	SQR	Binary data square root operation	LD

Command category	Name	Description	Supported Language
	POW	Floating-point number power operation	LD
Data conversion	INT	Binary floating point → BIN integer conversion	LD
	BCD	Convert binary data to BCD data	LD
	BIN	Convert BCD data to binary data	LD
	FLT	Binary data → binary floating point conversion	LD
	EBCD	Binary floating point → decimal floating point conversion	LD
	EBIN	Decimal floating point → binary floating point conversion	LD
	DABIN	Conversion of decimal ASCII→BIN	LD
	BINDA	BIN → decimal ASCII conversion	LD
	WTOB	Data separation in byte units	LD
	BITW	Assign bit element to word element	LD, LiteST
	BTOW	Byte Unit Data Binding	LD
	WBIT	Assign word element to bit element	LD, LiteST
	WTODW	Convert 16-bit word elements to 32-bit word elements	LD
	DWTOW	Convert 32-bit word elements to 16-bit word elements	LD
	MCPY	Data copy (memory copy, type conversion) instruction	LD, LiteST
	MSET	Data setting (memory setting and reset) instructions	LD, LiteST
	UNI	4-bit combination of 16-bit data	LD
	DIS	4-bit separation of 16-bit data	LD
	ASCI	HEX→ASCII conversion	LD
	HEX	ASCII→HEX conversion	LD
	DECO	data decoding	LD
	ENCO	data encoding	LD
Data transmission	MOV	assignment transfer	LD
	EMOV	binary floating point transfer	LD
	SMOV	shift transfer	LD
	BMOV	data bulk transfer	LD
	FMOV	Data one-to-many transfer	LD
	CML	data reverse transfer	LD
	CMP	data comparison	LD
	ECMP	Floating point comparison instructions	LD
	ZCP	regional comparison	LD
	EZCP	Floating-point area comparison instructions	LD
	SORTR	data sorting	LD
	SORTC	data sorting 2	LD
	SER	data search	LD
	FDEL	Data table deletion	LD
	FINS	Data table insertion	LD

Command category	Name	Description	Supported Language
	POP	Reading of late-entry data	LD
	RAMP	ramp command	LD
Data shift	ROR	cycle right	LD
	ROL	cycle left	LD
	RCR	Rotate right with carry	LD
	RCL	Rotate left with carry	LD
	SFTR	bit shift right	LD
	SFTL	shift left	LD
	WSFR	word right shift	LD
	WSFL	word shift left	LD
	SFWR	FIFO data writing	LD
	SFRD	FIFO data readout	LD
	SFR	16-bit data n-bit right shift (with carry)	LD
	SFL	16-bit data n-bit left shift (with carry)	LD
Other data processing	SWAP	high and low byte swapping	LD
	BON	ON bit judgment	LD
	SUM	Count the total number of ON bits	LD
	RAND	Generate random data with limited range	LD
	XCH	data exchange	LD
	ABS	Absolute value instruction	LD
	EABS	Floating point absolute value instructions	LD
	EFMOV	floating point multicast instruction	LD
	CCD	check code	LD
	CRC	CRC check code calculation	LD
	LRC	LRC check code calculation	LD
Matrix Operations	BK+	Data Block Addition	LD
	BK-	Data Block Subtraction	LD
	MAND	Matrix AND operation	LD
	MOR	matrix or operation	LD
	MXOR	Matrix XOR operation	LD
	MXNR	Matrix exclusive-or operation	LD
	MINV	Matrix inversion operation	LD
Matrix comparison	BKCMP=	Matrix equals comparison (S1=S2)	LD
	BKCMP>	Matrix greater than comparison (S1>S2)	LD
	BKCMP<	Matrix less than comparison (S1<S2)	LD
	BKCMP<>	Matrix not equal comparison (S1≠S2)	LD
	BKCMP<=	Matrix less than or equal comparison (S1≤S2)	LD
	BKCMP>=	Matrix greater than or equal comparison (S1≥S2)	LD
String command	STR	integer → string conversion	LD
	STRMOV	String Direct Assignment Instructions	LD
	VAL	String→Integer conversion	LD
	ESTR	Binary floating point number → string conversion	LD

Command category	Name	Description	Supported Language
	EVAL	String → binary floating point conversion	LD
	$ADD	combination of strings	LD
	LEN	Check out the length of the string	LD
	INSTR	string search	LD
	RIGHT	Extract from the right side of the string	LD
	LEFT	Extract from the left side of the string	LD
	MIDR	Randomly extract from a string	LD
	MIDW	Any replacement in the string	LD
	$MOV	transmission of strings	LD
Clock instruction	TCMP	Clock Data Comparison	LD
	TZCP	Clock Data Interval Comparison	LD
	TADD	Clock Data Addition	LD
	TSUB	Clock data subtraction	LD
	HTOS	Second conversion of hour, minute and second data	LD
	STOH	Second to hour, minute, second data conversion	LD
	TRD	clock data read	LD
	TWR	clock data write	LD
	HOUR	Chronograph	LD
High speed counter (H5U)	HC_Counter	High-speed counter enable	LD, LiteST
	HC_Preset	High-speed counter preset value	LD, LiteST
	HC_TouchProbe	Touch probe function	LD, LiteST
	HC_Compare	High-speed counter comparison	LD, LiteST
	HC_ArrayCompare	High-speed counter array comparison	LD, LiteST
	HC_StepCompare	High-speed counter equidistant distance comparison	LD, LiteST
Bus encoder axis	ENC_Counter	Encoder enable	LD, LiteST
	ENC_Reset	Encoder reset	LD, LiteST
	ENC_Preset	Encoder preset	LD, LiteST
	ENC_TouchProbe	Encoder probe	LD, LiteST
	ENC_ArrayCompare	Encoder 1D array comparison	LD, LiteST
	ENC_StepCompare	Encoder 1D Step Size Comparison	LD, LiteST
	ENC_GroupArrayCompare	Encoder 2D array comparison	LD, LiteST
	ENC_ReadStatus	Encoder status acquisition	LD, LiteST
	ENC_DigitalOutput	Encoder digital output control	LD, LiteST
	ENC_ResetCompare	Encoder reset comparison output	LD, LiteST
	ENC_SetUnit	Set shaft gear ratio	LD, LiteST
	ENC_SetLineRotationMode	Set the operating mode of the axis	LD, LiteST
Encoder axis	ENC_Counter	Encoder enable	LD, LiteST
	ENC_Reset	Encoder reset	LD, LiteST
	ENC_Preset	Encoder preset	LD, LiteST
	ENC_TouchProbe	Encoder probe	LD, LiteST
	ENC_ArrayCompare	Encoder 1D array comparison	LD, LiteST
	ENC_StepCompare	Encoder 1D Step Size Comparison	LD, LiteST
	ENC_Compare	Single point compare output	LD, LiteST
	ENC_GroupArrayCompare	Encoder 2D array comparison	LD, LiteST

Command category	Name	Description	Supported Language
	ENC_ReadStatus	Encoder status acquisition	LD, LiteST
	ENC_DigitalOutput	Encoder digital output control	LD, LiteST
	ENC_ResetCompare	Encoder reset comparison output	LD, LiteST
	ENC_SetUnit	Set shaft gear ratio	LD, LiteST
	ENC_SetLineRotationMode	Set the operating mode of the axis	LD, LiteST
Timer	TPR	Pulse timer	LD, LiteST
	TONR	On-delay timer	LD, LiteST
	TOFR	Off-delay timer	LD, LiteST
	TACR	Time accumulation timer	LD, LiteST
Pointer instruction	PTGET	pointer variable assignment instruction	LD
	PTINC	Pointer variable address increment instruction	LD
	PTDEC	Needle variable address minus 1 instruction	LD
	PTADD	Pointer variable address addition instruction	LD
	PTSUB	Pointer variable address subtraction instruction	LD
	PTSET	pointer variable assignment instruction	LD
	PTMOV	Mutual assignment of pointer variables	LD
	PTLD>	Pointer variable contacts compare greater than instruction	LD
	PTLD>=	Pointer variable contact comparison greater than or equal to instruction	LD
	PTLD<=	Pointer variable contact comparison less than or equal to instruction	LD
	PTLD=	Pointer variable contact compare equals instruction	LD
	PTLD<>	pointer variable contact compare not equal instruction	LD
	PTAND>	Pointer variable and contact compare greater than instruction	LD
	PTAND>=	Pointer variable and contact comparison greater than or equal to instruction	LD
	PTAND<	Pointer variable compared with contact less than instruction	LD
	PTAND<=	Pointer variable and contact comparison less than or equal to instruction	LD
	PTAND=	Pointer variable and contact compare equals instruction	LD
	PTAND<>	Pointer variable compared to contact not equal to instruction	LD
	PTOR>	Pointer variable or contact compares greater than instruction	LD
	PTOR>=	Pointer variable or contact comparison greater than or equal to instruction	LD

Command category	Name	Description	Supported Language
	PTOR<	Pointer variable or contact compare less than instruction	LD
	PTOR<=	Pointer variable or contact comparison less than or equal to instruction	LD
	PTOR=	Pointer variable or contact compare equals instruction	LD
	PTOR<>	Pointer variable or contact compares not equal to instruction	LD
FB/FC instruction	PROG_AUTH	Program block (FB/FC, etc.) authorization verification instruction	LD
Communication protocol instruction	SerialSR	Serial free protocol sending and receiving	LD
	TCP_Listen	TCP listen command	LD
	TCP_Accept	TCP accept connection request instruction	LD
	TCP_Connect	TCP initiates connection request command	LD
	TCP_Close	TCP close connection command	LD
	TCP_Send	TCP send data command	LD
	TCP_Receive	TCP receive data command	LD
	UDP_Bind	UDP socket binding instructions	LD
	UDP_Send	UDP send data command	LD
	UDP_Receive	UDP receive data instruction	LD
	ETC_ReadParame-ter_CoE	Read the SDO parameters of the ETC_RestartMaster slave	LD, LiteST
	ETC_WriteParame-ter_CoE	Write to the SDO parameters of the ETC_RestartMaster slave	LD, LiteST
	ETC_RestartMaster	Restart the EtherCAT master	LD, LiteST
	EIP_Generic_Service	Call the "generic" service	LD
	EIP_Get_Attributes_All	Call the "Get All Properties" service	LD
	EIP_Get_Attribute_Single		LD
	EIP_Set_Attributes_All		LD
	EIP_Set_Attribute_Single		LD
	EIP_Apply_Attributes		LD
	EIP_NOP		LD
	EIP_Reset		LD
	EIP_Start		LD
	EIP_Stop		LD
EtherCAT motion control axis	MC_Power		LD, LiteST
	MC_Reset		LD, LiteST
	MC_ReadStatus		LD, LiteST
	MC_ReadAxisError		LD, LiteST
	MC_ReadDigitalInput		LD, LiteST
	MC_ReadActualPosition		LD, LiteST
	MC_ReadActualVelocity		LD, LiteST
	MC_ReadActualTorque		LD, LiteST
	MC_SetPosition		LD, LiteST
	MC_TouchProbe		LD, LiteST

Command category	Name	Description	Supported Language
	MC_MoveRelative		LD, LiteST
	MC_MoveAbsolute		LD, LiteST
	MC_MoveVelocity		LD, LiteST
	MC_Jog		LD, LiteST
	MC_TorqueControl		LD, LiteST
	MC_Home		LD, LiteST
	MC_Stop		LD, LiteST
	MC_Halt		LD, LiteST
	MC_MoveFeed		LD, LiteST
	MC_MoveBuffer		LD, LiteST
	MC_ImmediateStop		LD, LiteST
	MC_MoveSuperImposed		LD, LiteST
	MC_MoveVelocityCSV		LD, LiteST
	MC_SyncMoveVelocity		LD, LiteST
	MC_FollowVelocity		LD, LiteST
	MC_SyncTorqueControl		LD, LiteST
	MC_SetAxisConfigPara		LD, LiteST
Electronic CAM command	MC_CamIn		LD, LiteST
	MC_CamOut		LD, LiteST
	MC_GetCamTablePhase		LD, LiteST
	MC_GetCamTableDistance		LD, LiteST
	MC_DigitalCamSwitch		LD, LiteST
	MC_GearIn		LD, LiteST
	MC_GearOut		LD, LiteST
	MC_Phasing		LD, LiteST
	MC_SaveCamTable		LD, LiteST
	MC_GenerateCamTable		LD, LiteST
Axes group control	MC_MoveLinear		LD, LiteST
	MC_MoveCircular		LD, LiteST
	MC_MoveEllipse		LD, LiteST
	MC_GroupStop		LD, LiteST
	MC_GroupPause		LD, LiteST
CANopen Motion Control Axis Command	MC_Power_CO		LD
	MC_Reset_CO		LD
	MC_ReadActualPosition_CO		LD
	MC_ReadActualVelocity_CO		LD
	MC_Halt_CO		LD
	MC_Stop_CO		LD
	MC_MoveAbsolute_CO		LD
	MC_MoveRelative_CO		LD
	MC_MoveVelocity_CO		LD
	MC_Jog_CO		LD
	MC_Home_CO		LD
	MC_WriteParameter_CO		LD
	MC_ReadParameter_CO		LD
Other instructions	PID		LD

7.1 LITEST LANGUAGE EXCLUSIVE INSTRUCTIONS

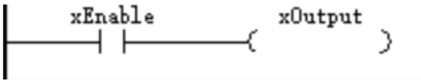
Command category	Name	Description	Supported Language
Trigonometric functions	SIN	Sine Operation Instruction	LiteST
	COS	cosine operation instruction	LiteST
	TAN	Tangent operation instruction	LiteST
	ASIN	Inverse Sine Operation Instruction	LiteST
	ACOS	Inverse cosine instruction	LiteST
	ATAN	Arctangent operation instruction	LiteST
Exponential operation	LOG	Base 10 logarithm	LiteST
	LN	Logarithmic operation with base e(2.71828)	LiteST
	SQRT	Square root operation instruction	LiteST
	EXPT	Power instruction	LiteST
Explicit conversion	INT_TO_DINT	Convert INT type to DINT type	LiteST
	INT_TO_REAL	Convert INT type to REAL type	LiteST
	INT_TO_BOOL	Convert INT type to BOOL type	LiteST
	DINT_TO_INT	Convert DINT type to INT type	LiteST
	DINT_TO_REAL	Convert DINT type to REAL type	LiteST
	DINT_TO_BOOL	Convert DINT type to BOOL type	LiteST
	BOOL_TO_INT	Convert BOOL type to INT type	LiteST
	BOOL_TO_DINT	Convert BOOL type to DINT type	LiteST
	BOOL_TO_REAL	Convert BOOL type to REAL type	LiteST
	REAL_TO_INT	Convert REAL type to INT type	LiteST
	REAL_TO_DINT	Convert REAL type to DINT type	LiteST
	REAL_TO_BOOL	Convert REAL type to BOOL type	LiteST
	TO_REAL	Convert variable to REAL type	LiteST
	TO_INT	Convert variable to INT type	LiteST
	TO_DINT	Convert variable to DINT type	LiteST
	TO_BOOL	Convert variable to BOOL type	LiteST
Comparison instruction	MAX	Take the larger value operation	LiteST
	MIN	Take the smaller value operation	LiteST
Shift instruction	SHL	Left shift operation	LiteST
	SHR	Right shift operation	LiteST
Select instruction	SEL	Select operation	LiteST
Absolute value operation instruction	ABS	Absolute value operation	LiteST
Bit operation	AND	And operation	LiteST
	OR	Or operation	LiteST
	XOR	XOR operation	LiteST
	NOT	Negate operation	LiteST

8 FROM LADDER TO ST-PROGRAMMING

This section describes how a LADDER program sequence (LD) can be translated into Structured Programming (ST).

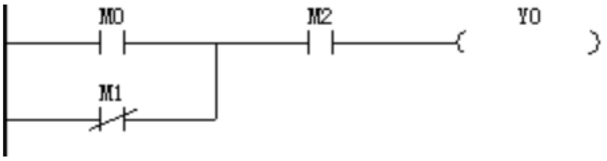
AutoShop does not have any tools to translate an LD program to a structured code program, so this section describes the steps necessary to translate the basic instructions from LD to ST.

8.1 EXAMPLE 1: SIMPLE CONTACT



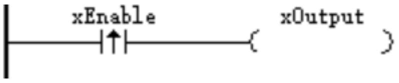
```
//Option 1
xOutput:=xEnable;

//Option 2
IF xEnable THEN
  xOutput:= TRUE;
ELSE
  xOutput:= FALSE;
END_IF;

//Option 3
IF xEnable=TRUE THEN xOutput:= TRUE; ELSE xOutput:= FALSE; END_IF;
```

8.2 EXAMPLE 2: SERIE AND PARALLEL CONTACTS



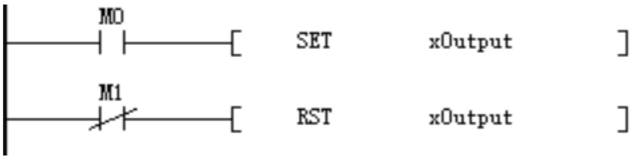
```
//Option 1
Y0:=(M0 OR NOT M1) AND M2;

//Option 2
IF (M0 OR NOT M1) AND M2 THEN
    Y0:= TRUE;
ELSE
    Y0:= FALSE;
END_IF;

//Option 3
IF (M0=TRUE OR M1=FALSE) AND M2=TRUE THEN Y0:= TRUE; ELSE Y0:= FALSE; END_IF;
```

8.3 EXAMPLE 3: RISING EDGE



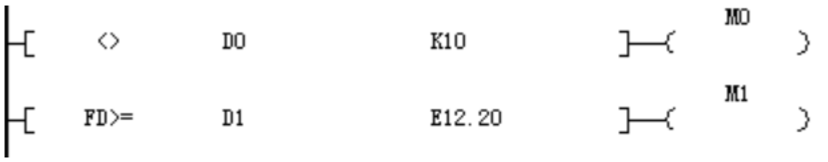
```
//Option 1
RisingEdge(CLK := xEnable);
IF RisingEdge.Q THEN
    xOutput:= TRUE;
ELSE
    xOutput:= FALSE;
END_IF;

//Option 2
RisingEdge(CLK := xEnable, Q => xOutput);
```

8.4 EXAMPLE 4: SET/RESET



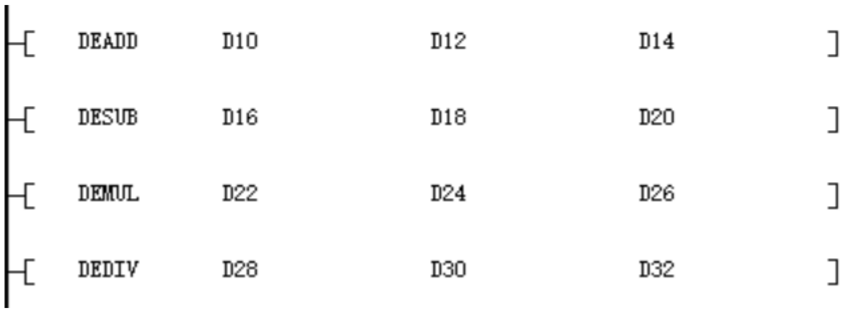
```
//Option 1
IF M0=TRUE THEN
    xOutput := TRUE;
END_IF;
IF M1=FALSE THEN
    xOutput := FALSE;
END_IF;

//Option 2
IF M0 THEN xOutput := TRUE; END_IF;
IF NOT M1 THEN xOutput := FALSE; END_IF;
```

8.5 EXAMPLE 5: COMPARISON



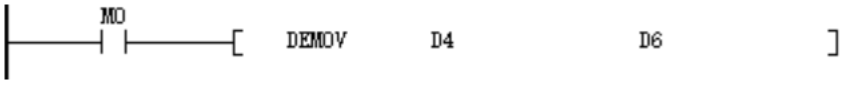
```
//Option 1
IF D0<>10 THEN
    M0 := TRUE;
ELSE
    M0 := FALSE;
END_IF;
IF D1>=12.20 THEN
    M1 := TRUE;
ELSE
    M1 := FALSE;
END_IF;

//Option 2
IF D0<>10 THEN M0 := TRUE; ELSE M0 := FALSE; END_IF;
IF D1>=12.20 THEN M1 := TRUE; ELSE M1 := FALSE; END_IF;
```

8.6 EXAMPLE 6: ARITHMETIC OPERATIONS



```
D14 := D10 + D12;
D20 := D16 - D18;
D26 := D22 x D24;
D32 := D28 / D30;
```

8.7 EXAMPLE 7: MOVE

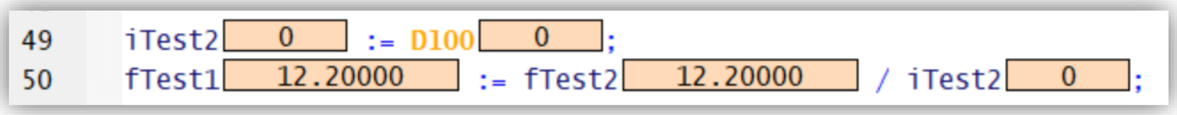


```
IF M0 THEN
    D6 := D4;
END_IF;
```

9 EXCEPTIONS

9.1 DIVISION BY 0 (ER5081)

If in the program code there is an instruction that performs division by zero, the PLC will go into fault mode and display error **5081**. The PLC display will alternatively show the values Er 50 81.



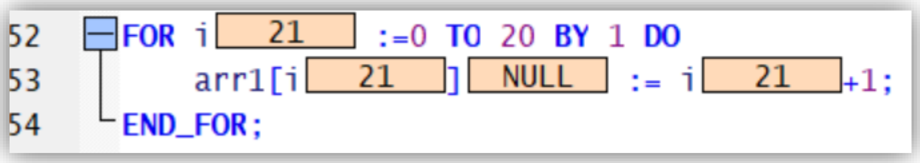
```
iTest2 0 := D100 0 ;
fTest1 12.20000 := fTest2 12.20000 / iTest2 0 ;
```

9.2 ARRAY OUT OF BOUNDS (ER5080)

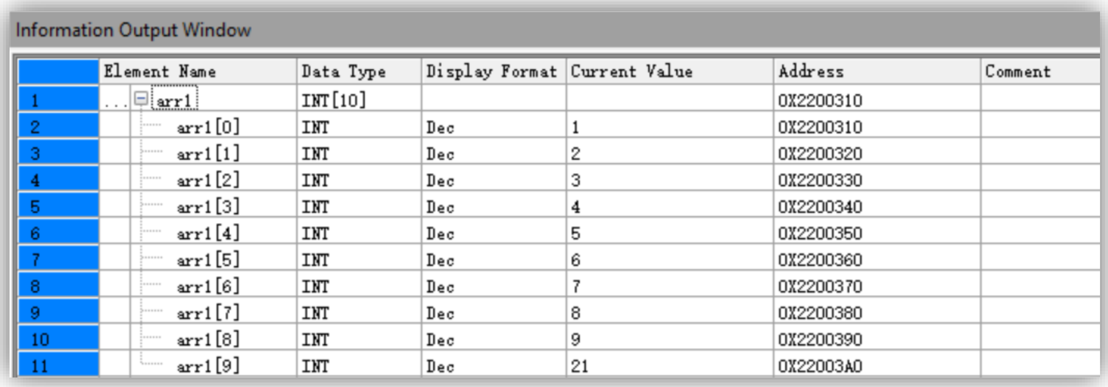
The PLC checks if the matrix is out of limits. Constants are checked at compile time and variables are checked at run time. If in the program exceeds the array limits, the PLC will go into fault mode and display error **5080**. The PLC display will alternatively show the values Er 50 80.

When the upper bound is exceeded, the out-of-bounds value is stored in the element with the highest index in the array. When the lower bound is exceeded, the out-of-bounds value is stored in the array with index 0.

As shown in the following figure: Set the maximum number of elements in the array of type INT to 10. After the limit is exceeded, the software will report an error and store the limit value in the element with the largest subscript (9) in the matrix.



```
FOR i 21 :=0 TO 20 BY 1 DO
    arr1[i 21 ] NULL := i 21 +1;
END_FOR;
```



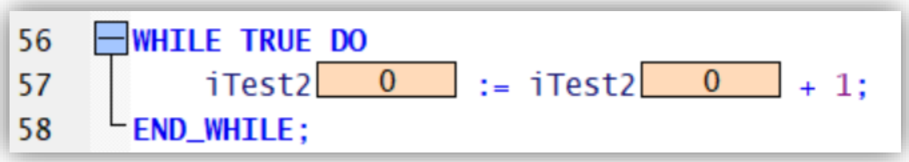
Information Output Window

	Element Name	Data Type	Display Format	Current Value	Address	Comment
1	arr1	INT[10]			0X2200310	
2	arr1[0]	INT	Dec	1	0X2200310	
3	arr1[1]	INT	Dec	2	0X2200320	
4	arr1[2]	INT	Dec	3	0X2200330	
5	arr1[3]	INT	Dec	4	0X2200340	
6	arr1[4]	INT	Dec	5	0X2200350	
7	arr1[5]	INT	Dec	6	0X2200360	
8	arr1[6]	INT	Dec	7	0X2200370	
9	arr1[7]	INT	Dec	8	0X2200380	
10	arr1[8]	INT	Dec	9	0X2200390	
11	arr1[9]	INT	Dec	21	0X22003A0	

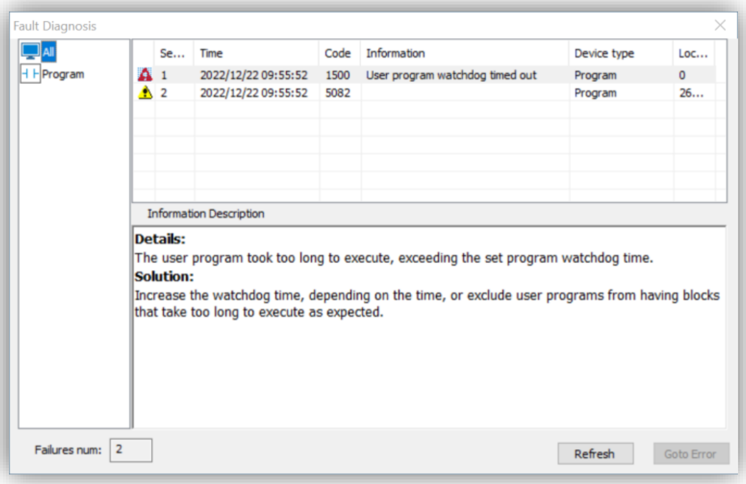
9.3 INFINITE LOOP (ER1500)

The PLC software verifies the infinite cycle of the program at execution time. If the PLC detects an infinite loop, it will report an error message and automatically jump out of the loop. The program will report error **1500** and **5082** and stop running. The PLC display will alternatively show the values Er 15 00 50 82.

The **1500** error is the watchdog timeout, and the **5082** error is the infinite cycle.



```
WHILE TRUE DO
    iTest2 := iTest2 + 1;
END_WHILE;
```



	Se...	Time	Code	Information	Device type	Loc...
⚠	1	2022/12/22 09:55:52	1500	User program watchdog timed out	Program	0
⚠	2	2022/12/22 09:55:52	5082		Program	26...

Details:
The user program took too long to execute, exceeding the set program watchdog time.
Solution:
Increase the watchdog time, depending on the time, or exclude user programs from having blocks that take too long to execute as expected.

Failures num: 2

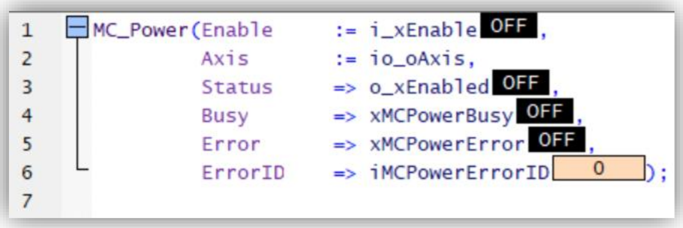
10 IEC-61131-3 COMPATIBILITY

LiteST is a structured language implementation based on the IEC 61131-3 standard. This implementation does not comply 100% with the rules of the standard but most of FC and FB conform to the norm.

Some of the functions that differ from the IEC 61131-3 standard are described below.

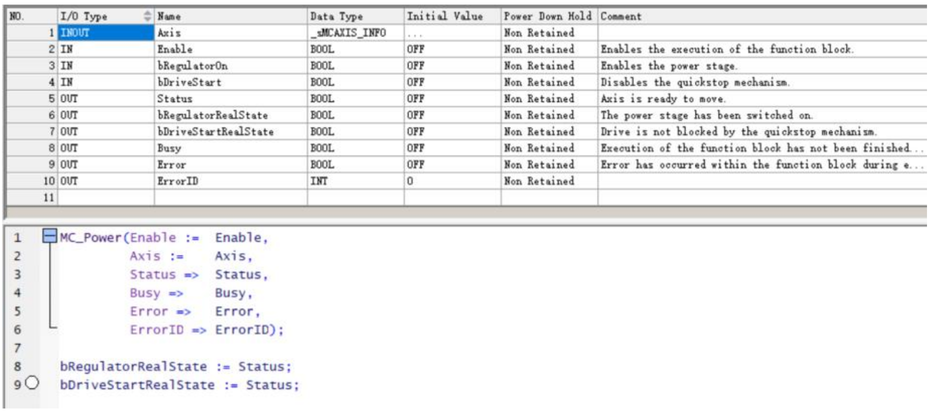
10.1 FB INSTANTIATION

All motion FBs do not need an instance. The name of the motion block itself serves as an instance. In order to access the state of the block, it is necessary to use intermediate variables.



```
MC_Power(Enable      := i_xEnable OFF ,
         Axis        := io_oAxis,
         Status      => o_xEnabled OFF ,
         Busy        => xMCPowerBusy OFF ,
         Error       => xMCPowerError OFF ,
         ErrorID     => iMCPowerErrorID 0 );
```

To be able to use instances with these FBs, you can create an FB that replicates the behavior:



NO.	I/O Type	Name	Data Type	Initial Value	Power Down Hold	Comment
1	INOUT	Axis	_sMCAXIS_INFO	...	Non Retained	
2	IN	Enable	BOOL	OFF	Non Retained	Enables the execution of the function block.
3	IN	bRegulatorOn	BOOL	OFF	Non Retained	Enables the power stage.
4	IN	bDriveStart	BOOL	OFF	Non Retained	Disables the quickstop mechanism.
5	OUT	Status	BOOL	OFF	Non Retained	Axis is ready to move.
6	OUT	bRegulatorRealState	BOOL	OFF	Non Retained	The power stage has been switched on.
7	OUT	bDriveStartRealState	BOOL	OFF	Non Retained	Drive is not blocked by the quickstop mechanism.
8	OUT	Busy	BOOL	OFF	Non Retained	Execution of the function block has not been finished...
9	OUT	Error	BOOL	OFF	Non Retained	Error has occurred within the function block during e...
10	OUT	ErrorID	INT	0	Non Retained	
11						

```
MC_Power(Enable :=  Enable,
         Axis :=    Axis,
         Status =>  Status,
         Busy =>    Busy,
         Error =>   Error,
         ErrorID => ErrorID);

bRegulatorRealState := Status;
bDriveStartRealState := Status;
```

NOTE: The Timer functions also don’t need an instance to be used in programs.

Timer	TPR	Pulse timer
	TONR	On-delay timer
	TOFR	Off-delay timer
	TACR	Time accumulation timer

10.2 SUPPORTED DATA TYPES

The following table shows the data supported by the LiteST and those that are not supported by the IEC 61131-3 standard:

Keyword	Data type	Bit size	LiteST
BOOL	Boolean	1	Supported
SINT	Short integer	8	Not supported
INT	Integer	16	Supported
DINT	Double integer	32	Supported
LINT	Long integer	64	Not supported
USINT	Unsigned short integer	8	Not supported
UINT	Unsigned integer	16	Not supported
UDINT	Unsigned double integer	32	Not supported
ULINT	Unsigned long integer	64	Not supported
REAL	Real numbers	32	Supported
LREAL	Long reals	64	Not supported
TIME	Duration		Not supported
DATE	Date (only)		Not supported
TIME_OF_DAY or TOD	Time of day (only)		Not supported
DATE_AND_TIME or DT	Date and time of Day		Not supported
STRING	Variable-length single-byte character string		Not supported
WSTRING	Variable-length double-byte character string		Not supported
CHAR	Single-byte character		Not supported
WCHAR	Double-byte character		Not supported
BYTE	Bit string of length 8	8	Not supported
WORD	Bit string of length 16	16	Not supported
DWORD	Bit string of length 32	32	Not supported
LWORD	Bit string of length 64	64	Not supported
ANY	Generic Data Types		Not supported

NOTE : The **REAL** type complies with the standard **IEEE-754** Floating Point **32-bits single precision** representation. For example, a single precision floating point number has a maximum of 7 decimal significant digits. If the floating-point number 1234567.89 is transferred to the D0 register, the value of D0 is 1234567.9.

10.3 STANDARD FUNCTIONS

Category		Function	Description	LiteST
Numerical and arithmetic functions	General functions	ABS	Absolute value	Supported
		SQRT	Square root	Supported
	Logarithmic functions	LN	Natural logarithm	Supported
		LOG	Logarithm base 10	Supported
		EXP	Natural exponential	Supported [EXPT]
	Trigonometric functions	SIN	Sine of input in radians	Supported
		COS	Cosine in radians	Supported
		TAN	Tangent in radians	Supported
		ASIN	Principal arc sine	Supported
		ACOS	Principal arc cosine	Supported
		ATAN	Principal arc tangent	Supported
Arithmetic functions	Extensible arithmetic functions	+	Addition	Supported
		*	Multiplication	Supported
	Non-extensible arithmetic functions	-	Subtraction	Supported
		/	Division	Supported
		MOD	Modulo	Supported
		EXP	Exponentiation	Supported
		:=	Move	Supported
Bit shift functions		SHL	Left-shifted by N bits, zero-filled on right	Supported
		SHR	Right-shifted by N bits, zero-filled on left	Supported
		ROR	Right-rotated by N bits, circular	Not supported
		ROL	Left-rotated by N bits, circular	Not supported
Bitwise Boolean functions		And (&)	AND operand	Supported [AND]
		Or (>=1)	OR operand	Supported [OR]
		Exclusive Or	XOR operand	Supported [XOR]
		Not	Invert operand	Supported [NOT]
Selection functions		MOVE	Move (assignment)	Supported
		SEL	Binary selection	Supported
		MAX	Extensible maximum function	Supported
		MIN	Extensible minimum function	Supported
		LIMIT	Limiter	Supported
		MUX	Extensible multiplexer	Not supported
Comparison functions		GT	Decreasing sequence [>]	Supported
		GE	Monotonic sequence [>=]	Supported

Category		Function	Description	LiteST
		EQ	Equality [=]	Supported
		LE	Monotonic sequence [<=]	Supported
		LT	Increasing sequence [<]	Supported
		NE	Inequality [<>]	Supported
Character string functions		LEN	String length	Not supported
		LEFT	Left	Not supported
		RIGHT	Right	Not supported
		MID	Middle	Not supported
		CONCAT	Extensible concatenation	Not supported
		INSERT	Insert	Not supported
		DELETE	Delete	Not supported
		REPLACE	Replace	Not supported
		FIND	Find	Not supported
Numerical functions of time and duration data types		ADD		Not supported
		ADD_TIME		Not supported
		ADD_LTIME		Not supported
		ADD		Not supported
		ADD_TOD_TIME		Not supported
		ADD_LTOD_LTIME		Not supported
		ADD		Not supported
		ADD_DT_TIME		Not supported
		ADD_LDT_LTIME		Not supported
		SUB		Not supported
		SUB_TIME		Not supported
		SUB_LTIME		Not supported
		SUB		Not supported
		SUB_DATE_DATE		Not supported
		SUB_LDATE_LDATE		Not supported
		SUB		Not supported
		SUB_TOD_TIME		Not supported
		SUB_LTOD_LTIME		Not supported
		SUB		Not supported
		SUB_TOD_TOD		Not supported
		SUB_TOD_TOD		Not supported
		SUB		Not supported
		SUB_DT_TIME		Not supported
		SUB_LDT_LTIME		Not supported
		SUB		Not supported
		SUB_DT_DT		Not supported
		SUB_LDT_LDT		Not supported
		MUL		Not supported
		MUL_TIME		Not supported
		MUL_LTIME		Not supported
		DIV		Not supported
		DIV_TIME		Not supported
		DIV_LTIME		Not supported

10.4 STANDARD FUNCTION BLOCKS

Category	FB	Description	LiteST
Bistable elements	SR	Bistable function block (set dominant)	Not supported
	RS	Bistable function block (reset dominant)	Not supported
Edge detection	R_TRIG	Rising edge detector	Supported
	F_TRIG	Falling edge detector	Supported
Counters	CTU	Up counter. INT values	Not supported
	CTU_DINT	Up counter. DINT values	Not supported
	CTU_LINT	Up counter. LINT values	Not supported
	CTU_UDINT	Up counter. UDINT values	Not supported
	CTU_ULINT	Up counter. ULINT values	Not supported
	CTD	Down counter. INT values	Not supported
	CTD_DINT	Down counter. DINT values	Not supported
	CTD_LINT	Down counter. LINT values	Not supported
	CTD_UDINT	Down counter. UDINT values	Not supported
	CTD_ULINT	Down counter. ULINT values	Not supported
	CTUD	Up-down counter. INT values	Not supported
	CTUD_DINT	Up-down counter. DINT values	Not supported
	CTUD_LINT	Up-down counter. LINT values	Not supported
	CTUD_UDINT	Up-down counter. UDINT values	Not supported
	CTUD_ULINT	Up-down counter. ULINT values	Not supported
Timers	TP	Pulse (TP) timing	Supported [TPR]
	TON	On-delay (TON) timing	Supported [TONR]
	TOF	Off-delay (TOF) timing	Supported [TOFR]