# Kollmorgen Automation Suite

## KAS Reference Manual – PLC Library


**Document Edition: U, December 2021**
Valid for KAS Software Revision 3.07
Part Number: 959717

For safe and proper use, follow these instructions. Keep for future use.

# Trademarks and Copyrights

## Copyrights



## Trademarks

- KAS and AKD are registered trademarks of Kollmorgen.
- Kollmorgen is part of the Altra Industrial Motion Company.
- EnDat is a registered trademark of Dr. Johannes Heidenhain GmbH
- EtherCAT is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH
- Ethernet/IP is a registered trademark of ODVA, Inc.
- Ethernet/IP Communication Stack: copyright (c) 2009, Rockwell Automation
- HIPERFACE is a registered trademark of Max Stegmann GmbH
- PROFINET is a registered trademark of PROFIBUS and PROFINET International (PI)
- SIMATIC is a registered trademark of SIEMENS AG
- Windows is a registered trademark of Microsoft Corporation
- PLCopen is an independent association providing efficiency in industrial automation.
- Codemeter is a registered trademark of WIBU-Systems AG.
- SyCon® is a registered trademark of Hilscher GmbH.

Kollmorgen Automation Suite is based on the work of:

- 7-zip (distributed under the terms of the LGPL and the BSD 3-clause licenses - see terms)
- The C++ Mathematical Expression Library (distributed under the MIT License)
- curl software library
- JsonCpp software (distributed under the MIT License – see terms)
- Mongoose software (distributed under the MIT License - see terms)
- Qt cross-platform SDK (distributed under the terms of the LGPL3; Qt source is available on KDN)
- Qwt project (distributed under the terms of the Qwt License)
- U-Boot, a universal boot loader is used by the AKD PDMM and PCMM (distributed under the terms of the GNU General Public License). The U-Boot source files, copyright notice, and readme are available on the distribution disk that is included with the AKD PDMM and PCMM.
- Zlib software library

All other product and brand names listed in this document may be trademarks or registered trademarks of their respective owners.

## Disclaimer

The information in this document (Version U published on 11/30/2021) is believed to be accurate and reliable at the time of its release. Notwithstanding the foregoing, Kollmorgen assumes no responsibility for any damage or loss resulting from the use of this help, and expressly disclaims any liability or damages for loss of data, loss of use, and property damage of any kind, direct, incidental or consequential, in regard to or arising out of the performance or form of the materials presented herein or in any software programs that accompany this document.

All timing diagrams, whether produced by Kollmorgen or included by courtesy of the PLCopen organization, are provided with accuracy on a best-effort basis with no warranty, explicit or implied, by Kollmorgen. The user releases Kollmorgen from any liability arising out of the use of these timing diagrams.

# 1 Table of Contents

# 2 Programming Languages

This chapter presents details on the syntax, structure and use of the declarations and statements supported by the KAS IDE application language.

Below are the available programming languages of the IEC 61131-3 standard:

- Sequential Function Chart (SFC)
- Function Block Diagram (FBD)
- Free Form Ladder Diagram (FFLD)
- Structured Text (ST)
- Instruction List (IL)

You have to select a language for each program or User-Defined Function Block of the application.

**NOTE**

Replace with note text When using FFLD or FBD languages, be sure to review Use of ST expressions in graphic language.

## 2.1 Sequential Function Chart (SFC)

The SFC language is a state diagram. Graphical steps are used to represent stable states, and transitions describe the conditions and events that lead to a change of state. Using SFC highly simplifies the programming of sequential operations as it saves a lot of variables and tests just for maintaining the program context.

**IMPORTANT**

You must not use SFC as a decision diagram. Using a step as a point of decision and transitions as conditions in an algorithm must never appear in an SFC chart. Using SFC as a decision language leads to poor performance and complicate charts. ST must be preferred when programming a decision algorithm that has no sense in term of "program state".

Below are basic components of an SFC chart:

| Chart | Programming |
|---|---|
| Steps and initial steps<br>Transitions and divergences<br>Parallel branches<br>Macro-steps<br>Jump to a step | Actions within a step<br>Timeout on a step<br>Programming a transition condition<br><br>How SFC is executed |

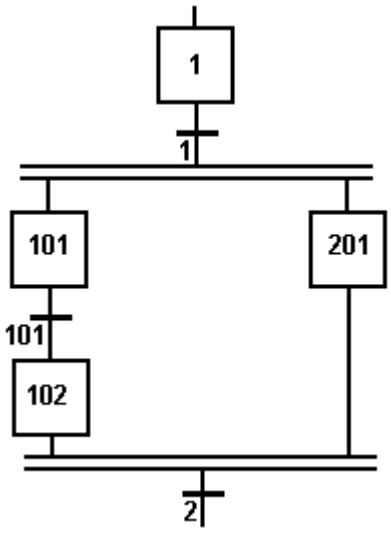
The KAS IDE fully supports SFC programming with several hierarchical levels of charts: i.e. a chart that controls another chart. Working with a hierarchy of SFC charts is an easy and powerful way for managing complex sequences and saves performances at runtime. Refer to the following sections for further details:

- Hierarchy of SFC programs
- Controlling a SFC child program

### 2.1.1 SFC Execution at Runtime

SFC programs are executed sequentially within a target cycle, according to the order defined when entering programs in the hierarchy tree. A parent SFC program is executed before its children. This implies that when a parent starts or stops a child, the corresponding actions in the child program are performed during the same cycle.

Within a chart, all valid transitions are evaluated first, and then actions of active steps are performed. The chart is evaluated from the left to the right and from the top to the bottom. Below is an example:




Execution order:

- Evaluate transitions:

  1, 101, 2

- Manage steps:

  1, 101, 201, 102

In case of a divergence, all conditions are considered as exclusive, according to a "left to right" priority order. It means that a transition is considered as FALSE if at least one of the transitions connected to the same divergence on its left side is TRUE.

The initial steps define the initial status of the program when it is started. All top level (main) programs are started when the application starts. Child programs are explicitly started from action blocks within the parent programs.

The evaluation of transitions leads to changes of active steps, according to the following rules:

- A transition in crossed if:
  - its condition is TRUE
  - and if all steps linked to the top of the transition (before) are active
- When a transition is crossed:
  - all steps linked to the top of the transition (before) are deactivated
  - all steps linked to the bottom of the transition (after) are activated

**IMPORTANT**

Execution of SFC within the IEC 61131 target is sampled according to the target cycles. When a transition is crossed within a cycle, the following steps are activated, and the evaluation of the chart will continue on the next cycle. If several consecutive transitions are TRUE within a branch, only one of them is crossed within one target cycle.

**IMPORTANT**

Some run-time systems can support exclusivity of the transitions within a ulergence or not. Please refer to OEM instructions for further information about SFC support.

### 2.1.2 Hierarchy of SFC programs

Each SFC program can have one or more "child programs". Child programs are written in SFC and are started (launched) or stopped (killed) in the actions of the father program. A child program can also have children. The number of hierarchy levels must not exceed 19.

When a child program is stopped, its children are also implicitly stopped.

When a child program is started, it must explicitly in its actions start its children.

A child program is controlled (started or stopped) from the action blocks of its parent program. Designing a child program is a simple way to program an action block in SFC language.

Using child programs is very useful for designing a complex process and separate operations due to different aspects of the process. For instance, it is common to manage the execution modes in a parent program and to handle details of the process operations in child programs.

### 2.1.3 Controlling a SFC child program

Controlling a child program can be simply achieved by specifying the name of the child program as an action block in a step of its parent program. Below are possible qualifiers that can be applied to an action block for handling a child program:

| | |
|---|---|
| `Child (N);` | Starts the child program when the step is activated and stops (kills) it when the step is deactivated. |
| `Child (S);` | Starts the child program when the step is activated (Initial steps of the child program are activated) |
| `Child (R);` | Stops (kills) the child program when the step is activated (All active steps of the child program are deactivated) |

Alternatively, you can use the following statements in an action block programmed in ST language. In the following table, "`prog`" represents the name of the child program:

| | |
|---|---|
| `GSTART (prog);` | Starts the child program when the step is activated (Initial steps of the child program are activated) |
| `GKILL (prog);` | Stops (kills) the child program when the step is activated (All active steps of the child program are deactivated) |
| `GFREEZE (prog);` | Suspends the execution of a child program |
| `GRST (prog);` | Restarts a program suspended by a `GFREEZE`command. |

You can also use the "GSTATUS" function in expressions. This function returns the current state of a child SFC program:

| | |
|---|---|
| `GSTATUS (prog)` | Returns the current state of a child SFC program:<br>0: program is inactive<br>1: program is active<br>2: program is suspended |

**NOTE**

When a child program is started by its parent program, it keeps the "inactive" status until it is executed (further in the cycle). If you start a child program in an SFC chart, GSTATUS will return 1 (active) on the next cycle.

## 2.2 Function Block Diagram (FBD)

A function block Diagram is a data flow between constant expressions or variables and operations represented by rectangular blocks. Operations can be basic operations, function calls, or function block calls.

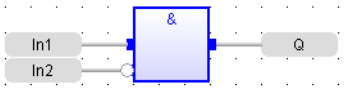
Use of ST instructions in graphic languages

The name of the operation or function, or the type of function block is written within the block rectangle. In case of a function block call, the name of the called instance is written in the header of the block rectangle, such as in the example below:

### 2.2.1 Data flow

The data flow represents values of any data type. All connections must be from input and outputs points having the same data type.

In case of a Boolean connection, you can use a connection link terminated by a small circle, that indicates a Boolean **negation** of the data flow. .



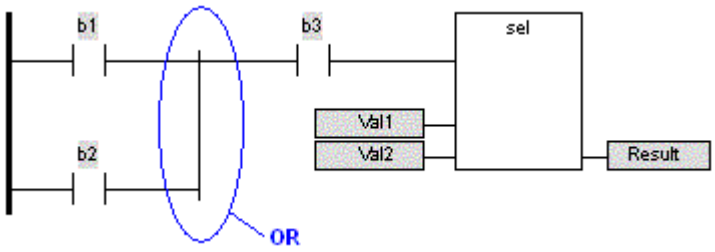

The data flow must be understood from the left to the right and from the top to the bottom. It is possible to use labels and jumps to change the default data flow execution.

### 2.2.2 FFLD symbols

FFLD symbols can also be entered in FBD diagrams and linked to FBD objects. Refer to the following sections for further information about components of the FFLD language:

Contacts
Coils
Power Rails

Special vertical lines are available in FBD language for representing the merging of FFLD parallel lines. Such vertical lines represent a OR operation between the connected inputs. Below is an example of an OR vertical line used in a FBD diagram:




## 2.3 Instruction List (IL)

This language is more appropriate when your algorithm refers to the Boolean algebra.

A program written in IL language is a list of *instructions*.

- Each instruction is written on one line of text.
- An instruction can have one or more *operands*.
- Operands are variables or constant expressions.
- Each instruction can begin with a label, followed by the ":" character.
- Labels are used as destination for jump instructions.

The KAS IDE allows you to mix ST and IL languages in textual program. ST is the default language. When you enter IL instructions, the program must be entered between "BEGIN_IL" and "END_IL" keywords, such as in the following example

```
BEGIN_IL
FFLD   var1
ST     var2
END_IL
```

### 2.3.1 Comments

Comment texts can be entered at the end of a line containing an instruction. Comment texts have no meaning for the execution of the program. A comment text must begin with "**(***" and end with "***)**". Comments can also be entered on empty lines (with no instruction), and on several lines (i.e. a comment text can include line breaks). Comment texts cannot be nested.

```
(* My comment *)
LD a
ST b   (* Store value in d *)
```

### 2.3.2 Data flow

An IL complete statement is made of instructions for:

- first: evaluating an expression (called *current result*)
- then: use the current result for performing actions

### 2.3.3 Evaluation of expressions

The order of instructions in the program is the one used for evaluating expressions, unless parentheses are inserted. Below are the available instructions for evaluation of expressions:

| Instruction | Operand | Meaning |
|---|---|---|
| FFLD / FFLDN | any type | loads the operand in the current result |
| AND (&) | Boolean | AND between the operand and the current result |
| OR / ORN | Boolean | OR between the operand and the current result |
| XOR / XORN | Boolean | XOR between the operand and the current result |
| ADD | numerical | adds the operand and the current result |
| SUB | numerical | subtract the operand from the current result |
| MUL | numerical | multiply the operand and the current result |
| DIV | numerical | divide the current result by the operand |
| GT | numerical | compares the current result with the operand |
| GE | numerical | compares the current result with the operand |
| LT | numerical | compares the current result with the operand |
| LE | numerical | compares the current result with the operand |
| EQ | numerical | compares the current result with the operand |
| NE | numerical | compares the current result with the operand |
| Function call | func. arguments | calls a function |
| Parenthesis | | changes the execution order |

**NOTE**

Instructions suffixed by **N** uses the Boolean negation of the operand.

### 2.3.4 Actions

The following instructions perform actions according to the value of current result. Some of these instructions do not need a current result to be evaluated:

| Instruction | Operand | Meaning |
|---|---|---|
| ST / STN | any type | stores the current result in the operand |
| JMP | label | jump to a label - no current result needed |
| JMPC | label | jump to a label if the current result is TRUE |
| JMPNC / JMPCN | label | jump to a label if the current result is FALSE |
| RET | | Jump to the end of the current program - no current result needed |
| RETC / RETNC / RETCN | | Jump to the end of the current program if the current result is TRUE / FALSE |
| S | Boolean | sets the operand to TRUE if the current result is TRUE |
| R | Boolean | sets the operand to FALSE if the current result is TRUE |
| CAL | f. block | calls a function block (no current result needed) |
| CALC | f. block | calls a function block if the current result is TRUE |
| CALNC / CALCN | f. block | calls a function block if the current result is FALSE |

**NOTE**

Instructions suffixed by **N** uses the Boolean negation of the operand.

**NOTE**

IL program cannot be called if there is no entry variable, or if its type is complex (e.g. array)

## 2.4 Structured Text (ST)

ST is a structured literal programming language. A ST program is a list of *statements*. Each statement describes an action and must end with a semi-colon ("**;**").

The presentation of the text has no meaning for a ST program. You can insert blank characters and line breaks where you want in the program text.

### 2.4.1 Comments

Comment texts can be entered anywhere in a ST program. Comment texts have no meaning for the execution of the program. A comment text must begin with "**(***" and end with "***)**". Comments can be entered on several lines (i.e. a comment text can include line breaks). Comment texts cannot be nested.

You can also use **//** to add a comment on a single line as shown below:

```
//My main comment
(* My comment *)
a := d + e;
(* A comment can also
be on several lines *)
b := d * e;

c := d - e; (* My comment *)
```

### 2.4.2 Expressions

Each statement describes an action and can include evaluation of complex expressions. An expression is evaluated:

- from the left to the right
- according to the default priority order of operators
- the default priority can be changed using parentheses

Arguments of an expression can be:
- declared variables
- constant expressions
- function calls

### 2.4.3 Statements

Below are available basic statements that can be entered in a ST program:

- assignment
- function block calling

Below are the available conditional statements in ST language:

- IF / THEN / ELSE
  Simple binary switch.
  One or several ELSIF are allowed.

```
IF a = b THEN
   c := 0;
ELSIF a < b THEN
   c := 1;
ELSE
   c := -1;
END_IF;
```

- CASE
  Switch between enumerated statements according to an expression.
  The selector can be any integer or a STRING.

```
CASE iChoice OF
0:
MyString := 'Nothing';
1 .. 2,5:
MyString := 'First case';
3,4:
MyString := 'Second case';
ELSE
MyString := 'Other case';
```

```
END_CASE;
```

Below are the available statements for describing loops in ST language:

**IMPORTANT**

Loop instructions can lead to infinite loops that block the target cycle.
Never test the state of an input in the condition as the input will not be refreshed before the next cycle.

- WHILE
  Repeat a list of statements.
  Condition is evaluated on loop **entry** before the statements.

```
iCount := 0;
WHILE iCount < 100 DO
iCount := iCount +1;
MyVar := MyVar + 1;
END_WHILE;
```

- REPEAT
  Repeat a list of statements.
  Condition is evaluated on loop **exit** after the statements.

```
iCount := 0;
REPEAT
MyVar := MyVar + 1;
iCount := iCount + 1;
UNTIL iCount < 100 END_REPEAT;
```

- FOR
  Iteration of statement execution.
  The **BY** statement is optional (default value is 1)

```
FOR iCount := 0 TO 100 BY 2 DO
MyVar := MyVar + 1;
END_FOR;
```

**NOTE**

Loops with FOR instructions are slow, so you can optimize your code by replacing such iterations with a WHILE statement.

Below are some other statements in ST language:

- WAIT / WAIT_TIME (suspend the execution)

- ON ... DO (conditional execution of statements: provides a simpler syntax for checking the rising edge of a Boolean condition)

**TIP**

ST also provides an automatic completion of typed words. See .

## 2.5 Free Form Ladder Diagram (FFLD)

A Ladder Diagram is a list of *rungs*. Each rung represents a Boolean data flow from a power rail on the left. The power rail represents the TRUE state. The data flow must be understood from the left to the right. Each symbol connected to the rung either changes the rung state or performs an operation. Below are possible graphic items to be entered in FFLD diagrams:

Power Rails
Contacts and Coils
Operations, Functions and Function blocks, represented by rectangular blocks
Labels and Jumps
Use of ST instructions in graphic languages

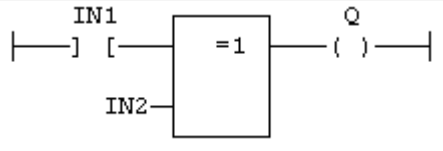
### 2.5.1 Use of the "EN" input and the "ENO" output for blocks

The rung state in a FFLD diagram is always Boolean. Blocks are connected to the rung with their first input and output. This implies that special "EN" and "ENO" input and output are added to the block if its first input or output is not Boolean.

The "EN" input is a condition. It means that the operation represented by the block is not performed if the rung state (EN) is FALSE. The "ENO" output always represents the sane status as the "EN" input: the rung state is not modified by a block having an ENO output.

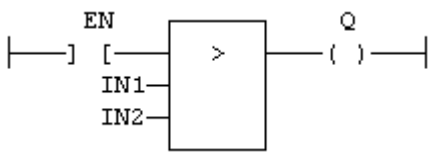
- Below is the example of the "XOR" block, having Boolean inputs and outputs, and requiring no EN or ENO pin:

```
(* First input is the rung. The rung is the output *)
```



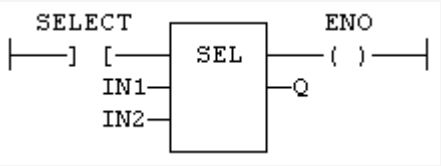

- Below is the example of the ">" (greater than) block, having non Boolean inputs and a Boolean output. This block has an "EN" input in FFLD language:

```
(* The comparison is executed only if EN is TRUE *)
```



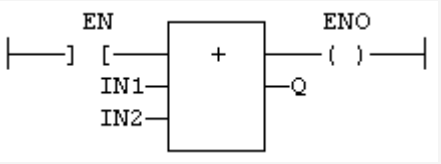

- Below is the example of the "SEL" function, having a first Boolean input, but an integer output. This block has an "ENO" output in FFLD language:

```
(* the input rung is the selector *)
(* ENO has the same value as SELECT *)
```




- Finally, below is the example of an addition, having only numerical arguments. This block has both "EN" and "ENO" pins in FFLD language:

```
(* The addition is executed only if EN is TRUE *)
(* ENO is equal to EN *)
```




### 2.5.2 Contacts and coils

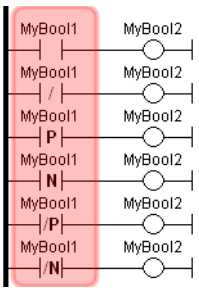
The table below contains a list of the contact and coil types available:

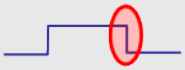
| Contacts | Coils |
|---|---|
| Normally Open -\| \|- | Energize -( )- |
| Normally Closed -\|/\|- | De-energize -(/)- |
| Positive Transition -\|P\|- | Set (Latch) -(S)- |
| Negative Transition -\|N\|- | Reset (Unlatch) -(R)- |
| Normally closed positive transition -\|/P\|- | Positive transition sensing coil -(P)- |
| Normally closed negative transition -\|/N\|- | Negative transition sensing coil -(N)- |

#### 2.5.2.1 FFLD Contacts

Contacts are basic graphic elements of the FFLD language. A contact is associated with a Boolean variable which is displayed above the graphic symbol. A contact sets the state of the rung on its right-hand side, according to the value of the associated variable and the rung state on its left-hand side.

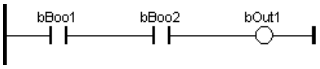
Below are the six possible contact symbols and how they change the flow:



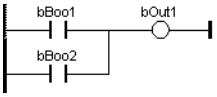
| Contacts | Description |
|---|---|
| boolVariable<br>-] [- | **Normal**: The flow on the right is the Boolean AND operation between: (1) the flow on the left and (2) the associated variable. |
| boolVariable<br>-]/[- | **Negated**: The flow on the right is the Boolean AND operation between: (1) the flow on the left and (2) the negation of the associated variable. |
| boolVariable<br>-]P[- | **Positive Transition**: The flow on the right is TRUE when the flow on the left is TRUE and the associated variable is TRUE and was FALSE the last time this contact was scanned (rising edge).<br>  |
| boolVariable<br>-]N[- | **Negative Transition**: The flow on the right is TRUE when the flow on the left is TRUE and the associated variable is FALSE and was TRUE last time this contact was scanned (falling edge).<br>  |
| boolVariable<br>-]/P[- | **Normally Closed Positive Transition**: The flow on the right is TRUE when the flow on the left is TRUE and the associated variable does not change from FALSE to TRUE from the last scan of this contact to this scan (NOT rising edge). |
| boolVariable<br>-]/N[- | **Normally Closed Negative Transition**: The flow on the right is TRUE when the flow on the left is TRUE and the associated variable does not change from TRUE to FALSE from the last scan of this contact to this scan (NOT falling edge). |

### Serialized and Parallel contacts

Two serial normal contacts represent an AND operation.




Two contacts in parallel represent an OR operation.

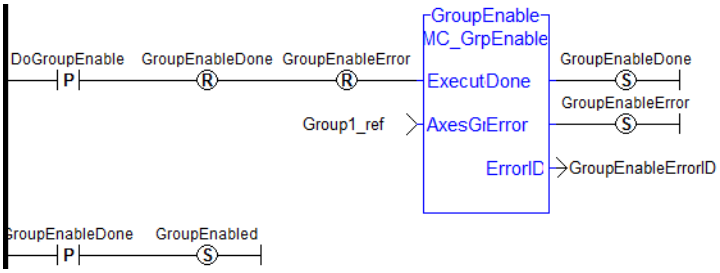



### Transition Contacts

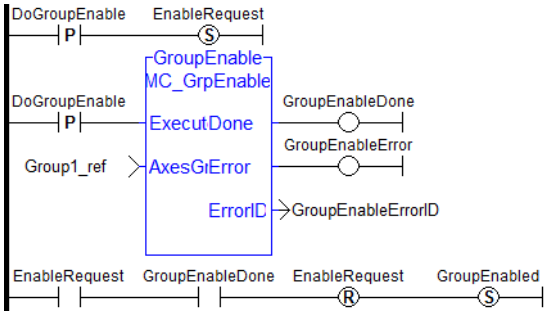
The transition contacts -|P|-, -|N -|/P|-, and -|/N|- compare the current state of the Boolean variable to the Boolean's state the last time the contact was scanned. This means that the Boolean variable could change states several times during a scan, but if it's back to the same state when the transition contact is scanned, the transition contact will not produce a TRUE. Also, some function blocks can complete immediately. Therefore a different approach, other than using transition contacts, is needed to determine if a function block completed successfully.

For example:

MC_GrpEnable executes and turns on its Done output immediately. In the following code, the GroupEnableDone positive transition contact will only provide a TRUE the first time MC_GrpEnable is executed. For all subsequent executions, the positive transition contact will not provide a TRUE since GroupEnableDone will be TRUE every time the contact is scanned.



To remedy this, the following code uses the SET and RESET of a Boolean (i.e. EnableRequest) to provide a way to detect each successful execution of the function block:




**TIP**

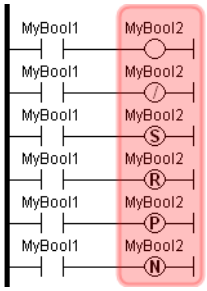
When a contact or coil is selected, you can press the **Spacebar** to change its type (normal, negated...).

When your application is running, you can select a contact and press the **Spacebar** to swap its value between TRUE and FALSE.

### 2.5.2.2 FFLD Coils

Coils are basic graphic elements of the FFLD language. A coil is associated with a Boolean variable which is displayed above the graphic symbol. A coil performs a change of the associated variable according to the flow on its left-hand side.

Below are the six possible coil symbols:




| Coils | Description |
|---|---|
| `boolVariable -( )-` | **Normal**: the associated variable is forced to the value of the flow on the left of the coil. |
| `boolVariable -(/)-` | **Negated**: the associated variable is forced to the negation of the flow on the left of the coil. |
| `boolVariable -(S)-` | **Set**: the associated variable is forced to TRUE if the flow on the left is TRUE. (no action if the flow is FALSE)<br>*Rules for Set coil animation:*<br>• Power Flow on left is TRUE:<br>  • The horizontal wires on either side of the (S) are red<br>  • The variable and the (S) are red<br>• Power Flow on left is FALSE and the (S) variable is Energized (ON)<br>  • The horizontal lines on either sided of (S) are black<br>  • The variable and the (S) are red<br>• In all other cases:<br>  • The horizontal wires are black<br>  • The variable and the (S) are black |

| Coils | Description |
|---|---|
| `boolVariable -(R)-` | **Reset**: the associated variable is forced to FALSE if the flow on the left is TRUE. (no action if the rung state is FALSE)<br>*Rules for Reset coil animation:*<br>• Power Flow on left is TRUE:<br>  • The horizontal lines are red<br>  • The variable above (R) is black<br>  • The R and the circle around the R are black<br>• Power Flow on left is FALSE and variable above reset coil is NOT Energized (OFF)<br>  • The horizontal lines are black<br>  • The variable above (R) is black<br>  • The R and the circle around the R are black<br>• Power Flow on left is FALSE and variable above reset coil is Energized (ON)<br>  • The horizontal lines are black<br>  • The variable above (R) is red<br>  • The R and the circle around the R are red |
| `boolVariable -(P)-` | **Positive transition**: the associated variable is forced to TRUE if the flow on the left changes from **FALSE to TRUE**(and forced to FALSE in all other cases) |
| `boolVariable -(N)-` | **Negative transition**: the associated variable is forced to TRUE if the flow on the left changes from **TRUE to FALSE**(and forced to FALSE in all other cases) |

**TIP**

When a contact or coil is selected, you can press the **Spacebar** to change its type (normal, negated...).

When your application is running, you can select a contact and press the **Spacebar** to swap its value between TRUE and FALSE.

**IMPORTANT**

Although coils are commonly put at the end, the rung can be continued after a coil. The flow is **never changed** by a coil symbol.

# 3 PLC Standard Libraries

The following topics detail the set of programming features and standard blocks:



Note: Some other functions not documented here are reserved for diagnostics and special operations. Please contact your technical support for further information.

## 3.1 Basic Operations

Below are the language features for basic data manipulation:

- Variable assignment
- Bit access
- Parenthesis
- Calling a function
- Calling a function block
- Calling a sub-program
- MOVEBLOCK: Copying/moving array items
- COUNTOF: Number of items in an array
- INC: Increase a variable
- DEC: decrease a variable
- NEG: integer negation (unary operator)

Below are the language features for controlling the execution of a program:

- Labels
- Jumps
- RETURN

Below are the structured statements for controlling the execution of a program:

| | |
|---|---|
| IF | Conditional execution of statements. |
| WHILE | Repeat statements while a condition is TRUE. |
| REPEAT | Repeat statements until a condition is TRUE. |
| FOR | Execute iterations of statements. |
| CASE | Switch to one of various possible statements. |
| EXIT | Exit from a loop instruction. |
| WAIT | Delay program execution. |
| ON | Conditional execution. |

### 3.1.1 := PLCopen

*Operator* - variable assignment.

#### 3.1.1.1 Inputs

`IN : ANY` Any variable or complex expression

#### 3.1.1.2 Outputs

`Q : ANY` Forced variable

#### 3.1.1.3 Remarks

The output variable and the input expression must have the same type. The forced variable cannot have the "read only" attribute. In FFLD and FBD languages, the "1" block is available to perform a "1 gain" data copy (1 copy). In FFLD language, the input rung (EN) enables the assignment, and the output rung keeps the state of the input rung. In IL language, the FFLD instruction loads the first operand, and the ST instruction stores the current result into a variable. The current result and the operand of ST must have the same type. Both FFLD and ST instructions can be modified by "N" in case of a Boolean operand for performing a Boolean negation.

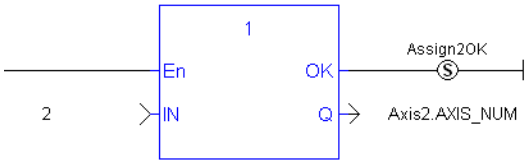
#### 3.1.1.4 ST Language

```
Q := IN; (* copy IN into variable Q *)
 Q := (IN1 + (IN2 / IN 3)) * IN4; (* assign the result of a complex expression *)
 result := SIN (angle); (* assign a variable with the result of a function *)
 time := MyTon.ET; (* assign a variable with an output parameter of a function block *)
```

#### 3.1.1.5 FBD Language


#### 3.1.1.6 FFLD Language

```
(* The copy is executed only if EN is TRUE *)
```




#### 3.1.1.7 IL Language:

```
Op1: FFLD  IN   (* current result is: IN *)
 ST  Q    (* Q is: IN *)
 FFLDN IN1  (* current result is: NOT (IN1) *)
 ST  Q    (* Q is: NOT (IN1) *)
 FFLD  IN2  (* current result is: IN2 *)
 STN Q    (* Q is: NOT (IN2) *)
```

**See also:**

Parenthesis

### 3.1.2 Access to Bits of an Integer

You can directly specify a bit within an integer variable in expressions and diagrams, using the following notation:

*Variable.BitNo*

Where:

*Variable*: is the name of an integer variable
*BitNo*: is the number of the bit in the integer.

The variable can have one of the following data types:

- SINT, USINT, BYTE (8 bits from .0 to .7)
- INT, UINT, WORD (16 bits from .0 to .15)
- DINT, UDINT, DWORD (32 bits from .0 to 31)
- LINT, ULINT, LWORD (64 bits from .0 to .63)

0 always represents the less significant bit.

### 3.1.3 Differences Between Functions and Function Blocks

It is important to clearly understand what is different between functions and function blocks.

- A Function[1] (  ) is called once and it performs an action. This is synchronous.
- A Function Block[2] (  ) or "FB" is an instance that has its own set of data. A FB very likely maintains its own, internal machine state and very often has an output to indicate when the work is done. A FB is most likely to be asynchronous.

The best way to work with a function block is to call it during multiple scan. This triggers the action the first time, then you may monitor the status of this action, especially via the "done" output.

### 3.1.4 Calling a Sub-Program

A sub-program is called by another program. Unlike function blocks, local variables of a sub-program are not instantiated, and thus you do not need to declare instances. A call to a sub-program processes the block algorithm using the specified input parameters. Output parameters can then be accessed.

#### 3.1.4.1 ST Language

To call a sub-program in ST, you have to specify its name, followed by the input parameters written between parentheses and separated by comas. To have access to an output parameter, use the name of the sub-program followed by a dot '.' and the name of the wished parameter:

```
MySubProg (i1, i2); (* calls the sub-program *)
Res1 := MySubProg.Q1;
Res2 := MySubProg.Q2;
```

Alternatively, if a sub-program has one and only one output parameter, it can be called as a function in ST language:

```
Res := MySubProg (i1, i2);
```

#### 3.1.4.2 FBD and FFLD Languages

To call a sub-program in FBD or FFLD languages, you just need to insert the block in the diagram and to connect its inputs and outputs.

#### 3.1.4.3 IL Language

To call a sub-program in IL language, you must use the CAL instruction with the name of the sub-program, followed by the input parameters written between parentheses and separated by comas. Alternatively the CALC, CALCN or CALNC conditional instructions can be used:

```
CAL    Calls the sub-program
CALC   Calls the sub-program if the current result is TRUE
CALNC  Calls the sub-program if the current result is FALSE
CALCN  same as CALNC
```

Here is an example:

```
Op1: CAL  MySubProg (i1, i2)
FFLD  MySubProg.Q1
ST  Res1
```

1A function calculates a result according to the current value of its inputs. A function has no internal data and is not linked to declared instances.
2A function block groups an algorithm and a set of private data. It has inputs and outputs.

```
FFLD  MySubProg.Q2
ST  Res2
```

### 3.1.5 CASE OF ELSE END_CASE

*Statement* - switch between enumerated statements.

#### 3.1.5.1 Syntax

```
CASE <DINT expression> OF
<value> :
    <statements>
<value> , <value> :
    <statements>;
<value> .. <value> :
    <statements>;
ELSE
    <statements>
END_CASE;
```

#### 3.1.5.2 Remarks

All enumerated values correspond to the evaluation of the DINT expression and are possible cases in the execution of the statements. The statements specified after the ELSE keyword are executed if the expression takes a value which is not enumerated in the switch. For each case, you must specify either a value, or a list of possible values separated by comas (",") or a range of values specified by a "min .. max" interval. You must enter space characters before and after the ".." separator.

#### 3.1.5.3 ST Language

```
(* this example check first prime numbers *)
CASE iNumber OF
0 :
  Alarm := TRUE;
  AlarmText := '0 gives no result';
1 .. 3, 5 :
  bPrime := TRUE;
4, 6 :
  bPrime := FALSE;
ELSE
  Alarm := TRUE;
  AlarmText := 'I don't know after 6 !';
END_CASE;
```

#### 3.1.5.4 FBD Language

*Not available*

#### 3.1.5.5 FFLD Language

*Not available*

#### 3.1.5.6 IL Language

*Not available*

**See also**

IF WHILE REPEAT FOR EXIT

## 3.1.6 COUNTOF PLCopen

*Function* - Returns the number of items in an array

### 3.1.6.1 Inputs

`ARR : ANY` Declared array

### 3.1.6.2 Outputs

`Q : DINT` Total number of items in the array

### 3.1.6.3 Remarks

The input must be an array and can have any data type. This function is particularly useful to avoid writing directly the actual size of an array in a program, and thus keep the program independent from the declaration. Example:

```
FOR i := 1 TO CountOf (MyArray) DO
   MyArray[i-1] := 0;
END_FOR;
```

In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

**Examples**

| array | return |
|---|---|
| Arr1 [ 0..9 ] | 10 |
| Arr2 [ 0..4 , 0..9 ] | 50 |

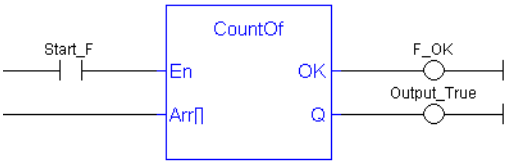
### 3.1.6.4 ST Language

Q := CountOf (ARR);

### 3.1.6.5 FBD Language


### 3.1.6.6 FFLD Language

(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)




### 3.1.6.7 IL Language

*Not available*

## 3.1.7 DEC

PLCopen

*Function* - Decrease a numerical variable

### 3.1.7.1 Inputs

`IN : ANY` Numerical variable (increased after call).

### 3.1.7.2 Outputs

`Q : ANY` Decreased value

### 3.1.7.3 Remarks

When the function is called, the variable connected to the "IN" input is decreased and copied to Q. All data types are supported except BOOL and STRING: for these types, the output is the copy of IN.

For real values, variable is decreased by "1.0". For time values, variable is decreased by 1 ms.

The IN input must be directly connected to a variable, and cannot be a constant or complex expression.

This function is particularly designed for ST language. It allows simplified writing as assigning the result of the function is not mandatory.

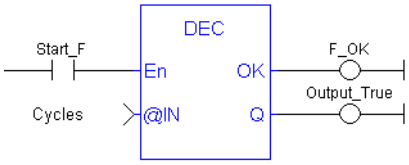
### 3.1.7.4 ST Language

```
IN := 2;
Q := DEC (IN);
(* now: IN = 1 ; Q = 1 *)
```

```
DEC (IN); (* simplified call *)
```

### 3.1.7.5 FBD Language


### 3.1.7.6 FFLD Language




### 3.1.7.7 IL Language

*not available*

## 3.1.8 EXIT

*Statement* - Exit from a loop statement

### 3.1.8.1 Remarks

The EXIT statement indicates that the current loop (WHILE, REPEAT or FOR) must be finished. The execution continues after the END_WHILE, END_REPEAT or END_FOR keyword or the loop where the EXIT is. EXIT quits only one loop and cannot be used to exit at the same time several levels of nested loops.

**IMPORTANT**

loop instructions can lead to infinite loops that block the target cycle.

#### 3.1.8.2 ST Language

```
(* this program searches for the first non null item of an array *)
iFound = -1; (* means: not found *)
FOR iPos := 0 TO (iArrayDim - 1) DO
   IF iPos <> 0 THEN
      iFound := iPos;
      EXIT;
   END_IF;
END_FOR;
```

#### 3.1.8.3 FBD Language

*Not available*

#### 3.1.8.4 FFLD Language

*Not available*

#### 3.1.8.5 IL Language

*Not available*

**See also**

IF WHILE REPEAT FOR CASE

### 3.1.9 FOR TO BY END_FOR

*Statement* - Iteration of statement execution.

#### 3.1.9.1 Syntax

FOR <*index*> := <*minimum*> TO <*maximum*> BY <*step*> DO
  <statements>
END_FOR;

*index* = DINT internal variable used as index
*minimum* = DINT expression: initial value for *index*
*maximum* = DINT expression: maximum allowed value for *index*
*step* = DINT expression: increasing step of *index* after each iteration (default is 1)

#### 3.1.9.2 Remarks

The "BY <*step*>" statement can be omitted. The default value for the step is 1.

#### 3.1.9.3 ST Language

```
iArrayDim := 10;
```

```
(* resets all items of the array to 0 *)
FOR iPos := 0 TO (iArrayDim - 1) DO
   MyArray[iPos] := 0;
END_FOR;

(* set all items with odd index to 1 *)
FOR iPos := 1 TO 9 BY 2 DO
   MyArray[ipos] := 1;
END_FOR;
```

#### 3.1.9.4 FBD Language

*Not available*

#### 3.1.9.5 FFLD Language

*Not available*

#### 3.1.9.6 IL Language

*Not available*

**See also**

IF WHILE REPEAT CASE EXIT

### 3.1.10 IF THEN ELSE ELSIF END_IF

*Statement* - Conditional execution of statements.

#### 3.1.10.1 Syntax

IF <BOOL expression> THEN
<statements>
ELSIF <BOOL expression> THEN
<statements>
ELSE
<statements>
END_IF;

#### 3.1.10.2 Remarks

The IF statement is available in ST only. The execution of the statements is conditioned by a Boolean expression. ELSIF and ELSE statements are optional. There can be several ELSIF statements.

#### 3.1.10.3 ST Language

```
(* simple condition *)

IF bCond THEN
   Q1 := IN1;
   Q2 := TRUE;
END_IF;

(* binary selection *)
IF bCond THEN
   Q1 := IN1;
```

```
   Q2 := TRUE;
   ELSE
      Q1 := IN2;
      Q2 := FALSE;
END_IF;

(* enumerated conditions *)
IF bCond1 THEN
   Q1 := IN1;
   ELSIF bCond2 THEN
      Q1 := IN2;
      ELSIF bCond3 THEN
         Q1 := IN3;
         ELSE
            Q1 := IN4;
END_IF;
```

#### 3.1.10.4 FBD Language

*Not available*

#### 3.1.10.5 FFLD Language

*Not available*

#### 3.1.10.6 IL Language

*Not available*

**See also**

WHILE REPEAT FOR CASE EXIT

### 3.1.11 INC PLCopen

*Function* - Increase a numerical variable

#### 3.1.11.1 Inputs

`IN : ANY` Numerical variable (increased after call).

#### 3.1.11.2 Outputs

`Q : ANY` Increased value

#### 3.1.11.3 Remarks

When the function is called, the variable connected to the "IN" input is increased and copied to Q. All data types are supported except BOOL and STRING: for these types, the output is the copy of IN.

For real values, variable is increased by "1.0". For time values, variable is increased by 1 ms.

The IN input must be directly connected to a variable, and cannot be a constant or complex expression.

This function is particularly designed for ST language. It allows simplified writing as assigning the result of the function is not mandatory.

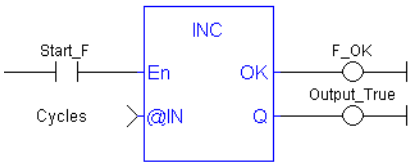
#### 3.1.11.4 ST Language

```
IN := 1;
Q := INC (IN);
(* now: IN = 2 ; Q = 2 *)
```

```
INC (IN); (* simplified call *)
```

#### 3.1.11.5 FBD Language


#### 3.1.11.6 FFLD Language




#### 3.1.11.7 IL Language

*not available*

### 3.1.12 MOVEBLOCK PLCopen

*Function* - Move/Copy items of an array.

#### 3.1.12.1 Inputs

```
SRC: ANY (*)   Array containing the source of the copy
DST : ANY (*)    Array containing the destination of the copy
PosSRC: DINT   Index of the first character in SRC
PosDST : DINT  Index of the destination in DST
NB : DINT      Number of items to be copied
```

(*) `SRC and DST` cannot be a STRING

#### 3.1.12.2 Outputs

```
OK : BOOL        TRUE if successful
```

#### 3.1.12.3 Remarks

Arrays of string are not supported by this function.

In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The function is not available in IL language.

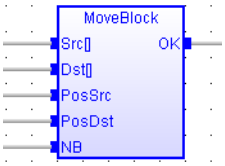
The function copies a number (NB) of consecutive items starting at the PosSRC index in SRC array to PosDST position in DST array. SRC and DST can be the same array. In that case, the function avoids lost items when source and destination areas overlap.

This function checks array bounds and is always safe. The function returns TRUE if successful. It returns FALSE if input positions and number do not fit the bounds of SRC and DST arrays.

#### 3.1.12.4 ST Language

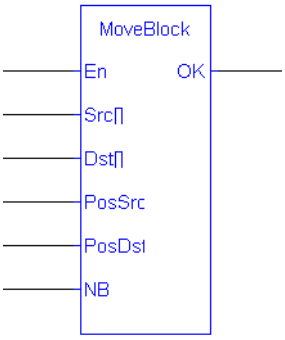
```
OK := MOVEBLOCK (SRC, DST, PosSRS, PosDST, NB);
```

#### 3.1.12.5 FBD Language




#### 3.1.12.6 FFLD Language

(* The function is executed only if EN is TRUE *)




#### 3.1.12.7 IL Language

*Not available*

### 3.1.13 NEG - PLCopen

*Operator* - Performs an integer negation of the input.

#### 3.1.13.1 Inputs

```
IN : DINT  Integer value
```

#### 3.1.13.2 Outputs

```
Q : DINT   Integer negation of the input
```

#### 3.1.13.3 Truth table (examples)

| IN | Q |
|---|---|
| 0 | 0 |
| 1 | -1 |
| -123 | 123 |

#### 3.1.13.4 Remarks

- In FBD and FFLD language, the block "NEG" can be used.
- In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.
- This feature is not available in IL language. In ST language, "-" can be followed by a complex Boolean expression between parentheses.

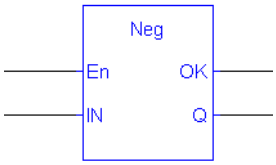
#### 3.1.13.5 ST Language

```
Q := -IN;
Q := - (IN1 + IN2);
```

#### 3.1.13.6 FBD Language


#### 3.1.13.7 FFLD Language

(* The negation is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)




### 3.1.14 ON

*Statement* - Conditional execution of statements.

The ON instruction provides a simpler syntax for checking the rising edge of a Boolean condition.

#### 3.1.14.1 Syntax

```
ON <BOOL expression> DO
   <statements>
END_DO;
```

#### 3.1.14.2 Remarks

Statements within the ON structure are executed only when the Boolean expression rises from FALSE to TRUE. The ON instruction avoids systematic use of the R_TRIG function block or other "last state" flags.

The ON syntax is available in any program, sub-program or UDFB.

This statement is an extension to the standard and is not IEC61131-3 compliant.

**IMPORTANT**

This instruction **should not be used inside UDFBs**. This instruction is not UDFB safe.

#### 3.1.14.3 ST Language

```
(* This example counts the rising edges of variable bIN *)
ON bIN DO
```

```
 diCount := diCount + 1;
D_DO;
```

### 3.1.15 ( )

*Operator* - force the evaluation order in a complex expression.

#### 3.1.15.1 Remarks

Parentheses are used in ST and IL language for changing the default evaluation order of various operations within a complex expression. For instance, the default evaluation of "2 * 3 + 4" expression in ST language gives a result of 10 as "*" operator has highest priority. Changing the expression as "2 * ( 3 + 4 )" gives a result of 14. Parentheses can be nested in a complex expression.

Below is the default evaluation order for ST language operations (1rst is highest priority):

| | |
|---|---|
| Unary operators | - NOT |
| Multiply/Divide | * / |
| Add/Subtract | + - |
| Comparisons | < > <= >= = <> |
| Boolean And | & AND |
| Boolean Or | OR |
| Exclusive OR | XOR |

In IL language, the default order is the sequence of instructions. Each new instruction modifies the current result sequentially. In IL language, the opening parenthesis "(" is written between the instruction and its operand. The closing parenthesis ")" must be written alone as an instruction without operand.

#### 3.1.15.2 ST Language

```
Q := (IN1 + (IN2 / IN 3)) * IN4;
```

#### 3.1.15.3 FBD Language

*Not available*

#### 3.1.15.4 FFLD Language

*Not available*

#### 3.1.15.5 IL Language

```
Op1: FFLD(  IN1
     ADD( IN2
     MUL  IN3
     )
     SUB  IN4
     )
     ST   Q     (* Q is: (IN1 + (IN2 * IN3) -  IN4) *)
```

**See also**

Assignment

## 3.1.16 REPEAT UNTIL END_REPEAT

*Statement* - Repeat a list of statements.

### 3.1.16.1 Syntax

```
REPEAT
   <statements>
UNTIL <BOOL expression> END_REPEAT;
```

### 3.1.16.2 Remarks

The statements between "REPEAT" and "UNTIL" are executed until the Boolean expression is TRUE. The condition is evaluated ***after*** the statements are executed. Statements are executed at least once.

**IMPORTANT**

Loop instructions can lead to infinite loops that block the target cycle. Never test the state of an input in the condition as the input will not be refreshed before the next cycle.

### 3.1.16.3 ST Language

```
iPos := 0;
REPEAT
   MyArray[iPos] := 0;
   iNbCleared := iNbCleared + 1;
   iPos := iPos + 1;
UNTIL iPos = iMax END_REPEAT;
```

### 3.1.16.4 FBD Language

*Not available*

### 3.1.16.5 FFLD Language

*Not available*

### 3.1.16.6 IL Language

*Not available*

**See also**

IF WHILE FOR CASE EXIT

## 3.1.17 RETURN RET RETC RETNC RETCN

*Statement* - Jump to the end of the program.

### 3.1.17.1 Remarks

The "RETURN" statement jumps to the end of the program. In FBD language, the return statement is represented by the "<RETURN>" symbol. The input of the symbol must be connected to a valid Boolean signal. The jump is performed only if the input is TRUE. In FFLD language, the "<RETURN>" symbol is used as a coil at the end of a rung. The jump is performed only if the rung state is TRUE. In IL language, RET, RETC, RETCN and RETNC instructions are used.

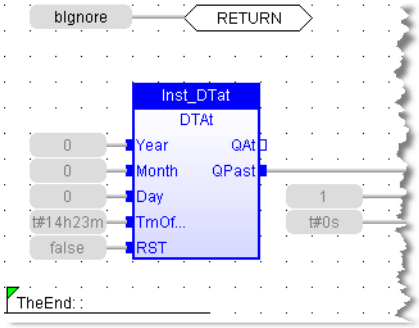
When used within an action block of an SFC step, the RETURN statement jumps to the end of the action block.

### 3.1.17.2 ST Language

```
IF NOT bEnable THEN
   RETURN;
END_IF;
(* the rest of the program will not be executed if bEnable is FALSE *)
```

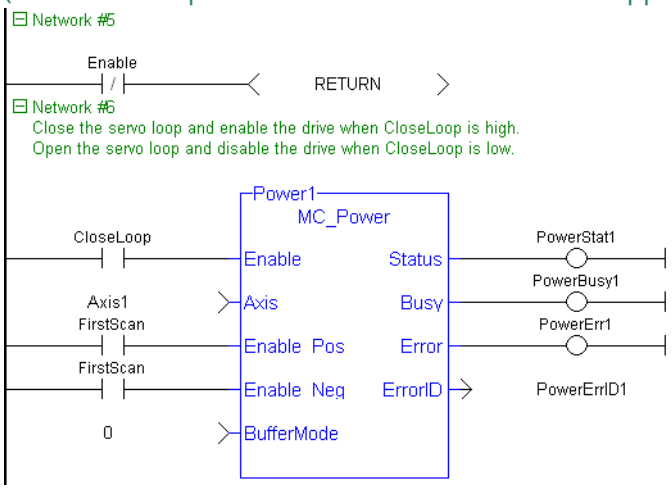
### 3.1.17.3 FBD Language

(* In this example the DTat block will not be called if bIgnore is TRUE *)




### 3.1.17.4 FFLD Language

(* In this example all the networks above 5 are skipped if ENABLE is FALSE *)




### 3.1.17.5 IL Language

Below is the meaning of possible instructions:

`RET` Jump to the end always
`RETC` Jump to the end if the current result is TRUE
`RETNC` Jump to the end if the current result is FALSE
`RETCN` Same as `RETNC`

```
Start: FFLD    IN1
       RETC           (* Jump to the end if IN1 is TRUE *)

       FFLD    IN2    (* these instructions are not executed *)
       ST      Q2      (* if IN1 is TRUE *)
       RET            (* Jump to the end unconditionally *)
```

```
    FFLD    IN3    (* these instructions are never executed *)
    ST      Q3
```

See also

Labels Jumps

### 3.1.18 WAIT / WAIT_TIME

*Statement* - Suspend the execution of a ST program.

The WAIT instruction provides an easy way to program a state machine. This avoids the use of complex CASE structures.

#### 3.1.18.1 Syntax

```
WAIT <BOOL expression> ;
WAIT_TIME <TIME expression> ;
```

#### 3.1.18.2 Remarks

The WAIT statement checks the attached Boolean expression and does the following:

- If the expression is TRUE, the program continues normally.
- If the expression is FALSE, then the execution of the program is suspended up to the **next PLC cycle**. The Boolean expression will be checked again during next cycles until it becomes TRUE. The execution of other programs is not affected.

The WAIT_TIME statement suspends the execution of the program for the specified duration. The execution of other programs is not affected.

These instructions are available in ST language only and has no correspondence in other languages. These instructions cannot be called in a User-Defined Function Block (UDFB). The use of WAIT or WAIT_TIME in a UDFB provokes a compile error.

WAIT and WAIT_TIME instructions can be called in a sub-program. However, it can lead to some unsafe situation if the same sub program is called from various programs. Re-entrancy is not supported by WAIT and WAIT_TIME instructions. Avoiding this situation is the responsibility of the programmer. The compiler outputs some warning messages if a sub-program containing a WAIT or WAIT_TIME instruction is called from more than one program.

These instructions must not be called from ST parts of SFC programs. This makes no sense as SFC is already a state machine. The use of WAIT or WAIT_TME in SFC or in a sub-program called from SFC provokes a compile error.

These instructions are not available when the code is compiled through a "C" compiler. Using "C" code generation with a program containing a WAIT or WAIT_TIME instruction provokes an error during post-compiling.

These statement are extensions to the standard and are not IEC61131-3 compliant.

**IMPORTANT**

This instruction **should not be used inside UDFBs**. This instruction is not UDFB safe.

#### 3.1.18.3 ST Language

```
(* use of WAIT with different kinds of BOOL expressions *)
WAIT BoolVariable;
WAIT (diLevel > 100) AND NOT bAlarm;
```

```
WAIT SubProgCall ();
(* use of WAIT_TIME with different kinds of TIME expressions *)
WAIT_TIME t#2s;
WAIT_TIME TimeVariable;
```

### 3.1.19 WHILE DO END_WHILE

*Statement* - Repeat a list of statements.

#### 3.1.19.1 Syntax

```
WHILE <BOOL expression> DO
  <statements>
END_WHILE ;
```

#### 3.1.19.2 Remarks

The statements between "DO" and "END_WHILE" are executed while the Boolean expression is TRUE. The condition is evaluated ***before*** the statements are executed. If the condition is FALSE when WHILE is first reached, statements are never executed.

**IMPORTANT**

Loop instructions can lead to infinite loops that block the target cycle. Never test the state of an input in the condition as the input will not be refreshed before the next cycle.

#### 3.1.19.3 ST Language

```
iPos := 0;
WHILE iPos < iMax DO
   MyArray[iPos] := 0;
   iNbCleared := iNbCleared + 1;
END_WHILE;
```

#### 3.1.19.4 FBD Language

*Not available*

#### 3.1.19.5 FFLD Language

*Not available*

#### 3.1.19.6 IL Language

*Not available*

**See also**

IF REPEAT FOR CASE EXIT

## 3.2 Boolean operations PLCopen

Below are the standard operators for managing Booleans:

| | |
|---|---|
| AND | performs a Boolean AND |
| OR | performs a Boolean OR |
| XOR | performs an exclusive OR |
| NOT | performs a Boolean negation of its input |
| QOR | qualified OR |
| S | force a Boolean output to TRUE |
| R | force a Boolean output to FALSE |

Below are the available blocks for managing Boolean signals:

| | |
|---|---|
| RS | reset dominant bistable |
| SR | set dominant bistable |
| R_TRIG | rising pulse detection |
| F_TRIG | falling pulse detection |
| SEMA | semaphore |
| FLIPFLOP | flipflop^bistable |

### 3.2.1 FLIPFLOP PLCopen

*Function Block* - Flipflop bistable.

#### 3.2.1.1 Inputs

```
IN  : BOOL Swap command (on rising edge)
RST : BOOL Reset to FALSE
```

#### 3.2.1.2 Outputs

```
Q   : BOOL Output
```

#### 3.2.1.3 Remarks

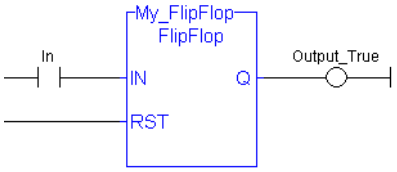
The output is systematically reset to FALSE if RST is TRUE.
The output changes on each rising edge of the IN input, if RST is FALSE.

#### 3.2.1.4 ST Language

```
(* MyFlipFlop is declared as an instance of FLIPFLOP function block *)
MyFlipFlop (IN, RST);
Q := MyFlipFlop.Q;
```

#### 3.2.1.5 FBD Language


#### 3.2.1.6 FFLD Language




#### 3.2.1.7 IL Language

```
(* MyFlipFlop is declared as an instance of FLIPFLOP function block *)
Op1: CAL MyFlipFlop (IN, RST)
    FFLD  MyFlipFlop.Q
    ST  Q1
```

**See also**

R S SR

### 3.2.2 F_TRIG

PLCopen

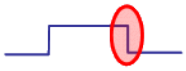
*Function Block* - Falling pulse detection

#### 3.2.2.1 Inputs

`CLK : BOOL` Boolean signal

#### 3.2.2.2 Outputs

`Q : BOOL` TRUE when the input changes from TRUE to FALSE



#### 3.2.2.3 Truth table

| CLK | CLK *prev* | Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

#### 3.2.2.4 Remarks

Although ]P[ and ]N[ contacts can be used in FFLD language, it is recommended to use declared instances of R_TRIG or F_TRIG function blocks in order to avoid contingencies during an Online Change.

#### 3.2.2.5 ST Language

```
(* MyTrigger is declared as an instance of F_TRIG function block *)
MyTrigger (CLK);
Q := MyTrigger.Q;
```

#### 3.2.2.6 FBD Language


#### 3.2.2.7 FFLD Language


#### 3.2.2.8 IL Language:

```
(* MyTrigger is declared as an instance of F_TRIG function block *)
Op1: CAL MyTrigger (CLK)
LD  MyTrigger.Q
ST  Q
```

**See also**

R_TRIG

### 3.2.3 NOT PLCopen

*Operator* - Performs a Boolean negation of the input.

#### 3.2.3.1 Inputs

`IN : BOOL` Boolean value

#### 3.2.3.2 Outputs

`Q : BOOL` Boolean negation of the input

#### 3.2.3.3 Truth table

| IN | Q |
|---|---|
| 0 | 1 |
| 1 | 0 |

#### 3.2.3.4 Remarks

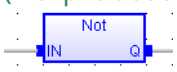
In FBD language, the block "NOT" can be used. Alternatively, you can use a link terminated by a "o" negation. In FFLD language, negated contacts and coils can be used. In IL language, the "N" modifier can be used with instructions FFLD, AND, OR, XOR and ST. It represents a negation of the operand. In ST language, NOT can be followed by a complex Boolean expression between parentheses.

#### 3.2.3.5 ST Language

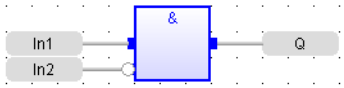
```
Q := NOT IN;
Q := NOT (IN1 OR IN2);
```

#### 3.2.3.6 FBD Language

```
(* explicit use of the "NOT" block *)
```

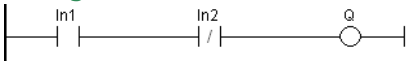



```
(* use of a negated link: Q is IN1 AND NOT IN2 *)
```

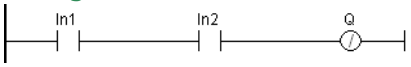



#### 3.2.3.7 FFLD Language

(* Negated contact: Q is: IN1 AND NOT IN2 *)




(* Negated coil: Q is NOT (IN1 AND IN2) *)




#### 3.2.3.8 IL Language:

```
Op1: FFLDN IN1
     OR  IN2
     ST  Q    (* Q is equal to: (NOT IN1) OR IN2 *)
Op2: FFLD  IN1
     AND IN2
     STN Q    (* Q is equal to: NOT (IN1 AND IN2) *)
```

**See also**

AND OR XOR

### 3.2.4 QOR PLCopen

*Operator* - Count the number of TRUE inputs.

#### 3.2.4.1 Inputs

`IN1 ... INn : BOOL` Boolean inputs

#### 3.2.4.2 Outputs

`Q : DINT` Number of inputs being TRUE

#### 3.2.4.3 Remarks

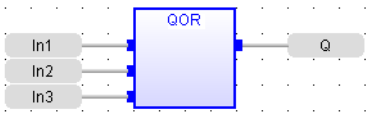
The block accepts a non-fixed number of inputs.

#### 3.2.4.4 ST Language

```
Q := QOR (IN1, IN2);
Q := QOR (IN1, IN2, IN3, IN4, IN5, IN6);
```

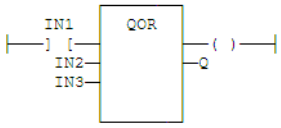
#### 3.2.4.5 FBD Language

(* the block may have up to 16 inputs *)




#### 3.2.4.6 FFLD Language

(* the block may have up to 16 inputs *)



#### 3.2.4.7 IL Language

```
Op1: LD  IN1
     QOR  IN2, IN3
     ST  Q
```

### 3.2.5 R

*Operator* - Force a Boolean output to FALSE.

#### 3.2.5.1 Inputs

`RESET : BOOL` Condition

#### 3.2.5.2 Outputs

`Q : BOOL` Output to be forced

#### 3.2.5.3 Truth table

| RESET | Q *prev* | Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

#### 3.2.5.4 Remarks

S and R operators are available as standard instructions in the IL language. In FFLD languages they are represented by (S) and (R) coils. In FBD language, you can use (S) and (R) coils, but you must prefer RS and SR function blocks. Set and reset operations are not available in ST language.

#### 3.2.5.5 ST Language

*Not available.*

#### 3.2.5.6 FBD Language

*Not available. Use RS or SR function blocks.*

#### 3.2.5.7 FFLD Language

(* use of "R" coil *)


#### 3.2.5.8 IL Language:

```
Op1: FFLD  RESET
   R  Q   (* Q is forced to FALSE if RESET is TRUE *)
          (* Q is unchanged if RESET is FALSE *)
```

**See also**

S RS SR

### 3.2.6 RS PLCopen

*Function Block* - Reset dominant bistable.

### 3.2.6.1 Inputs

`SET` : `BOOL` Condition for forcing to TRUE
`RESET1` : `BOOL` Condition for forcing to FALSE (highest priority command)

### 3.2.6.2 Outputs

`Q1` : `BOOL` Output to be forced

### 3.2.6.3 Truth table

| SET | RESET1 | Q1 *prev* | Q1 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

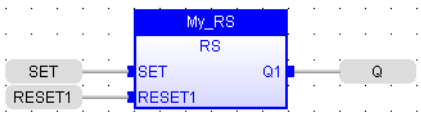
### 3.2.6.4 Remarks

The output is unchanged when both inputs are FALSE. When both inputs are TRUE, the output is forced to FALSE (reset dominant).

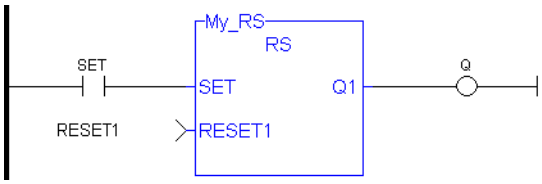
### 3.2.6.5 ST Language

```
(* MyRS is declared as an instance of RS function block *)
MyRS (SET, RESET1);
Q1 := MyRS.Q1;
```

### 3.2.6.6 FBD Language



#### 3.2.6.7 FFLD Language




#### 3.2.6.8 IL Language:

```
(* MyRS is declared as an instance of RS function block *)
Op1: CAL MyRS (SET, RESET1)
     FFLD  MyRS.Q1
     ST  Q1
```

**See also**

R S SR

### 3.2.7 R_TRIG PLCopen

*Function Block* - Rising pulse detection

#### 3.2.7.1 Inputs

`CLK : BOOL` Boolean signal

#### 3.2.7.2 Outputs

`Q : BOOL` TRUE when the input changes from FALSE to TRUE

#### 3.2.7.3 Truth table

| CLK | CLK *prev* | Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

#### 3.2.7.4 Remarks

Although ]P[ and ]N[ contacts can be used in FFLD language, it is recommended to use declared instances of R_TRIG or F_TRIG function blocks in order to avoid contingencies during an Online Change.

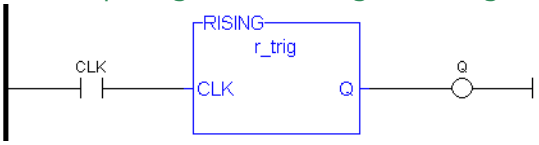
#### 3.2.7.5 ST Language

```
(* MyTrigger is declared as an instance of R_TRIG function block *)
MyTrigger (CLK);
Q := MyTrigger.Q;
```

#### 3.2.7.6 FBD Language

#### 3.2.7.7 FFLD Language

(* the input signal is the rung - the rung is the output *)




#### 3.2.7.8 IL Language:

```
(* MyTrigger is declared as an instance of R_TRIG function block *)
Op1: CAL MyTrigger (CLK)
FFLD  MyTrigger.Q
ST  Q
```

**See also**

F_TRIG

### 3.2.8 S

*Operator* - Force a Boolean output to TRUE.

#### 3.2.8.1 Inputs

`SET : BOOL` Condition

#### 3.2.8.2 Outputs

`Q : BOOL` Output to be forced

#### 3.2.8.3 Truth table

| SET | Q prev | Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

#### 3.2.8.4 Remarks

S and R operators are available as standard instructions in the IL language. In FFLD languages they are represented by (S) and (R) coils In FBD language, you can use (S) and (R) coils, but you must prefer RS and SR function blocks. Set and reset operations are not available in ST language.

#### 3.2.8.5 ST Language

*Not available.*

#### 3.2.8.6 FBD Language

*Not available. Use RS or SR function blocks.*

#### 3.2.8.7 FFLD Language

(* use of "S" coil *)


#### 3.2.8.8 IL Language:

```
Op1: FFLD  SET
   S  Q   (* Q is forced to TRUE if SET is TRUE *)
          (* Q is unchanged if SET is FALSE *)
```

**See also**

R RS SR

### 3.2.9 SEMA

*Function Block* - Semaphore.

#### 3.2.9.1 Inputs

| | |
|---|---|
| `CLAIM : BOOL` | Takes the semaphore |
| `RELEASE : BOOL` | Releases the semaphore |

#### 3.2.9.2 Outputs

| | |
|---|---|
| `BUSY : BOOL` | True if semaphore is busy |

#### 3.2.9.3 Remarks

The function block implements the following algorithm:

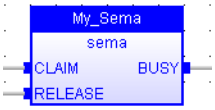
```
BUSY := mem;
if CLAIM then
   mem := TRUE;
else if RELEASE then
   BUSY := FALSE;
   mem := FALSE;
end_if;
```

In FFLD language, the input rung is the CLAIM command. The output rung is the BUSY output signal.

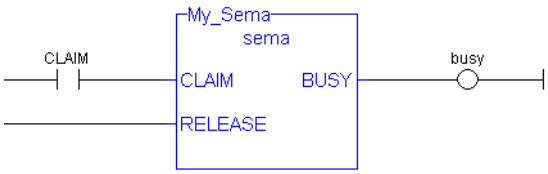
#### 3.2.9.4 ST Language

```
(* MySema is a declared instance of SEMA function block *)
MySema (CLAIM, RELEASE);
BUSY := MyBlinker.BUSY;
```

#### 3.2.9.5 FBD Language




#### 3.2.9.6 FFLD Language




#### 3.2.9.7 IL Language:

```
(* MySema is a declared instance of SEMA function block *)
Op1: CAL MySema (CLAIM, RELEASE)
    FFLD  MyBlinker.BUSY
    ST  BUSY
```

### 3.2.10 SR

PLCopen

*Function Block* - Set dominant bistable.

#### 3.2.10.1 Inputs

`SET1` : `BOOL` Condition for forcing to TRUE (highest priority command)
`RESET` : `BOOL` Condition for forcing to FALSE

#### 3.2.10.2 Outputs

`Q1` : `BOOL` Output to be forced

#### 3.2.10.3 Truth table

| SET1 | RESET | Q1 *prev* | Q1 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

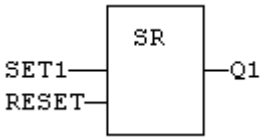
#### 3.2.10.4 Remarks

The output is unchanged when both inputs are FALSE. When both inputs are TRUE, the output is forced to TRUE (set dominant).

#### 3.2.10.5 ST Language

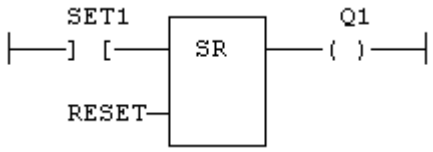
```
(* MySR is declared as an instance of SR function block *)
MySR (SET1, RESET);
Q1 := MySR.Q1;
```

#### 3.2.10.6 FBD Language


SR
SET1
RESET
Q1

#### 3.2.10.7 FFLD Language

(* the SET1 command is the rung - the rung is the output *)




#### 3.2.10.8 IL Language:

```
(* MySR is declared as an instance of SR function block *)
Op1: CAL MySR (SET1, RESET)
     FFLD  MySR.Q1
     ST  Q1
```

**See also**

R S RS

### 3.2.11 XOR XORN PLCopen

*Operator* - Performs an exclusive OR of all inputs.

#### 3.2.11.1 Inputs

`IN1 : BOOL` First Boolean input
`IN2 : BOOL` Second Boolean input

#### 3.2.11.2 Outputs

`Q : BOOL` Exclusive OR of all inputs

#### 3.2.11.3 Truth table

| IN1 | IN2 | Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

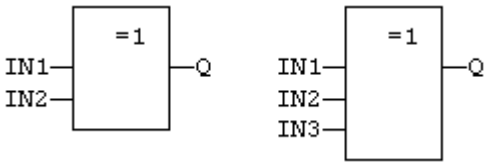
#### 3.2.11.4 Remarks

The block is called "=1" in FBD and FFLD languages. In IL language, the XOR instruction performs an exclusive OR between the current result and the operand. The current result must be Boolean. The XORN instruction performs an exclusive between the current result and the Boolean negation of the operand.

#### 3.2.11.5 ST Language

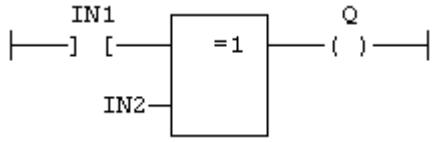
Q := IN1 XOR IN2;
Q := IN1 XOR IN2 XOR IN3;

#### 3.2.11.6 FBD Language




#### 3.2.11.7 FFLD Language

(* First input is the rung. The rung is the output *)




#### 3.2.11.8 IL Language

```
Op1: FFLD   IN1
     XOR  IN2
     ST   Q    (* Q is equal to: IN1 XOR IN2 *)
Op2: FFLD   IN1
     XORN IN2
     ST   Q    (* Q is equal to: IN1 XOR (NOT IN2) *)
```

See also

AND OR NOT

## 3.3 Arithmetic operations PLCopen

Below are the standard operators that perform arithmetic operations:

| | |
|---|---|
| + Addition | addition |
| - Subtraction | subtraction |
| * Multiply | multiplication |
| / Divide | division |
| - (NEG) | integer negation (unary operator) |

Below are the standard functions that perform arithmetic operations:

| | |
|---|---|
| MIN | get the minimum of two integers or an ANY |
| MAX | get the maximum of two integers or an ANY |
| LIMIT | bound an integer to low and high limits or an ANY |
| MOD | modulo |
| ODD | test if an integer is odd |
| SetWithin | Force a value when within an interval |

### 3.3.1 + Addition

*Operator* - Performs an addition of all inputs.

#### 3.3.1.1 Inputs

```
IN1 : ANY    First input
IN2 : ANY    Second input
```

#### 3.3.1.2 Outputs

```
Q : ANY      Result: IN1 + IN2
```

#### 3.3.1.3 Remarks

All inputs and the output must have the same type. In FBD language, the block can have up to 16 inputs. In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the same value as the input rung. In IL language, the ADD instruction performs an addition between the current result and the operand. The current result and the operand must have the same type.

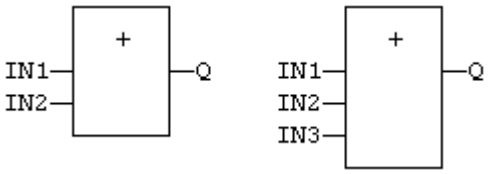
The addition can be used with strings. The result is the concatenation of the input strings.

#### 3.3.1.4 ST Language

```
Q := IN1 + IN2;
MyString := 'He' + 'll ' + 'o';    (* MyString is equal to 'Hello' *)
```

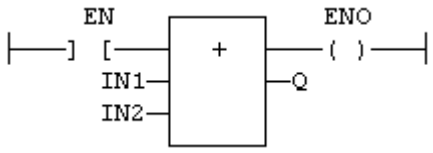
#### 3.3.1.5 FBD Language

(* the block can have up to 16 inputs *)




#### 3.3.1.6 FFLD Language

(* The addition is executed only if EN is TRUE *)
(* ENO is equal to EN *)



#### 3.3.1.7 IL Language

```
Op1: FFLD  IN1
     ADD IN2
     ST  Q    (* Q is equal to: IN1 + IN2 *)
Op2: FFLD  IN1
     ADD IN2
     ADD IN3
     ST  Q    (* Q is equal to: IN1 + IN2 + IN3 *)
```

See also

- * /

### 3.3.2 / Divide

*Operator* - Performs a division of inputs.

#### 3.3.2.1 Inputs

| | | |
|---|---|---|
| IN1 | : ANY_NUM | First input |
| IN2 | : ANY_NUM | Second input |

#### 3.3.2.2 Outputs

| | | |
|---|---|---|
| Q | : ANY_NUM | Result: IN1 / IN2 |

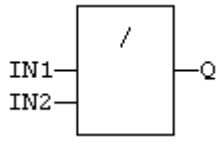
#### 3.3.2.3 Remarks

All inputs and the output must have the same type. In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the same value as the input rung. In IL language, the DIV instruction performs a division between the current result and the operand. The current result and the operand must have the same type.

#### 3.3.2.4 ST Language

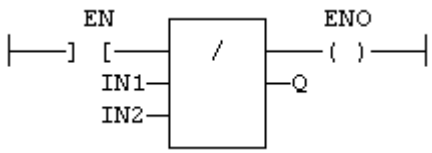
Q := IN1 / IN2;

#### 3.3.2.5 FBD Language




#### 3.3.2.6 FFLD Language

(* The division is executed only if EN is TRUE *)
(* ENO is equal to EN *)




#### 3.3.2.7 IL Language:

```
Op1: FFLD  IN1
    DIV IN2
    ST  Q    (* Q is equal to: IN1 / IN2 *)
Op2: FFLD  IN1
    DIV IN2
    DIV IN3
    ST  Q    (* Q is equal to: IN1 / IN2 / IN3 *)
```

**See also**

+ - *

### 3.3.3 NEG - PLCopen

*Operator* - Performs an integer negation of the input.

#### 3.3.3.1 Inputs

```
IN : DINT  Integer value
```

#### 3.3.3.2 Outputs

```
Q : DINT   Integer negation of the input
```

#### 3.3.3.3 Truth table (examples)

| IN | Q |
|---|---|
| 0 | 0 |
| 1 | -1 |
| -123 | 123 |

#### 3.3.3.4 Remarks

- In FBD and FFLD language, the block "NEG" can be used.
- In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.
- This feature is not available in IL language. In ST language, "-" can be followed by a complex Boolean expression between parentheses.

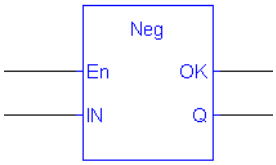
#### 3.3.3.5 ST Language

```
Q := -IN;
Q := - (IN1 + IN2);
```

#### 3.3.3.6 FBD Language


#### 3.3.3.7 FFLD Language

```
(* The negation is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




## 3.3.4 LIMIT

PLCopen

*Function* - Bounds an integer between low and high limits.

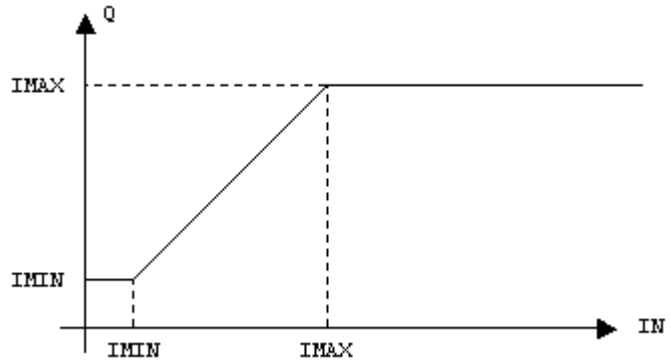
### 3.3.4.1 Inputs

`IMIN : DINT` Low bound
`IN : DINT` Inputvalue
`IMAX : DINT` High bound

### 3.3.4.2 Outputs

`Q : DINT` IMIN if IN < IMIN; IMAX if IN > IMAX; IN otherwise

### 3.3.4.3 Function diagram



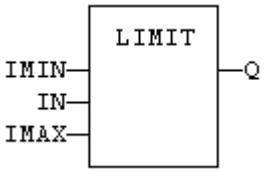

### 3.3.4.4 Remarks

In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the state of the input rung. In IL language, the first input must be loaded before the function call. Other inputs are operands of the function, separated by a coma.

### 3.3.4.5 ST Language

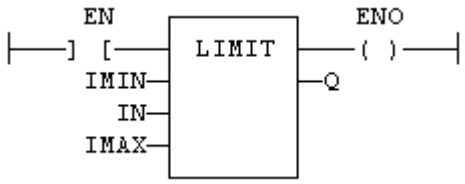
```
Q := LIMIT (IMIN, IN, IMAX);
```

### 3.3.4.6 FBD Language




### 3.3.4.7 FFLD Language

```
(* The comparison is executed only if EN is TRUE *)
(* ENO has the same value as EN *)
```




#### 3.3.4.8 IL Language:

```
Op1: LD    IMIN
     LIMIT IN, IMAX
     ST    Q
```

**See also**

MIN MAX MOD ODD

### 3.3.5 MAX

PLCopen

*Function* - Get the maximum of two integers.

#### 3.3.5.1 Inputs

`IN1 : DINT` First input
`IN2 : DINT` Second input

#### 3.3.5.2 Outputs

`Q : DINT` IN1 if IN1 > IN2; IN2 otherwise

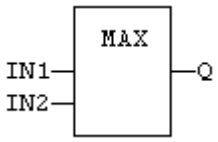
#### 3.3.5.3 Remarks

In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the state of the input rung. In IL language, the first input must be loaded before the function call. The second input is the operand of the function.

#### 3.3.5.4 ST Language

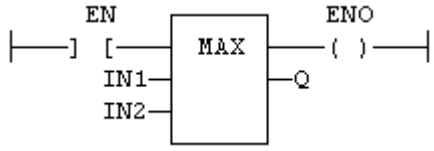
```
Q := MAX (IN1, IN2);
```

#### 3.3.5.5 FBD Language




#### 3.3.5.6 FFLD Language

```
(* The comparison is executed only if EN is TRUE *)
(* ENO has the same value as EN *)
```




#### 3.3.5.7 IL Language:

```
Op1: LD  IN1
     MAX IN2
     ST  Q   (* Q is the maximum of IN1 and IN2 *)
```

**See also**

MIN LIMIT MOD ODD

## 3.3.6 MIN

PLCopen

*Function* - Get the minimum of two integers.

#### 3.3.6.1 Inputs

`IN1 : DINT` First input
`IN2 : DINT` Second input

#### 3.3.6.2 Outputs

`Q : DINT` IN1 if IN1 < IN2; IN2 otherwise

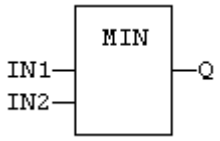
#### 3.3.6.3 Remarks

In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the state of the input rung. In IL language, the first input must be loaded before the function call. The second input is the operand of the function.

#### 3.3.6.4 ST Language

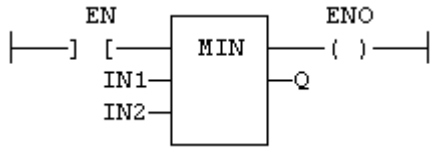
```
Q := MIN (IN1, IN2);
```

#### 3.3.6.5 FBD Language




#### 3.3.6.6 FFLD Language

```
(* The comparison is executed only if EN is TRUE *)
(* ENO has the same value as EN *)
```



#### 3.3.6.7 IL Language:

```
Op1: LD  IN1
     MIN IN2
     ST  Q    (* Q is the minimum of IN1 and IN2 *)
```

**See also**

MAX LIMIT MOD ODD

### 3.3.7 MOD / MODR / MODLR PLCopen

*Function* - Calculation of modulo.

The modulo is calculated as follows:

```
Q = IN - Trunc(IN/BASE) * BASE
```

where *Trunc(x)* calculates the truncated (rounded toward zero) value of x.

**NOTE**

If BASE = 0, then Q will return 0.
If IN = 0, then Q will return 0.

| Inputs | Function | | | Description |
|---|---|---|---|---|
| | MOD | MODR | MODLR | |
| IN | DINT | REAL | LREAL | Input value |
| BASE | DINT | REAL | LREAL | Base of the modulo |

| Output | Function | | | Description | Range |
|---|---|---|---|---|---|
| | MOD | MODR | MODLR | | |
| Q | DINT | REAL | LREAL | Modulo result | (-BASE, BASE) |

#### 3.3.7.1 Examples

*MOD Examples*

| IN | BASE | Q |
|---|---|---|
| 11 | 7 | 4 |
| 7 | 7 | 0 |
| 5 | 7 | 5 |
| 0 | 7 | 0 |
| -5 | 7 | -5 |
| -7 | 7 | 0 |
| -11 | 7 | -4 |
| 11 | -7 | 4 |
| 7 | -7 | 0 |
| 5 | -7 | 5 |
| 0 | -7 | 0 |
| -5 | -7 | -5 |

| IN | BASE | Q |
|---|---|---|
| -7 | -7 | 0 |
| -11 | -7 | -4 |
| 5 | 0 | 0 |

*MODR / MODLR Examples*

| IN | BASE | Q |
|---|---|---|
| 9.00 | 5.75 | 3.25 |
| 5.75 | 5.75 | 0 |
| 2.50 | 5.75 | 2.50 |
| 0 | 5.75 | 0 |
| -2.50 | 5.75 | -2.5 |
| -5.75 | 5.75 | 0 |
| -9.00 | 5.75 | -3.25 |
| 9.00 | -5.75 | 3.25 |
| 5.75 | -5.75 | 0 |
| 2.50 | -5.75 | 2.50 |
| 0 | -5.75 | 0 |
| -2.50 | -5.75 | -2.5 |
| -5.75 | -5.75 | 0 |
| -9.00 | -5.75 | -3.25 |
| 4.25 | 0 | 0 |

#### 3.3.7.2 Remarks

The MOD family of functions can return negative numbers. By adding BASE to the result, the modulo value may be forced to a positive number if the range is required to be [0, BASE). An example of how to accomplish this in ST code follows.

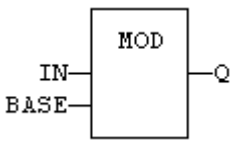
```
q := MOD(x, base);
IF q < 0 THEN
   q := q + base;
END_IF;
```

In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the state of the input rung. In IL language, the first input must be loaded before the function call. The second input is the operand of the function.

#### 3.3.7.3 ST Language

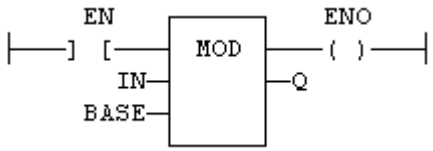
Q := MOD (IN, BASE);

#### 3.3.7.4 FBD Language




#### 3.3.7.5 FFLD Language

```
(* The comparison is executed only if EN is TRUE *)
(* ENO has the same value as EN *)
```




#### 3.3.7.6 IL Language

```
Op1: LD  IN
     MOD BASE
     ST  Q    (* Q is the rest of integer division: IN / BASE *)
```

**See also**

MIN MAX LIMIT ODD

### 3.3.8 * Multiply

*Operator* - Performs a multiplication of all inputs.

#### 3.3.8.1 Inputs

| | |
|---|---|
| `IN1 : ANY_NUM` | First input |
| `IN2 : ANY_NUM` | Second input |

#### 3.3.8.2 Outputs

| | |
|---|---|
| `Q : ANY_NUM` | Result: IN1 * IN2 |

#### 3.3.8.3 Remarks

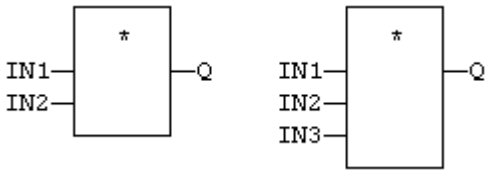
All inputs and the output must have the same type. In FBD language, the block can have up to 16 inputs. In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the same value as the input rung. In IL language, the MUL instruction performs a multiplication between the current result and the operand. The current result and the operand must have the same type.

#### 3.3.8.4 ST Language

```
Q := IN1 * IN2;
```

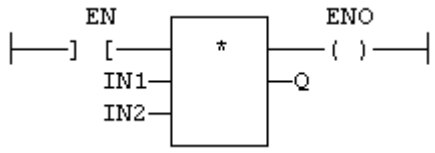
#### 3.3.8.5 FBD Language

```
(* the block can have up to 16 inputs *)
```




#### 3.3.8.6 FFLD Language

```
(* The multiplication is executed only if EN is TRUE *)
(* ENO is equal to EN *)
```




#### 3.3.8.7 IL Language:

```
Op1: FFLD  IN1
     MUL IN2
     ST  Q    (* Q is equal to: IN1 * IN2 *)
Op2: FFLD  IN1
     MUL IN2
     MUL IN3
     ST  Q    (* Q is equal to: IN1 * IN2 * IN3 *)
```

**See also**

+ - /

### 3.3.9 ODD

PLCopen

*Function* - Test if an integer is odd

#### 3.3.9.1 Inputs

`IN : DINT` Input value

#### 3.3.9.2 Outputs

`Q : BOOL` TRUE if IN is odd. FALSE if IN is even.

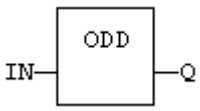
#### 3.3.9.3 Remarks

In FFLD language, the input rung (EN) enables the operation, and the output rung is the result of the function. In IL language, the input must be loaded before the function call.

#### 3.3.9.4 ST Language

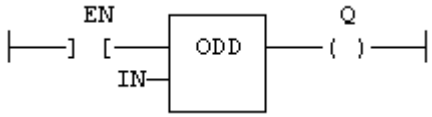
```
Q := ODD (IN);
```

#### 3.3.9.5 FBD Language




#### 3.3.9.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
```




#### 3.3.9.7 IL Language:

```
Op1: LD  IN
   ODD
   ST  Q   (* Q is TRUE if IN is odd *)
```

**See also**

MIN MAX LIMIT MOD

### 3.3.10 SetWithin PLCopen

*Function* - Force a value when inside an interval.

#### 3.3.10.1 Inputs

`IN : ANY` Input
`MIN : ANY` Low limit of the interval
`MAX : ANY` High limit of the interval
`VAL : ANY` Value to apply when inside the interval

#### 3.3.10.2 Outputs

`Q : ANY` Result

#### 3.3.10.3 Truth Table

| In | Q |
|---|---|
| IN < MIN | IN |
| IN > MAX | IN |
| MIN < IN < MAX | VAL |

#### 3.3.10.4 Remarks

The output is forced to VAL when the IN value is within the [MIN ... MAX] interval. It is set to IN when outside the interval.

### 3.3.11 - Subtraction

*Operator* - Performs a subtraction of inputs.

#### 3.3.11.1 Inputs

`IN1 : ANY_NUM / TIME` First input
`IN2 : ANY_NUM / TIME` Second input

#### 3.3.11.2 Outputs

`Q : ANY_NUM / TIME` Result: IN1 - IN2

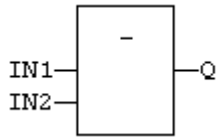
#### 3.3.11.3 Remarks

All inputs and the output must have the same type. In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the same value as the input rung. In IL language, the SUB instruction performs a subtraction between the current result and the operand. The current result and the operand must have the same type.

#### 3.3.11.4 ST Language

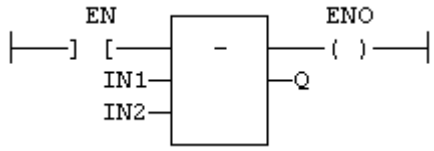
```
Q := IN1 - IN2;
```

### 3.3.11.5 FBD Language




### 3.3.11.6 FFLD Language

(* The subtraction is executed only if EN is TRUE *)
(* ENO is equal to EN *)




### 3.3.11.7 IL Language:

```
Op1: FFLD  IN1
     SUB IN2
     ST  Q    (* Q is equal to: IN1 - IN2 *)
Op2: FFLD  IN1
     SUB IN2
     SUB IN3
     ST  Q    (* Q is equal to: IN1 - IN2 - IN3 *)
```

**See also**

+ * /

## 3.4 Comparison Operations PLCopen

Below are the standard operators and blocks that perform comparisons:

| | |
|---|---|
| < | less than |
| > | greater than |
| <= | less or equal |
| >= | greater or equal |
| = | is equal |
| <> | is not equal |
| CMP | detailed comparison |

### 3.4.1 CMP PLCopen

*Function Block* - Comparison with detailed outputs for integer inputs

#### 3.4.1.1 Inputs

```
IN1 : DINT   First value
IN2 : DINT   Second value
```

#### 3.4.1.2 Outputs

```
LT : BOOL   TRUE if IN1 < IN2
EQ : BOOL   TRUE if IN1 = IN2
GT : BOOL   TRUE if IN1 > IN2
```

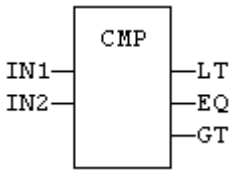
#### 3.4.1.3 Remarks

In FFLD language, the rung input (EN) validates the operation. The rung output is the result of "LT" (lower than) comparison).

#### 3.4.1.4 ST Language

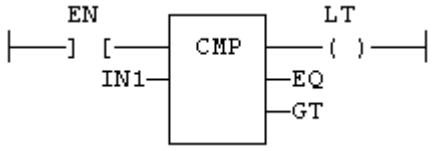
```
(* MyCmp is declared as an instance of CMP function block *)
MyCMP (IN1, IN2);
bLT := MyCmp.LT;
bEQ := MyCmp.EQ;
bGT := MyCmp.GT;
```

#### 3.4.1.5 FBD Language




#### 3.4.1.6 FFLD Language

```
(* the comparison is performed only if EN is TRUE  *)
```




#### 3.4.1.7 IL Language:

```
(* MyCmp is declared as an instance of CMP function block *)
Op1: CAL MyCmp (IN1, IN2)
    LD  MyCmp.LT
    ST  bLT
    LD  MyCmp.EQ
    ST  bEQ
    LD  MyCmp.GT
    ST  bGT
```

**See also**

> < >= <= = <>

### 3.4.2 >= GE PLCopen

*Operator* - Test if first input is greater than or equal to second input.

#### 3.4.2.1 Inputs

| | |
|---|---|
| `IN1 : ANY` | First input |
| `IN2 : ANY` | Second input |

#### 3.4.2.2 Outputs

| | |
|---|---|
| `Q : BOOL` | TRUE if IN1 >= IN2 |

#### 3.4.2.3 Remarks

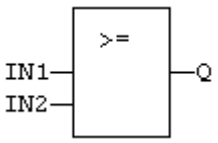
Both inputs must have the same type. In FFLD language, the input rung (EN) enables the operation, and the output rung is the result of the comparison. In IL language, the GE instruction performs the comparison between the current result and the operand. The current result and the operand must have the same type.

Comparisons can be used with strings. In that case, the lexical order is used for comparing the input strings. For instance, "ABC" is less than "ZX" ; "ABCD" is greater than "ABC".

#### 3.4.2.4 ST Language

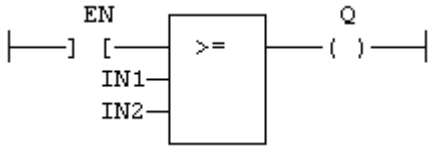
```
Q := IN1 >= IN2;
```

#### 3.4.2.5 FBD Language




#### 3.4.2.6 FFLD Language

```
(* The comparison is executed only if EN is TRUE *)
```




#### 3.4.2.7 IL Language:

```
Op1: FFLD  IN1
     GE  IN2
     ST  Q    (* Q is true if IN1 >= IN2 *)
```

**See also**

> < <= = <> CMP

### 3.4.3 > GT PLCopen

*Operator* - Test if first input is greater than second input.

#### 3.4.3.1 Inputs

| | |
|---|---|
| `IN1 : ANY` | First input |
| `IN2 : ANY` | Second input |

#### 3.4.3.2 Outputs

| | |
|---|---|
| `Q : BOOL` | TRUE if IN1 > IN2 |

#### 3.4.3.3 Remarks

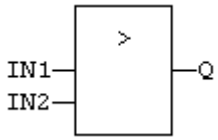
Both inputs must have the same type. In FFLD language, the input rung (EN) enables the operation, and the output rung is the result of the comparison. In IL language, the GT instruction performs the comparison between the current result and the operand. The current result and the operand must have the same type.

Comparisons can be used with strings. In that case, the lexical order is used for comparing the input strings. For instance, "ABC" is less than "ZX" ; "ABCD" is greater than "ABC".

#### 3.4.3.4 ST Language

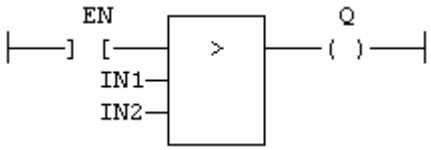
```
Q := IN1 > IN2;
```

#### 3.4.3.5 FBD Language




#### 3.4.3.6 FFLD Language

```
(* The comparison is executed only if EN is TRUE *)
```



#### 3.4.3.7 IL Language:

```
Op1: FFLD  IN1
     GT  IN2
     ST  Q    (* Q is true if IN1 > IN2 *)
```

**See also**

< >= <= = <> CMP

### 3.4.4 = EQ PLCopen

*Operator* - Test if first input is equal to second input.

#### 3.4.4.1 Inputs

| | |
|---|---|
| `IN1 : ANY` | First input |
| `IN2 : ANY` | Second input |

#### 3.4.4.2 Outputs

| | |
|---|---|
| `Q : BOOL` | TRUE if IN1 = IN2 |

#### 3.4.4.3 Remarks

Both inputs must have the same type. In FFLD language, the input rung (EN) enables the operation, and the output rung is the result of the comparison. In IL language, the EQ instruction performs the comparison between the current result and the operand. The current result and the operand must have the same type.

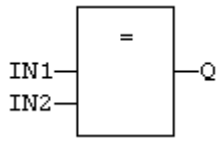
Comparisons can be used with strings. In that case, the lexical order is used for comparing the input strings. For instance, "ABC" is less than "ZX" ; "ABCD" is greater than "ABC".

Equality comparisons cannot be used with TIME variables. The reason is that the timer actually has the resolution of the target cycle and test can be unsafe as some values can never be reached.

#### 3.4.4.4 ST Language

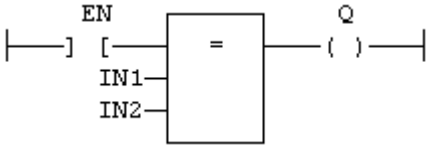
```
Q := IN1 = IN2;
```

#### 3.4.4.5 FBD Language




#### 3.4.4.6 FFLD Language

```
(* The comparison is executed only if EN is TRUE *)
```




#### 3.4.4.7 IL Language:

```
Op1: FFLD  IN1
   EQ  IN2
   ST  Q   (* Q is true if IN1 = IN2 *)
```

**See also**

> < >= <= <> CMP

## 3.4.5 <> NE PLCopen

*Operator* - Test if first input is not equal to second input.

### 3.4.5.1 Inputs

| | |
|---|---|
| `IN1 : ANY` | First input |
| `IN2 : ANY` | Second input |

### 3.4.5.2 Outputs

| | |
|---|---|
| `Q : BOOL` | TRUE if IN1 is not equal to IN2 |

### 3.4.5.3 Remarks

Both inputs must have the same type. In FFLD language, the input rung (EN) enables the operation, and the output rung is the result of the comparison. In IL language, the NE instruction performs the comparison between the current result and the operand. The current result and the operand must have the same type.

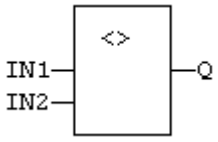
Comparisons can be used with strings. In that case, the lexical order is used for comparing the input strings. For instance, "ABC" is less than "ZX" ; "ABCD" is greater than "ABC".

Equality comparisons cannot be used with TIME variables. The reason is that the timer actually has the resolution of the target cycle and test can be unsafe as some values can never be reached

### 3.4.5.4 ST Language

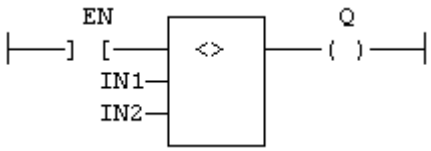
```
Q := IN1 <> IN2;
```

### 3.4.5.5 FBD Language




### 3.4.5.6 FFLD Language

```
(* The comparison is executed only if EN is TRUE *)
```




### 3.4.5.7 IL Language:

```
Op1: FFLD  IN1
   NE  IN2
   ST  Q   (* Q is true if IN1 is not equal to IN2 *)
```

**See also**

> < >= <= = CMP

## 3.4.6 <= LE PLCopen

*Operator* - Test if first input is less than or equal to second input.

### 3.4.6.1 Inputs

| | |
|---|---|
| `IN1 : ANY` | First input |
| `IN2 : ANY` | Second input |

### 3.4.6.2 Outputs

| | |
|---|---|
| `Q : BOOL` | TRUE if IN1 <= IN2 |

### 3.4.6.3 Remarks

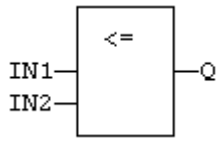
Both inputs must have the same type. In FFLD language, the input rung (EN) enables the operation, and the output rung is the result of the comparison. In IL language, the LE instruction performs the comparison between the current result and the operand. The current result and the operand must have the same type.

Comparisons can be used with strings. In that case, the lexical order is used for comparing the input strings. For instance, "ABC" is less than "ZX" ; "ABCD" is greater than "ABC".

### 3.4.6.4 ST Language

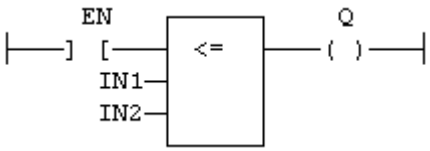
```
Q := IN1 <= IN2;
```

### 3.4.6.5 FBD Language




### 3.4.6.6 FFLD Language

```
(* The comparison is executed only if EN is TRUE *)
```




### 3.4.6.7 IL Language:

```
Op1: FFLD  IN1
     LE  IN2
     ST  Q    (* Q is true if IN1 <= IN2 *)
```

**See also**

> < >= = <> CMP

## 3.4.7 < LT PLCopen

*Operator* - Test if first input is less than second input.

#### 3.4.7.1 Inputs

| | |
|---|---|
| `IN1 : ANY` | First input |
| `IN2 : ANY` | Second input |

#### 3.4.7.2 Outputs

| | |
|---|---|
| `Q : BOOL` | TRUE if IN1 < IN2 |

#### 3.4.7.3 Remarks

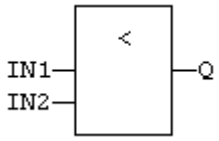
Both inputs must have the same type. In FFLD language, the input rung (EN) enables the operation, and the output rung is the result of the comparison. In IL language, the LT instruction performs the comparison between the current result and the operand. The current result and the operand must have the same type.

Comparisons can be used with strings. In that case, the lexical order is used for comparing the input strings. For instance, "ABC" is less than "ZX" ; "ABCD" is greater than "ABC".

#### 3.4.7.4 ST Language

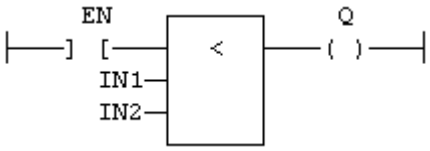
```
Q := IN1 < IN2;
```

#### 3.4.7.5 FBD Language




#### 3.4.7.6 FFLD Language

```
(* The comparison is executed only if EN is TRUE *)
```




#### 3.4.7.7 IL Language:

```
Op1: FFLD  IN1
   LT  IN2
   ST  Q    (* Q is true if IN1 < IN2 *)
```

**See also**

> >= <= = <> CMP

## 3.5 Type conversion functions

Below are the standard functions for converting a data into another data type:

| | |
|---|---|
| ANY_TO_BOOL | converts to Boolean |
| ANY_TO_SINT / ANY_TO_USINT | converts to small (8 bit) integer |
| ANY_TO_INT / ANY_TO_UINT | converts to 16 bit integer |
| ANY_TO_DINT / ANY_TO_UDINT | converts to integer (32 bit - default) |
| ANY_TO_LINT / ANY_TO_ULINT | converts to long (64 bit) integer |
| ANY_TO_REAL | converts to real |
| ANY_TO_LREAL | converts to double precision real |
| ANY_TO_TIME | converts to time |
| ANY_TO_STRING | converts to character string |

Below are the standard functions performing conversions in BCD format (*):

| | |
|---|---|
| BIN_TO_BCD | converts a binary value to a BCD value |
| BCD_TO_BIN | converts a BCD value to a binary value |

(*) BCD conversion functions may not be supported by all targets.

### 3.5.1 ANY_TO_BOOL PLCopen

*Operator* - Converts the input into Boolean value.

#### 3.5.1.1 Inputs

`IN : ANY` Input value

#### 3.5.1.2 Outputs

`Q : BOOL` Value converted to Boolean

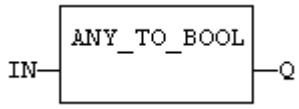
#### 3.5.1.3 Remarks

For DINT, REAL and TIME input data types, the result is FALSE if the input is 0. The result is TRUE in all other cases. For STRING inputs, the output is TRUE if the input string is not empty, and FALSE if the string is empty. In FFLD language, the conversion is executed only if the input rung (EN) is TRUE. The output rung is the result of the conversion. In IL Language, the ANY_TO_BOOL function converts the current result.

#### 3.5.1.4 ST Language

```
Q := ANY_TO_BOOL (IN);
```

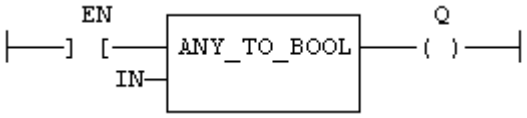
#### 3.5.1.5 FBD Language




#### 3.5.1.6 FFLD Language

```
(* The conversion is executed only if EN is TRUE *)
(* The output rung is the result of the conversion *)
```

```
(* The output rung is FALSE if the EN is FALSE *)
```




#### 3.5.1.7 IL Language:

```
Op1: FFLD  IN
   ANY_TO_BOOL
   ST  Q
```

#### 3.5.1.8 See also

ANY_TO_SINT ANY_TO_INT ANY_TO_DINT ANY_TO_LINT ANY_TO_REAL ANY_TO_LREAL ANY_TO_TIME ANY_TO_STRING

### 3.5.2 ANY_TO_DINT / ANY_TO_UDINT PLCopen

*Operator* - Converts the input into integer value (can be unsigned with ANY_TO_UDINT).

#### 3.5.2.1 Inputs

`IN : ANY` Input value

#### 3.5.2.2 Outputs

`Q : DINT` Value converted to integer

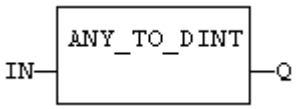
#### 3.5.2.3 Remarks

For BOOL input data types, the output is 0 or 1. For REAL input data type, the output is the integer part of the input real. For TIME input data types, the result is the number of milliseconds. For STRING inputs, the output is the number represented by the string, or 0 if the string does not represent a valid number. In FFLD language, the conversion is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung. In IL Language, the ANY_TO_DINT function converts the current result.

#### 3.5.2.4 ST Language

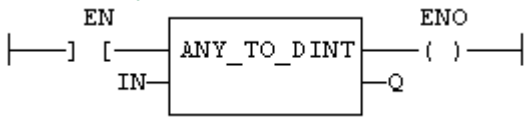
```
Q := ANY_TO_DINT (IN);
```

#### 3.5.2.5 FBD Language




#### 3.5.2.6 FFLD Language

```
(* The conversion is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.5.2.7 IL Language:

Op1: FFLD IN
ANY_TO_DINT
ST Q

#### 3.5.2.8 See also

ANY_TO_BOOL ANY_TO_SINT ANY_TO_INT ANY_TO_LINT ANY_TO_REAL ANY_TO_LREAL ANY_TO_TIME ANY_TO_STRING

### 3.5.3 ANY_TO_INT / ANY_TO_UINT PLCopen

*Operator* - Converts the input into 16 bit integer value (can be unsigned with ANY_TO_UINT).

#### 3.5.3.1 Inputs

`IN : ANY` Input value

#### 3.5.3.2 Outputs

`Q : INT` Value converted to 16 bit integer

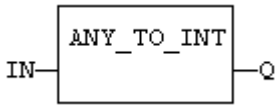
#### 3.5.3.3 Remarks

For BOOL input data types, the output is 0 or 1. For REAL input data type, the output is the integer part of the input real. For TIME input data types, the result is the number of milliseconds. For STRING inputs, the output is the number represented by the string, or 0 if the string does not represent a valid number. In FFLD language, the conversion is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung. In IL Language, the ANY_TO_INT function converts the current result.

#### 3.5.3.4 ST Language

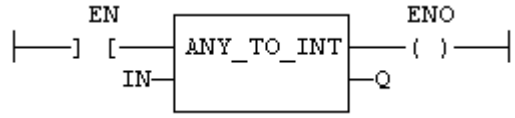
```
Q := ANY_TO_INT (IN);
```

#### 3.5.3.5 FBD Language




#### 3.5.3.6 FFLD Language

```
(* The conversion is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.5.3.7 IL Language:

Op1: FFLD IN
ANY_TO_INT
ST Q

#### 3.5.3.8 See also

ANY_TO_BOOL ANY_TO_SINT ANY_TO_DINT ANY_TO_LINT ANY_TO_REAL ANY_TO_LREAL ANY_TO_TIME ANY_TO_STRING

## 3.5.4 ANY_TO_LINT / ANY_TO_ULINT PLCopen

*Operator* - Converts the input into long (64 bit) integer value (can be unsigned with ANY_TO_ULINT).

### 3.5.4.1 Inputs

`IN : ANY` Input value

### 3.5.4.2 Outputs

`Q : LINT` Value converted to long (64 bit) integer

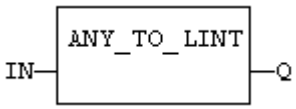
### 3.5.4.3 Remarks

For BOOL input data types, the output is 0 or 1. For REAL input data type, the output is the integer part of the input real. For TIME input data types, the result is the number of milliseconds. For STRING inputs, the output is the number represented by the string, or 0 if the string does not represent a valid number. In FFLD language, the conversion is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung. In IL Language, the ANY_TO_LINT function converts the current result.

### 3.5.4.4 ST Language

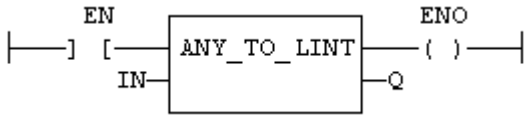
```
Q := ANY_TO_LINT (IN);
```

### 3.5.4.5 FBD Language




### 3.5.4.6 FFLD Language

```
(* The conversion is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




### 3.5.4.7 IL Language:

```
Op1: FFLD  IN
   ANY_TO_LINT
   ST  Q
```

### 3.5.4.8 See also

ANY_TO_BOOL ANY_TO_SINT ANY_TO_INT ANY_TO_DINT ANY_TO_REAL ANY_TO_LREAL ANY_TO_TIME ANY_TO_STRING

## 3.5.5 ANY_TO_LREAL PLCopen

*Operator* - Converts the input into double precision real value.

### 3.5.5.1 Inputs

`IN : ANY` Input value

#### 3.5.5.2 Outputs

`Q : LREAL` Value converted to double precision real

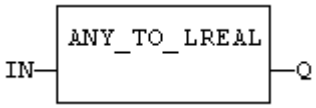
#### 3.5.5.3 Remarks

For BOOL input data types, the output is 0.0 or 1.0. For DINT input data type, the output is the same number. For TIME input data types, the result is the number of milliseconds. For STRING inputs, the output is the number represented by the string, or 0.0 if the string does not represent a valid number. In FFLD language, the conversion is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung. In IL Language, the ANY_TO_LREAL function converts the current result.

#### 3.5.5.4 ST Language

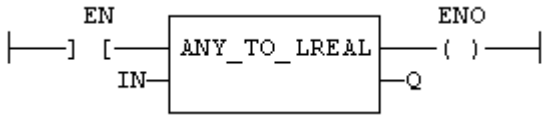
```
Q := ANY_TO_LREAL (IN);
```

#### 3.5.5.5 FBD Language




#### 3.5.5.6 FFLD Language

```
(* The conversion is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.5.5.7 IL Language:

```
Op1: FFLD  IN
   ANY_TO_LREAL
   ST  Q
```

#### 3.5.5.8 See also

ANY_TO_BOOL ANY_TO_SINT ANY_TO_INT ANY_TO_DINT ANY_TO_LINT ANY_TO_REAL ANY_TO_TIME ANY_TO_STRING

### 3.5.6 ANY_TO_REAL PLCopen

*Operator* - Converts the input into real value.

#### 3.5.6.1 Inputs

`IN : ANY` Input value

#### 3.5.6.2 Outputs

`Q : REAL` Value converted to real

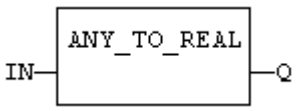
#### 3.5.6.3 Remarks

For BOOL input data types, the output is 0.0 or 1.0. For DINT input data type, the output is the same number. For TIME input data types, the result is the number of milliseconds. For STRING inputs, the output is the number represented by the string, or 0.0 if the string does not represent a valid number. In FFLD language, the conversion is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung. In IL Language, the ANY_TO_REAL function converts the current result.

#### 3.5.6.4 ST Language

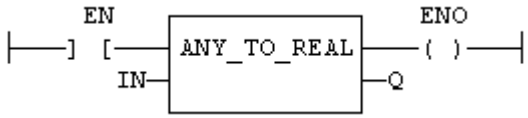
```
Q := ANY_TO_REAL (IN);
```

#### 3.5.6.5 FBD Language




#### 3.5.6.6 FFLD Language

```
(* The conversion is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.5.6.7 IL Language:

```
Op1: FFLD  IN
   ANY_TO_REAL
   ST  Q
```

#### 3.5.6.8 See also

ANY_TO_BOOL ANY_TO_SINT ANY_TO_INT ANY_TO_DINT ANY_TO_LINT ANY_TO_LREAL ANY_TO_TIME ANY_TO_STRING

### 3.5.7 ANY_TO_TIME PLCopen

*Operator* - Converts the input into time value.

#### 3.5.7.1 Inputs

`IN : ANY` Input value

#### 3.5.7.2 Outputs

`Q : TIME` Value converted to time

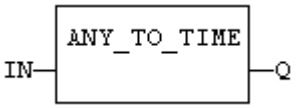
#### 3.5.7.3 Remarks

For BOOL input data types, the output is t#0 ms or t#1 ms. For DINT or REAL input data type, the output is the time represented by the input number as a number of milliseconds. For STRING inputs, the output is the time represented by the string, or t#0 ms if the string does not represent a valid time. In FFLD language, the conversion is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung. In IL Language, the ANY_TO_TIME function converts the current result.

#### 3.5.7.4 ST Language

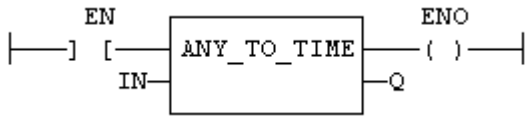
Q := ANY_TO_TIME (IN);

#### 3.5.7.5 FBD Language




#### 3.5.7.6 FFLD Language

(* The conversion is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)




#### 3.5.7.7 IL Language:

Op1: FFLD IN
ANY_TO_TIME
ST Q

#### 3.5.7.8 See also

ANY_TO_BOOL ANY_TO_SINT ANY_TO_INT ANY_TO_DINT ANY_TO_LINT ANY_TO_REAL ANY_TO_LREAL ANY_TO_STRING

### 3.5.8 ANY_TO_SINT / ANY_TO_USINT PLCopen

*Operator* - Converts the input into a small (8 bit) integer value (can be unsigned with ANY_TO_USINT).

#### 3.5.8.1 Inputs

`IN : ANY` Input value

#### 3.5.8.2 Outputs

`Q : SINT` Value converted to a small (8 bit) integer

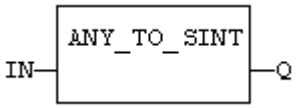
#### 3.5.8.3 Remarks

For BOOL input data types, the output is 0 or 1. For REAL input data type, the output is the integer part of the input real. For TIME input data types, the result is the number of milliseconds. For STRING inputs, the output is the number represented by the string, or 0 if the string does not represent a valid number. In FFLD language, the conversion is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung. In IL Language, the ANY_TO_SINT function converts the current result.

#### 3.5.8.4 ST Language

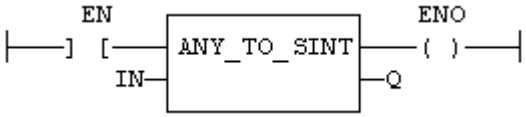
Q := ANY_TO_SINT (IN);

#### 3.5.8.5 FBD Language




#### 3.5.8.6 FFLD Language

```
(* The conversion is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.5.8.7 IL Language

```
Op1: FFLD  IN
   ANY_TO_SINT
   ST  Q
```

#### 3.5.8.8 See also

ANY_TO_BOOL ANY_TO_INT ANY_TO_DINT ANY_TO_LINT ANY_TO_REAL ANY_TO_LREAL ANY_TO_TIME ANY_TO_STRING

### 3.5.9 ANY_TO_STRING PLCopen

*Operator* - Converts the input into string value.

#### 3.5.9.1 Inputs

`IN : ANY` Input value

#### 3.5.9.2 Outputs

`Q : STRING` Value converted to string

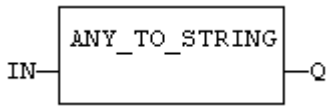
#### 3.5.9.3 Remarks

For BOOL input data types, the output is '1' or '0' for TRUE and FALSE respectively. For DINT, REAL or TIME input data types, the output is the string representation of the input number. It is a number of milliseconds for TIME inputs. In FFLD language, the conversion is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung. In IL language, the ANY_TO_STRING function converts the current result.

#### 3.5.9.4 ST Language

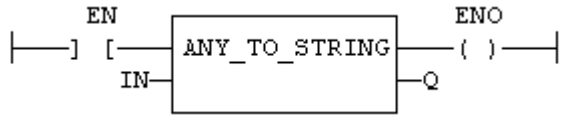
```
Q := ANY_TO_STRING (IN);
```

#### 3.5.9.5 FBD Language




#### 3.5.9.6 FFLD Language

```
(* The conversion is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.5.9.7 IL Language:

```
Op1: FFLD  IN
   ANY_TO_STRING
   ST  Q
```

#### 3.5.9.8 See also

ANY_TO_BOOL ANY_TO_SINT ANY_TO_INT ANY_TO_DINT ANY_TO_LINT ANY_TO_REAL ANY_TO_LREAL ANY_TO_TIME

### 3.5.10 NUM_TO_STRING PLCopen

*Function*- Converts a number into string value.

#### 3.5.10.1 Inputs

IN : ANY Input number.
WIDTH : DINT Wished length for the output string (see remarks)
DIGITS : DINT Number of digits after decimal point

#### 3.5.10.2 Outputs

Q : STRING Value converted to string.

#### 3.5.10.3 Remarks

This function converts any numerical value to a string. Unlike the ANY_TO_STRING function, it allows you to specify a wished length and a number of digits after the decimal points.

If WIDTH is 0, the string is formatted with the necessary length.

If WIDTH is greater than 0, the string is completed with heading blank characters in order to match the value of WIDTH.

If WIDTH is greater than 0, the string is completed with trailing blank characters in order to match the absolute value of WIDTH.

If DIGITS is 0 then neither decimal part nor point are added.

If DIGITS is greater than 0, the corresponding number of decimal digits are added. '0' digits are added if necessary

If the value is too long for the specified width, then the string is filled with '*' characters.

#### 3.5.10.4 Examples

```
Q := NUM_TO_STRING (123.4, 8, 2); (* Q is ' 123.40' *)
Q := NUM_TO_STRING (123.4, -8, 2); (* Q is '123.40 ' *)
Q := NUM_TO_STRING (1.333333, 0, 2); (* Q is '1.33' *)
Q := NUM_TO_STRING (1234, 3, 0); (* Q is '***' *)
```

### 3.5.11 BCD_TO_BIN PLCopen

*Function* - Converts a BCD (Binary Coded Decimal) value to a binary value

#### 3.5.11.1 Inputs

IN : DINT Integer value in BCD

#### 3.5.11.2 Outputs

Q : DINT Value converted to integer, or 0 if IN is not a valid positive BCD value

##### 3.5.11.2.1 Truth table (examples)

| IN | Q |
|---|---|
| -2 | 0 (invalid) |
| 0 | 0 |
| 16 (16#10) | 10 |
| 15 (16#0F) | 0 (invalid) |

#### 3.5.11.3 Remarks

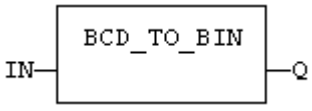
The input must be positive and must represent a valid BCD value. In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.5.11.4 ST Language

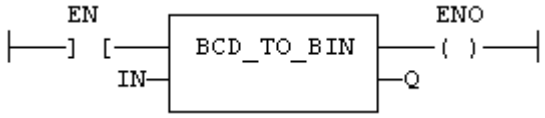
```
Q := BCD_TO_BIN (IN);
```

#### 3.5.11.5 FBD Language




#### 3.5.11.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.5.11.7 IL Language

```
Op1: LD          IN
     BCD_TO_BIN
     ST          Q
```

See also

BIN_TO_BCD

### 3.5.12 BIN_TO_BCD PLCopen

*Function* - Converts a binary value to a BCD (Binary Coded Decimal) value

#### 3.5.12.1 Inputs

`IN : DINT` Integer value

#### 3.5.12.2 Outputs

`Q : DINT` Value converted to BCD
or 0 if IN is less than 0

#### 3.5.12.3 Truth table (examples)

| IN | Q |
|---|---|
| -2 | 0 (invalid) |
| 0 | 0 |
| 10 | 16 (16#10) |
| 22 | 34 (16#22) |

#### 3.5.12.4 Remarks

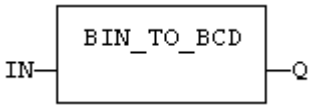
The input must be positive. In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.5.12.5 ST Language

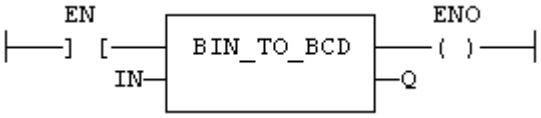
```
Q := BIN_TO_BCD (IN);
```

#### 3.5.12.6 FBD Language




#### 3.5.12.7 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.5.12.8 IL Language:

```
Op1: LD    IN
   BIN_TO_BCD
   ST    Q
```

See also

BCD_TO_BIN

## 3.6 Selectors

Below are the standard functions that perform data selection:

| | |
|---|---|
| SEL | 2 integer inputs |
| MUX4 | 4 integer inputs |
| MUX8 | 8 integer inputs |

### 3.6.1 MUX4

PLCopen

*Function* - Select one of the inputs - 4 inputs.

#### 3.6.1.1 Inputs

```
SELECT : DINT  Selection command
IN1    : ANY   First input
IN2    : ANY   Second input
...    :
IN4    : ANY   Last input
```

#### 3.6.1.2 Outputs

```
Q : ANY
```
IN1 or IN2 ... or IN4 depending on SELECT (see truth table)

#### 3.6.1.3 Truth table

| SELECT | Q |
|---|---|
| 0 | IN1 |
| 1 | IN2 |
| 2 | IN3 |
| 3 | IN4 |
| other | 0 |

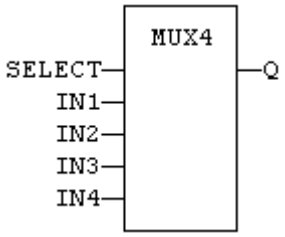
#### 3.6.1.4 Remarks

In FFLD language, the input rung (EN) enables the selection. The output rung keeps the same state as the input rung. In IL language, the first parameter (selector) must be loaded in the current result before calling the function. Other inputs are operands of the function, separated by comas.

#### 3.6.1.5 ST Language

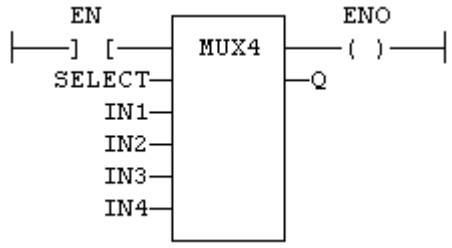
```
Q := MUX4 (SELECT, IN1, IN2, IN3, IN4);
```

#### 3.6.1.6 FBD Language




#### 3.6.1.7 FFLD Language

```
(* the selection is performed only if EN is TRUE *)
(* ENO has the same value as EN *)
```




#### 3.6.1.8 IL Language

```
Op1: LD   SELECT
     MUX4 IN1, IN2, IN3, IN4
     ST   Q
```

**See also**

SEL MUX8

### 3.6.2 MUX8

PLCopen

*Function* - Select one of the inputs - 8 inputs.

#### 3.6.2.1 Inputs

| | | |
|---|---|---|
| SELECT | : DINT | Selection command |
| IN1 | : ANY | First input |
| IN2 | : ANY | Second input |
| ... | : | |
| IN8 | : ANY | Last input |

#### 3.6.2.2 Outputs

Q : ANY IN1 or IN2 ... or IN8 depending on SELECT (see truth table)

#### 3.6.2.3 Truth table

| SELECT | Q |
|---|---|
| 0 | IN1 |
| 1 | IN2 |
| 2 | IN3 |
| 3 | IN4 |
| 4 | IN5 |
| 5 | IN6 |
| 6 | IN7 |
| 7 | IN8 |
| other | 0 |

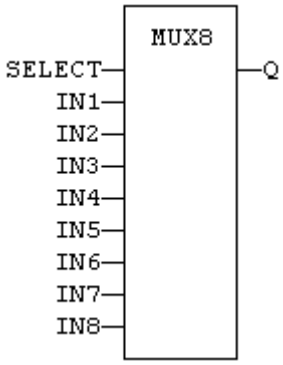
#### 3.6.2.4 Remarks

In FFLD language, the input rung (EN) enables the selection. The output rung keeps the same state as the input rung. In IL language, the first parameter (selector) must be loaded in the current result before calling the function. Other inputs are operands of the function, separated by comas.

#### 3.6.2.5 ST Language

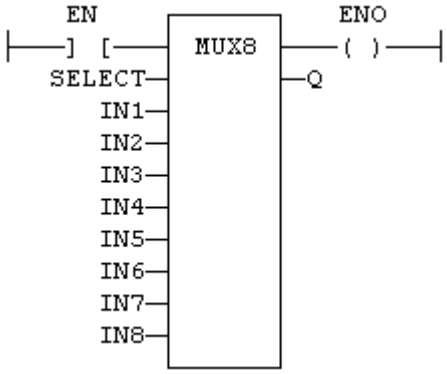
```
Q := MUX8 (SELECT, IN1, IN2, IN3, IN4, IN5, IN6, IN7, IN8);
```

#### 3.6.2.6 FBD Language




#### 3.6.2.7 FFLD Language

```
(* the selection is performed only if EN is TRUE *)
(* ENO has the same value as EN *)
```




#### 3.6.2.8 IL Language

Not available

```
Op1: LD   SELECT
     MUX8 IN1, IN2, IN3, IN4, IN5, IN6, IN7, IN8
     ST   Q
```

**See also**

SEL MUX4

### 3.6.3 SEL PLCopen

*Function* - Select one of the inputs - 2 inputs.

#### 3.6.3.1 Inputs

| | | |
|---|---|---|
| SELECT | : BOOL | Selection command |
| IN1 | : ANY | First input |
| IN2 | : ANY | Second input |

#### 3.6.3.2 Outputs

`Q : ANY` IN1 if SELECT is FALSE; IN2 if SELECT is TRUE

#### 3.6.3.3 Truth table

| SELECT | Q |
|---|---|
| 0 | IN1 |
| 1 | IN2 |

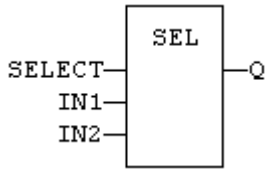
#### 3.6.3.4 Remarks

In FFLD language, the selector command is the input rung. The output rung keeps the same state as the input rung. In IL language, the first parameter (selector) must be loaded in the current result before calling the function. Other inputs are operands of the function, separated by comas.

#### 3.6.3.5 ST Language

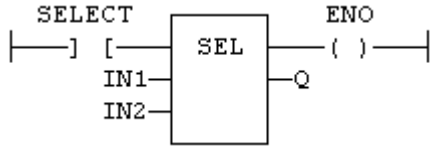
```
Q := SEL (SELECT, IN1, IN2);
```

#### 3.6.3.6 FBD Language




#### 3.6.3.7 FFLD Language

```
(* the input rung is the selector *)
(* ENO has the same value as SELECT *)
```




#### 3.6.3.8 IL Language

```
Op1: LD  SELECT
     SEL IN1, IN2
     ST  Q
```

**See also**

MUX4 MUX8

## 3.7 Registers

Below are the standard functions for managing 8 bit to 32 bit registers:

| | |
|---|---|
| SHL | shift left |
| SHR | shift right |
| ROL | rotation left |
| ROR | rotation right |

Below are advanced functions for register manipulation:

| | |
|---|---|
| MBShift | multibyte shift / rotate |

The following functions enable bit to bit operations on a 8 bit to 32 bit integers:

| | |
|---|---|
| AND_MASK | Boolean AND |
| OR_MASK | Boolean OR |
| XOR_MASK | exclusive OR |
| NOT_MASK | Boolean negation |

The following functions enable to pack/unpack 8, 16 and 32 bit registers

| | |
|---|---|
| LOBYTE | Get the lowest byte of a word |
| HIBYTE | Get the highest byte of a word |
| LOWORD | Get the lowest word of a double word |
| HIWORD | Get the highest word of a double word |
| MAKEWORD | Pack bytes to a word |
| MAKEDWORD | Pack words to a double word |
| PACK8 | Pack bits in a byte |
| UNPACK8 | Extract bits from a byte |

The following functions provide bit access in 8 bit to 32 bit integers:

| | |
|---|---|
| SETBIT | Set a bit in a register |
| TESTBIT | Test a bit of a register |

The following functions have been deprecated. They are available for backwards compatibility only. The functions listed above should be used for all current and future development.

| | |
|---|---|
| AND_WORD | AND_BYTE |
| OR_WORD | OR_BYTE |
| NOT_WORD | NOT_BYTE |
| XOR_WORD | XOR_BYTE |
| ROLW | RORW |
| ROLB | RORB |
| SHLW | SHRW |
| SHLB | SHRB |

### 3.7.1 AND_MASK

PLCopen

*Function* - Performs a bit to bit AND between two integer values

#### 3.7.1.1 Inputs

`IN : ANY` First input
`MSK : ANY` Second input (AND mask)

#### 3.7.1.2 Outputs

`Q : ANY` AND mask between IN and MSK inputs

#### 3.7.1.3 Remarks

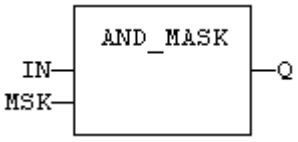
Arguments can be signed or unsigned integers from 8 to 32 bits.

In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the same value as the input rung. In IL language, the first parameter (IN) must be loaded in the current result before calling the function. The other input is the operands of the function.

#### 3.7.1.4 ST Language

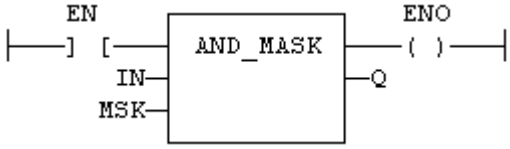
```
Q := AND_MASK (IN, MSK);
```

#### 3.7.1.5 FBD Language




#### 3.7.1.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO is equal to EN *)
```




#### 3.7.1.7 IL Language:

```
Op1: LD      IN
   AND_MASK MSK
   ST      Q
```

**See also**

OR_MASK XOR_MASK NOT_MASK

### 3.7.2 HIBYTE PLCopen

*Function* - Get the most significant byte of a word

#### 3.7.2.1 Inputs

`IN : UINT` 16 bit register

#### 3.7.2.2 Outputs

`Q : USINT` Most significant byte

#### 3.7.2.3 Remarks

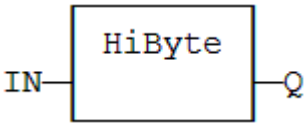
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.7.2.4 ST Language

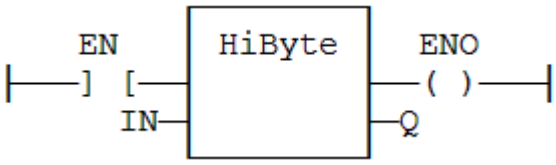
```
Q := HIBYTE (IN);
```

#### 3.7.2.5 FBD Language




#### 3.7.2.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.7.2.7 IL Language:

```
Op1: LD  IN
     HIBYTE
     ST  Q
```

**See also**

LOBYTE LOWORD HIWORD MAKEWORD MAKEDWORD

### 3.7.3 LOBYTE PLCopen

*Function* - Get the less significant byte of a word

#### 3.7.3.1 Inputs

`IN : UINT` 16 bit register

#### 3.7.3.2 Outputs

`Q : USINT` Lowest significant byte

#### 3.7.3.3 Remarks

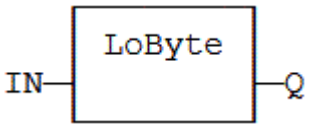
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.7.3.4 ST Language

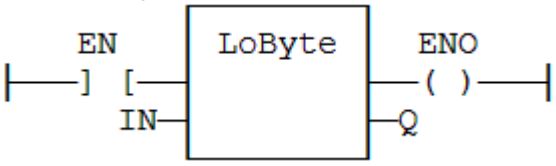
```
Q := LOBYTE (IN);
```

#### 3.7.3.5 FBD Language




#### 3.7.3.6 FFLD Language

(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)




#### 3.7.3.7 IL Language:

```
Op1: LD  IN
     LOBYTE
     ST  Q
```

**See also**

HIBYTE LOWORD HIWORD MAKEWORD MAKEDWORD

### 3.7.4 HIWORD PLCopen

*Function* - Get the most significant word of a double word

#### 3.7.4.1 Inputs

`IN : UDINT` 32 bit register

#### 3.7.4.2 Outputs

`Q : UINT` Most significant word

#### 3.7.4.3 Remarks

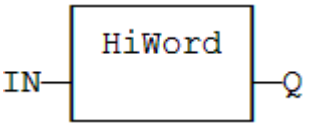
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.7.4.4 ST Language

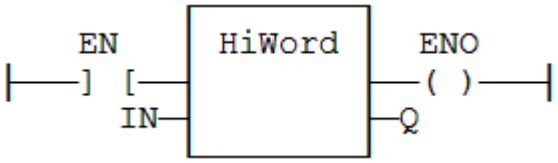
```
Q := HIWORD (IN);
```

#### 3.7.4.5 FBD Language




#### 3.7.4.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.7.4.7 IL Language:

```
Op1: LD  IN
     HIWORD
     ST  Q
```

See also

LOBYTE HIBYTE LOWORD MAKEWORD MAKEDWORD

## 3.7.5 LOWORD PLCopen

*Function* - Get the less significant word of a double word

### 3.7.5.1 Inputs

`IN : UDINT` 32 bit register

### 3.7.5.2 Outputs

`Q : UINT` Lowest significant word

### 3.7.5.3 Remarks

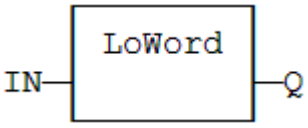
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

### 3.7.5.4 ST Language

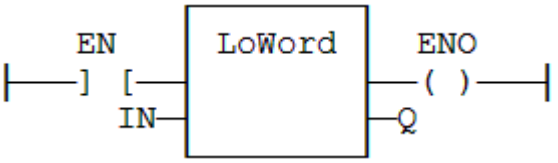
```
Q := LOWORD (IN);
```

### 3.7.5.5 FBD Language




### 3.7.5.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




### 3.7.5.7 IL Language:

```
Op1: LD  IN
     LOWORD
     ST  Q
```

**See also**

LOBYTE HIBYTE HIWORD MAKEWORD MAKEDWORD

## 3.7.6 MAKEDWORD PLCopen

*Function* - Builds a double word as the concatenation of two words

### 3.7.6.1 Inputs

`HI : USINT` Highest significant word
`LO : USINT` Lowest significant word

### 3.7.6.2 Outputs

`Q : UINT` 32 bit register

### 3.7.6.3 Remarks

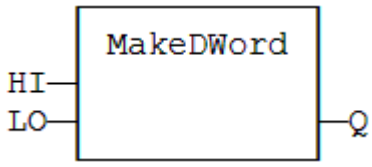
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the first input must be loaded in the current result before calling the function.

### 3.7.6.4 ST Language

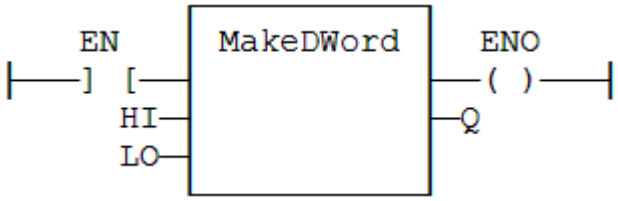
```
Q := MAKEDWORD (HI, LO);
```

### 3.7.6.5 FBD Language




### 3.7.6.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




### 3.7.6.7 IL Language:

```
Op1: LD      HI
     MAKEDWORD LO
     ST      Q
```

**See also**

LOBYTE HIBYTE LOWORD HIWORD MAKEWORD

## 3.7.7 MAKEWORD PLCopen

*Function* - Builds a word as the concatenation of two bytes

### 3.7.7.1 Inputs

`HI : USINT` Highest significant byte
`LO : USINT` Lowest significant byte

### 3.7.7.2 Outputs

`Q : UINT` 16 bit register

### 3.7.7.3 Remarks

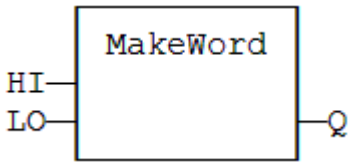
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the first input must be loaded in the current result before calling the function.

### 3.7.7.4 ST Language

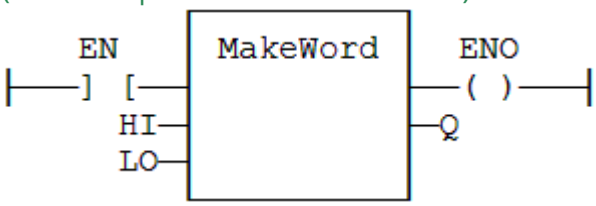
```
Q := MAKEWORD (HI, LO);
```

### 3.7.7.5 FBD Language




### 3.7.7.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




### 3.7.7.7 IL Language:

```
Op1: LD      HI
     MAKEWORD LO
     ST      Q
```

**See also**

LOBYTE HIBYTE LOWORD HIWORD MAKEDWORD

## 3.7.8 MBSHIFT PLCopen

*Function* - Multibyte shift / rotate

#### 3.7.8.1 Inputs

| | |
|---|---|
| `Buffer : SINT/USINT` | Array of bytes |
| `Pos : DINT` | Base position in the array |
| `NbByte : DINT` | Number of bytes to be shifted/rotated |
| `NbShift : DINT` | Number of shifts or rotations |
| `ToRight : BOOL` | TRUE for right / FALSE for left |
| `Rotate : BOOL` | TRUE for rotate / FALSE for shift |
| `InBit : BOOL` | Bit to be introduced in a shift |

#### 3.7.8.2 Outputs

| | |
|---|---|
| `Q : BOOL` | TRUE if successful |

#### 3.7.8.3 Remarks

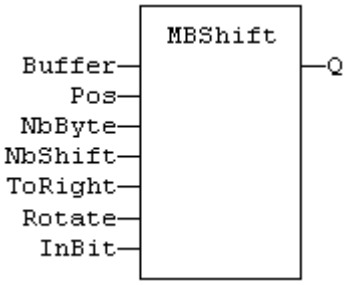
Use the "ToRight" argument to specify a shift to the left (FALSE) or to the right (TRUE). Use the "Rotate" argument to specify either a shift (FALSE) or a rotation (TRUE). In case of a shift, the "InBit" argument specifies the value of the bit that replaces the last shifted bit.

In FFLD language, the rung input (EN) validates the operation. The rung output is the result ("Q").

#### 3.7.8.4 ST Language

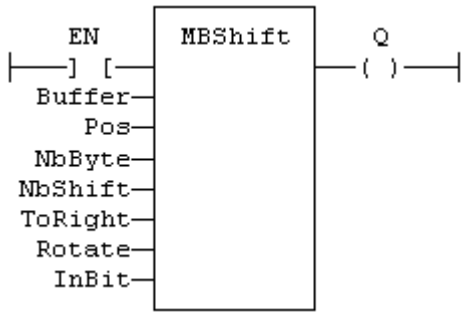
Q := MBShift (Buffer, Pos, NbByte, NbShift, ToRight, Rotate, InBit);

#### 3.7.8.5 FBD Language




#### 3.7.8.6 FFLD Language

(* the function is called only if EN is TRUE *)




#### 3.7.8.7 IL Language:

*Not available*

### 3.7.9 NOT_MASK PLCopen

*Function* - Performs a bit to bit negation of an integer value

#### 3.7.9.1 Inputs

`IN : ANY` Integer input

#### 3.7.9.2 Outputs

`Q : ANY` Bit to bit negation of the input

#### 3.7.9.3 Remarks

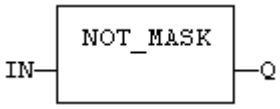
Arguments can be signed or unsigned integers from 8 to 32 bits.

In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the same value as the input rung. In IL language, the parameter (IN) must be loaded in the current result before calling the function.

#### 3.7.9.4 ST Language

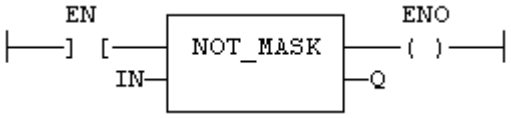
```
Q := NOT_MASK (IN);
```

#### 3.7.9.5 FBD Language




#### 3.7.9.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO is equal to EN *)
```




#### 3.7.9.7 IL Language:

```
Op1: LD    IN
     NOT_MASK
     ST    Q
```

**See also**

AND_MASK OR_MASK XOR_MASK

### 3.7.10 OR_MASK

PLCopen

*Function* - Performs a bit to bit OR between two integer values

#### 3.7.10.1 Inputs

`IN : ANY` First input
`MSK : ANY` Second input (OR mask)

#### 3.7.10.2 Outputs

`Q : ANY` OR mask between IN and MSK inputs

#### 3.7.10.3 Remarks

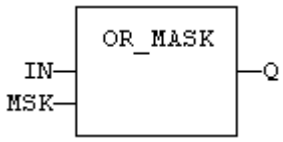
Arguments can be signed or unsigned integers from 8 to 32 bits.

In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the same value as the input rung. In IL language, the first parameter (IN) must be loaded in the current result before calling the function. The other input is the operands of the function.

#### 3.7.10.4 ST Language

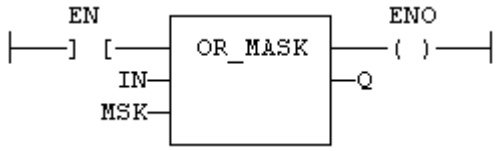
Q := OR_MASK (IN, MSK);

#### 3.7.10.5 FBD Language




#### 3.7.10.6 FFLD Language

(* The function is executed only if EN is TRUE *)
(* ENO is equal to EN *)




#### 3.7.10.7 IL Language:

```
Op1: LD    IN
     OR_MASK MSK
     ST    Q
```

See also

AND_MASK XOR_MASK NOT_MASK

### 3.7.11 PACK8 PLCopen

*Function* - Builds a byte with bits

#### 3.7.11.1 Inputs

`IN0 : BOOL` Less significant bit
`...`
`IN7 : BOOL` Most significant bit

#### 3.7.11.2 Outputs

`Q : USINT` Byte built with input bits

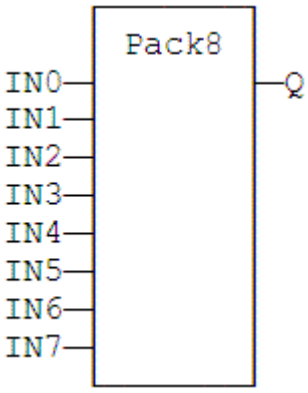
#### 3.7.11.3 Remarks

- In FFLD language, the input rung is the IN0 input. The output rung (ENO) keeps the same value as the input rung.
- In IL, the input must be loaded in the current result before calling the function.

#### 3.7.11.4 ST Language

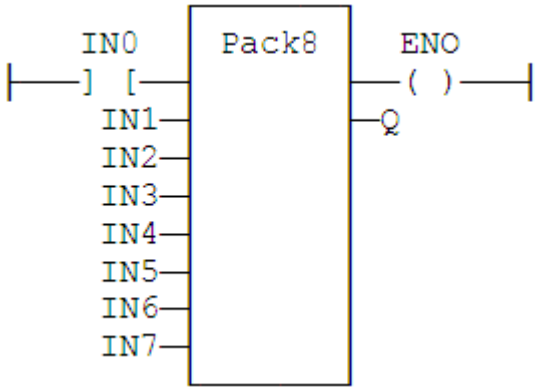
```
Q := PACK8 (IN0, IN1, IN2, IN3, IN4, IN5, IN6, IN7);
```

### 3.7.11.5 FBD Language



#### 3.7.11.6 FFLD Language

(* ENO keeps the same value as EN *)




#### 3.7.11.7 IL Language

```
Op1: LD    IN0
     PACK8 IN1, IN2, IN3, IN4, IN5, IN6, IN7
     ST    Q
```

See also

UNPACK8

### 3.7.12 ROL PLCopen

*Function* - Rotate bits of a register to the left.

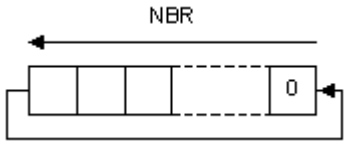
#### 3.7.12.1 Inputs

`IN : ANY` register
`NBR : DINT` Number of rotations (each rotation is 1 bit)

#### 3.7.12.2 Outputs

`Q : ANY` Rotated register

#### 3.7.12.3 Diagram




#### 3.7.12.4 Remarks

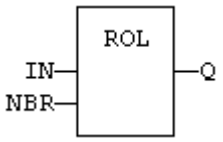
Arguments can be signed or unsigned integers from 8 to 32 bits.

In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the state of the input rung. In IL language, the first input must be loaded before the function call. The second input is the operand of the function.

#### 3.7.12.5 ST Language

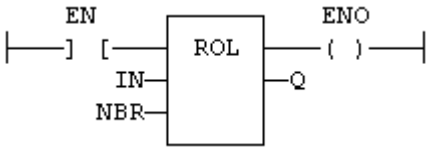
Q := ROL (IN, NBR);

#### 3.7.12.6 FBD Language




#### 3.7.12.7 FFLD Language

(* The rotation is executed only if EN is TRUE *)
(* ENO has the same value as EN *)




#### 3.7.12.8 IL Language:

```
Op1: LD  IN
     ROL NBR
     ST  Q
```

**See also**

SHL SHR ROR

### 3.7.13 ROR

*Function* - Rotate bits of a register to the right.

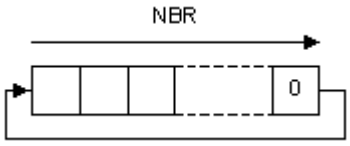
#### 3.7.13.1 Inputs

`IN : ANY` register
`NBR : ANY` Number of rotations (each rotation is 1 bit)

#### 3.7.13.2 Outputs

`Q : ANY` Rotated register

#### 3.7.13.3 Diagram




#### 3.7.13.4 Remarks

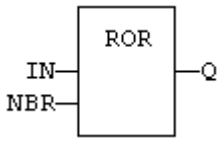
Arguments can be signed or unsigned integers from 8 to 32 bits.

In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the state of the input rung. In IL language, the first input must be loaded before the function call. The second input is the operand of the function.

#### 3.7.13.5 ST Language

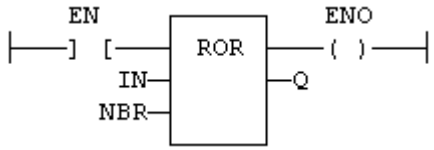
```
Q := ROR (IN, NBR);
```

#### 3.7.13.6 FBD Language




#### 3.7.13.7 FFLD Language

(* The rotation is executed only if EN is TRUE *)
(* ENO has the same value as EN *)




#### 3.7.13.8 IL Language:

```
Op1: LD  IN
     ROR NBR
     ST  Q
```

See also

SHL SHR ROL

### 3.7.14 RORb / ROR_SINT / ROR_USINT / ROR_BYTE

*Function* - Rotate bits of a register to the right.

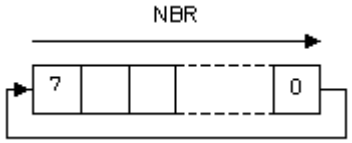
#### 3.7.14.1 Inputs

`IN : SINT` 8 bit register
`NBR : SINT` Number of rotations (each rotation is 1 bit)

#### 3.7.14.2 Outputs

`Q : SINT` Rotated register

#### 3.7.14.3 Diagram



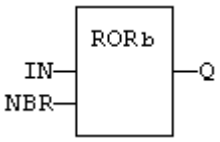

#### 3.7.14.4 Remarks

In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the state of the input rung. In IL language, the first input must be loaded before the function call. The second input is the operand of the function.

#### 3.7.14.5 ST Language

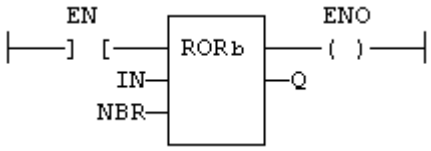
Q := RORb (IN, NBR);

#### 3.7.14.6 FBD Language




#### 3.7.14.7 FFLD Language

(* The rotation is executed only if EN is TRUE *)
(* ENO has the same value as EN *)




#### 3.7.14.8 IL Language:

```
Op1: FFLD   IN
     RORb NBR
     ST   Q
```

#### 3.7.14.9 See also

SHL SHR ROL ROR SHLb SHRb ROLb SHLw SHRw ROLw RORw

### 3.7.15 RORw / ROR_INT / ROR_UINT / ROR_WORD

*Function* - Rotate bits of a register to the right.

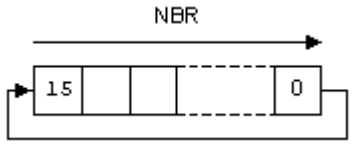
#### 3.7.15.1 Inputs

`IN : INT` 16 bit register
`NBR : INT` Number of rotations (each rotation is 1 bit)

#### 3.7.15.2 Outputs

`Q : INT` Rotated register

#### 3.7.15.3 Diagram



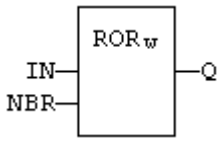

#### 3.7.15.4 Remarks

In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the state of the input rung. In IL language, the first input must be loaded before the function call. The second input is the operand of the function.

#### 3.7.15.5 ST Language

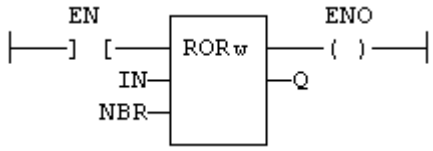
```
Q := RORw (IN, NBR);
```

#### 3.7.15.6 FBD Language




#### 3.7.15.7 FFLD Language

(* The rotation is executed only if EN is TRUE *)
(* ENO has the same value as EN *)




#### 3.7.15.8 IL Language:

```
Op1: FFLD  IN
   RORw NBR
   ST   Q
```

#### 3.7.15.9 See also

SHL SHR ROL ROR SHLb SHRb ROLb RORb SHLw SHRw ROLw

### 3.7.16 SETBIT PLCopen

*Function* - Set a bit in an integer register.

#### 3.7.16.1 Inputs

| | |
|---|---|
| `IN : ANY` | 8 to 64 bit integer register |
| `BIT : DINT` | Bit number (0 = less significant bit) |
| `VAL : BOOL` | Bit value to apply |

#### 3.7.16.2 Outputs

| | |
|---|---|
| `Q : ANY` | Modified register |

#### 3.7.16.3 Remarks

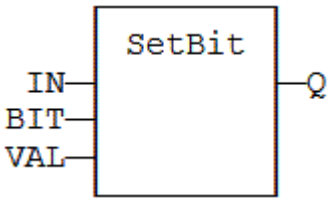
Types LINT, REAL, LREAL, TIME and STRING are not supported for IN and Q. IN and Q must have the same type. In case of invalid arguments (bad bit number or invalid input type) the function returns the value of IN without modification.

In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

#### 3.7.16.4 ST Language

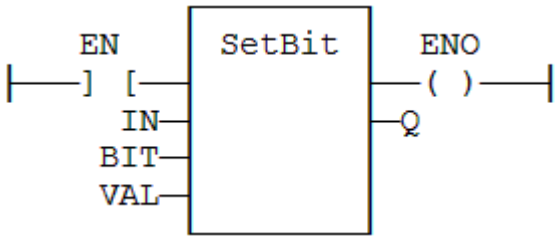
```
Q := SETBIT (IN, BIT, VAL);
```

#### 3.7.16.5 FBD Language




#### 3.7.16.6 FFLD Language

(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)




#### 3.7.16.7 IL Language

*Not available*

**See also**

TESTBIT

### 3.7.17 SHL

PLCopen

*Function* - Shift bits of a register to the left.

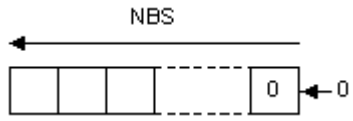
#### 3.7.17.1 Inputs

`IN : ANY` register
`NBS : ANY` Number of shifts (each shift is 1 bit)

#### 3.7.17.2 Outputs

`Q : ANY` Shifted register

#### 3.7.17.3 Diagram




#### 3.7.17.4 Remarks

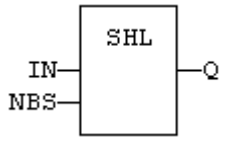
Arguments can be signed or unsigned integers from 8 to 32 bits.

In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the state of the input rung. In IL language, the first input must be loaded before the function call. The second input is the operand of the function.

#### 3.7.17.5 ST Language

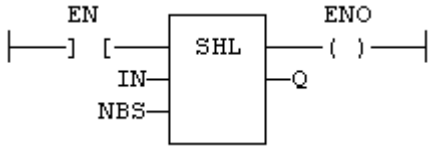
Q := SHL (IN, NBS);

#### 3.7.17.6 FBD Language




#### 3.7.17.7 FFLD Language

(* The shift is executed only if EN is TRUE *)
(* ENO has the same value as EN *)




#### 3.7.17.8 IL Language:

```
Op1: LD  IN
     SHL NBS
     ST  Q
```

**See also**

SHR ROL ROR

### 3.7.18 SHR PLCopen

*Function* - Shift bits of a register to the right.

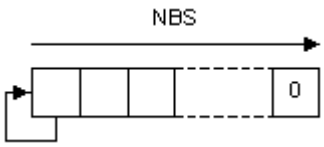
#### 3.7.18.1 Inputs

`IN : ANY` register
`NBS : ANY` Number of shifts (each shift is 1 bit)

#### 3.7.18.2 Outputs

`Q : ANY` Shifted register

#### 3.7.18.3 Diagram




#### 3.7.18.4 Remarks

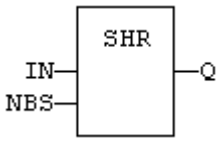
Arguments can be signed or unsigned integers from 8 to 32 bits.

In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the state of the input rung. In IL language, the first input must be loaded before the function call. The second input is the operand of the function.

#### 3.7.18.5 ST Language

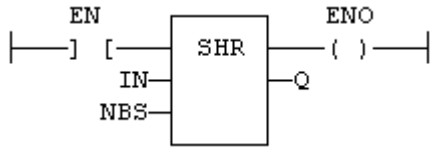
```
Q := SHR (IN, NBS);
```

#### 3.7.18.6 FBD Language




#### 3.7.18.7 FFLD Language

(* The shift is executed only if EN is TRUE *)
(* ENO has the same value as EN *)




#### 3.7.18.8 IL Language:

```
Op1: LD  IN
     SHR NBS
     ST  Q
```

**See also**

SHL ROL ROR

### 3.7.19 TESTBIT PLCopen

*Function* - Test a bit of an integer register.

#### 3.7.19.1 Inputs

`IN : ANY` 8 to 64 bit integer register
`BIT : DINT` Bit number (0 = less significant bit)

#### 3.7.19.2 Outputs

`Q : BOOL` Bit value

#### 3.7.19.3 Remarks

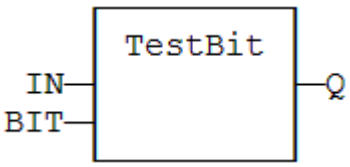
Types LINT, REAL, LREAL, TIME and STRING are not supported for IN and Q. IN and Q must have the same type. In case of invalid arguments (bad bit number or invalid input type) the function returns FALSE.

In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung is the output of the function.

#### 3.7.19.4 ST Language

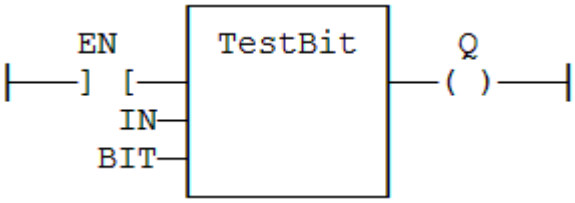
```
Q := TESTBIT (IN, BIT);
```

#### 3.7.19.5 FBD Language




#### 3.7.19.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
```




#### 3.7.19.7 IL Language

*Not available*

**See also**

SETBIT

### 3.7.20 UNPACK8 PLCopen

*Function block* - Extract bits of a byte

#### 3.7.20.1 Inputs

`IN : USINT` 8 bit register

#### 3.7.20.2 Outputs

`Q0 : BOOL` Less significant bit
`...`
`Q7 : BOOL` Most significant bit

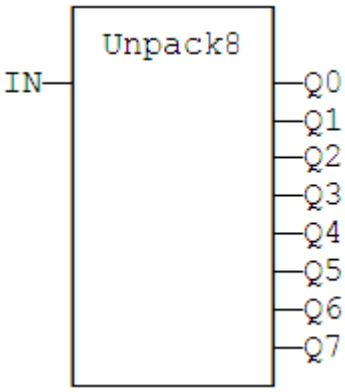
#### 3.7.20.3 Remarks

In FFLD language, the output rung is the Q0 output. The operation is executed only in the input rung (EN) is TRUE.

#### 3.7.20.4 ST Language

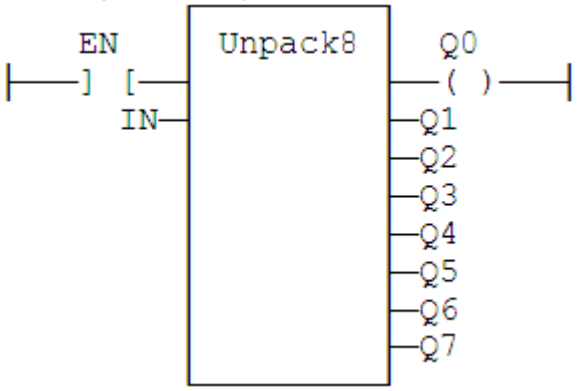
```
(* MyUnpack is a declared instance of the UNPACK8 function block *)
MyUnpack (IN);
Q0 := MyUnpack.Q0;
Q1 := MyUnpack.Q1;
Q2 := MyUnpack.Q2;
Q3 := MyUnpack.Q3;
Q4 := MyUnpack.Q4;
Q5 := MyUnpack.Q5;
Q6 := MyUnpack.Q6;
Q7 := MyUnpack.Q7;
```

#### 3.7.20.5 FBD Language




#### 3.7.20.6 FFLD Language

(* The operation is performed if EN = TRUE *)




#### 3.7.20.7 IL Language:

```
(* MyUnpack is a declared instance of the UNPACK8 function block *)
Op1: CAL MyUnpack (IN)
    FFLD  MyUnpack.Q0
    ST  Q0
    (* ... *)
    FFLD  MyUnpack.Q7
    ST  Q7
```

**See also**

PACK8

### 3.7.21 XOR_MASK PLCopen

*Function* - Performs a bit to bit exclusive OR between two integer values

#### 3.7.21.1 Inputs

`IN` : `ANY` First input
`MSK` : `ANY` Second input (XOR mask)

#### 3.7.21.2 Outputs

`Q` : `ANY` Exclusive OR mask between IN and MSK inputs

#### 3.7.21.3 Remarks

Arguments can be signed or unsigned integers from 8 to 32 bits.

In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the same value as the input rung. In IL language, the first parameter (IN) must be loaded in the current result before calling the function. The other input is the operands of the function.

### 3.7.21.4 ST Language

```
Q := XOR_MASK (IN, MSK);
```

### 3.7.21.5 FBD Language


### 3.7.21.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO is equal to EN *)
```


### 3.7.21.7 IL Language:

```
Op1: LD      IN
   XOR_MASK MSK
   ST      Q
```

**See also**

AND_MASK OR_MASK NOT_MASK

## 3.8 Counters

Below are the standard blocks for managing counters:

| | |
|---|---|
| CTU | Up counter |
| CTD | Down Counter |
| CTUD | Up / Down Counter |

### 3.8.1 CTD / CTDr PLCopen

*Function Block* - Down counter.

#### 3.8.1.1 Inputs

`CD : BOOL` Enable counting. Counter is decreased on each call when CD is TRUE
`LOAD : BOOL` Re-load command. Counter is set to PV when called with LOAD to TRUE
`PV : DINT` Programmed maximum value

#### 3.8.1.2 Outputs

`Q : BOOL` TRUE when counter is empty, i.e. when CV = 0
`CV : DINT` Current value of the counter

#### 3.8.1.3 Remarks

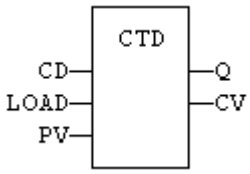
The counter is empty (CV = 0) when the application starts. The counter does not include a pulse detection for CD input. Use R_TRIG or F_TRIG function block for counting pulses of CD input signal. In FFLD language, CD is the input rung. The output rung is the Q output.

CTUr, CTDr, CTUDr function blocks operate exactly as other counters, except that all Boolean inputs (CU, CD, RESET, LOAD) have an implicit rising edge detection included.

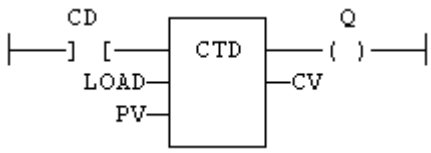
#### 3.8.1.4 ST Language

```
(* MyCounter is a declared instance of CTD function block *)
MyCounter (CD, LOAD, PV);
Q := MyCounter.Q;
CV := MyCounter.CV;
```

#### 3.8.1.5 FBD Language




#### 3.8.1.6 FFLD Language




#### 3.8.1.7 IL Language:

```
(* MyCounter is a declared instance of CTD function block *)
Op1: CAL    MyCounter (CD, LOAD, PV)
FFLD    MyCounter.Q
ST    Q
FFLD    MyCounter.CV
ST    CV
```

**See also**

CTU CTUD

## 3.8.2 CTU / CTUr PLCopen

*Function Block* - Up counter.

### 3.8.2.1 Inputs

`CU : BOOL` Enable counting. Counter is increased on each call when CU is TRUE
`RESET : BOOL` Reset command. Counter is reset to 0 when called with RESET to TRUE
`PV : DINT` Programmed maximum value

### 3.8.2.2 Outputs

`Q : BOOL` TRUE when counter is full, i.e. when CV = PV
`CV : DINT` Current value of the counter

### 3.8.2.3 Remarks

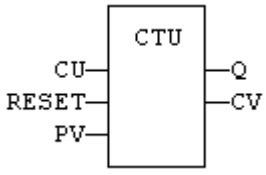
The counter is empty (CV = 0) when the application starts. The counter does not include a pulse detection for CU input. Use R_TRIG or F_TRIG function block for counting pulses of CU input signal. In FFLD language, CU is the input rung. The output rung is the Q output.

CTUr, CTDr, CTUDr function blocks operate exactly as other counters, except that all Boolean inputs (CU, CD, RESET, LOAD) have an implicit rising edge detection included.

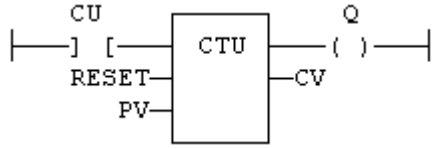
### 3.8.2.4 ST Language

```
(* MyCounter is a declared instance of CTU function block *)
MyCounter (CU, RESET, PV);
Q := MyCounter.Q;
CV := MyCounter.CV;
```

### 3.8.2.5 FBD Language




### 3.8.2.6 FFLD Language




### 3.8.2.7 IL Language:

```
(* MyCounter is a declared instance of CTU function block *)
Op1: CAL   MyCounter (CU, RESET, PV)
FFLD    MyCounter.Q
ST    Q
FFLD    MyCounter.CV
ST    CV
```

**See also**

CTD CTUD

## 3.8.3 CTUD / CTUDr PLCopen

*Function Block* - Up/down counter.

### 3.8.3.1 Inputs

| | |
|---|---|
| `CU : BOOL` | Enable counting. Counter is increased on each call when CU is TRUE |
| `CD : BOOL` | Enable counting. Counter is decreased on each call when CD is TRUE |
| `RESET : BOOL` | Reset command. Counter is reset to 0 called with RESET to TRUE |
| `LOAD : BOOL` | Re-load command. Counter is set to PV when called with LOAD to TRUE |
| `PV : DINT` | Programmed maximum value |

### 3.8.3.2 Outputs

| | |
|---|---|
| `QU : BOOL` | TRUE when counter is full, i.e. when CV = PV |
| `QD : BOOL` | TRUE when counter is empty, i.e. when CV = 0 |
| `CV : DINT` | Current value of the counter |

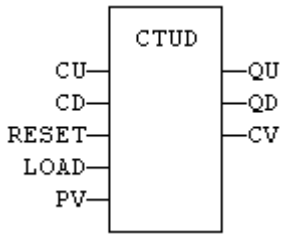
### 3.8.3.3 Remarks

- The counter is empty (CV = 0) when the application starts. The counter does not include a pulse detection for CU and CD inputs. Use R_TRIG or F_TRIG function blocks for counting pulses of CU or CD input signals. In FFLD language, CU is the input rung. The output rung is the QU output.
- CTUr, CTDr, CTUDr function blocks operate exactly as other counters, except that all Boolean inputs (CU, CD, RESET, LOAD) have an implicit rising edge detection included.

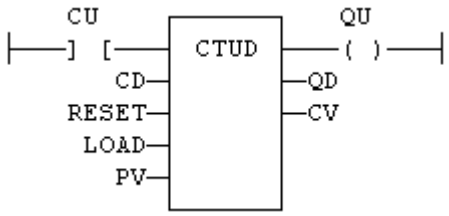
### 3.8.3.4 ST Language

```
(* MyCounter is a declared instance of CTUD function block *)
MyCounter (CU, CD, RESET, LOAD, PV);
QU := MyCounter.QU;
QD := MyCounter.QD;
CV := MyCounter.CV;
```

### 3.8.3.5 FBD Language



#### 3.8.3.6 FFLD Language




#### 3.8.3.7 IL Language:

```
(* MyCounter is a declared instance of CTUD function block *)
Op1: CAL    MyCounter (CU, CD, RESET, LOAD, PV)
FFLD     MyCounter.QU
ST     QU
FFLD     MyCounter.QD
ST     QD
FFLD     MyCounter.CV
ST     CV
```

**See also**

CTU CTD

## 3.9 Timers

Below are the standard functions for managing timers:

| | |
|---|---|
| TON | On timer |
| TOF | Off timer |
| TP | Pulse timer |
| BLINK | Blinker |
| BLINKA | Asymmetric blinker |
| PLS | Pulse signal generator |
| TMU | Up-counting stop watch |
| TMUsec | Up-counting stop watch (seconds) |
| TMD | Down-counting stop watch |

### 3.9.1 BLINK PLCopen

*Function Block* - Blinker.

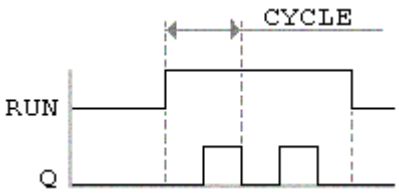
#### 3.9.1.1 Inputs

`RUN : BOOL` Enabling command
`CYCLE : TIME` Blinking period

#### 3.9.1.2 Outputs

`Q : BOOL` Output blinking signal

#### 3.9.1.3 Time diagram



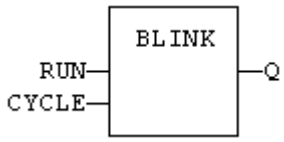

#### 3.9.1.4 Remarks

The output signal is FALSE when the RUN input is FALSE. The CYCLE input is the complete period of the blinking signal. In FFLD language, the input rung is the IN command. The output rung is the Q output signal.

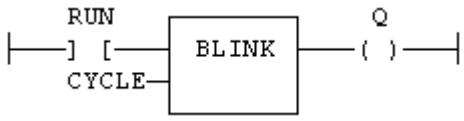
#### 3.9.1.5 ST Language

```
(* MyBlinker is a declared instance of BLINK function block *)
MyBlinker (RUN, CYCLE);
Q := MyBlinker.Q;
```

#### 3.9.1.6 FBD Language



#### 3.9.1.7 FFLD Language




#### 3.9.1.8 IL Language

```
(* MyBlinker is a declared instance of BLINK function block *)
Op1: CAL MyBlinker (RUN, CYCLE)
   FFLD  MyBlinker.Q
   ST  Q
```

**See also**

TON TOF TP

### 3.9.2 BLINKA PLCopen

*Function Block* - Asymmetric blinker.

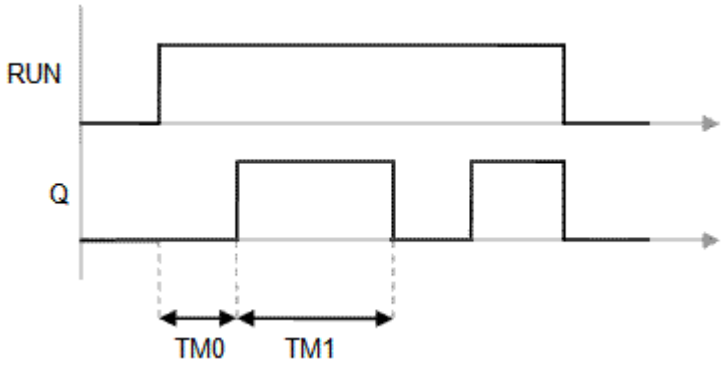
#### 3.9.2.1 Inputs

| | |
|---|---|
| `RUN : BOOL` | Enabling command |
| `TM0 : TIME` | Duration of FALSE state on output |
| `TM1 : TIME` | Duration of TRUE state on output |

#### 3.9.2.2 Outputs

| | |
|---|---|
| `Q : BOOL` | Output blinking signal |

#### 3.9.2.3 Time diagram



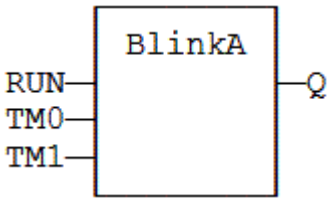

#### 3.9.2.4 Remarks

The output signal is FALSE when the RUN input is FALSE. In FFLD language, the input rung is the IN command. The output rung is the Q output signal.

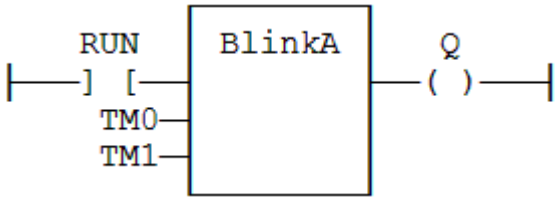
#### 3.9.2.5 ST Language

```
(* MyBlinker is a declared instance of BLINKA function block *)
MyBlinker (RUN, TM0, TM1);
Q := MyBlinker.Q;
```

#### 3.9.2.6 FBD Language




#### 3.9.2.7 FFLD Language




#### 3.9.2.8 IL Language:

```
(* MyBlinker is a declared instance of BLINKA function block *)
Op1: CAL MyBlinker (RUN, TM0, TM1)
    FFLD  MyBlinker.Q
    ST  Q
```

**See also**

TON TOF TP

### 3.9.3 PLS PLCopen

*Function Block* - Pulse signal generator

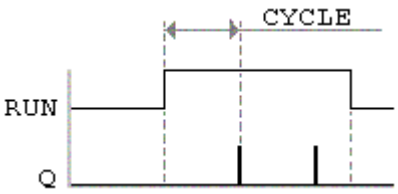
#### 3.9.3.1 Inputs

| | |
|---|---|
| `RUN : BOOL` | Enabling command |
| `CYCLE : TIME` | Signal period |

#### 3.9.3.2 Outputs

| | |
|---|---|
| `Q : BOOL` | Output pulse signal |

#### 3.9.3.3 Time diagram



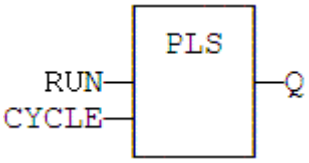

#### 3.9.3.4 Remarks

On every period, the output is set to TRUE during one cycle only. In FFLD language, the input rung is the IN command. The output rung is the Q output signal.

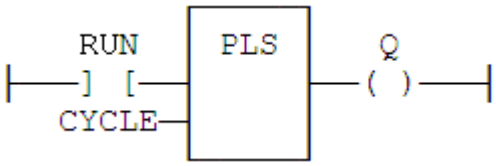
#### 3.9.3.5 ST Language

```
(* MyPLS is a declared instance of PLS function block *)
MyPLS (RUN, CYCLE);
Q := MyPLS.Q;
```

#### 3.9.3.6 FBD Language




#### 3.9.3.7 FFLD Language



#### 3.9.3.8 IL Language

```
(* MyPLS is a declared instance of PLS function block *)
Op1: CAL MyPLS (RUN, CYCLE)
     FFLD  MyPLS.Q
     ST  Q
```

**See also**

TON TOF TP

### 3.9.4 Sig_Gen PLCopen

*Function Block* - Generator of pseudo-analogical Signal

#### 3.9.4.1 Inputs

`RUN : BOOL` Enabling command

`PERIOD : TIME` Signal period

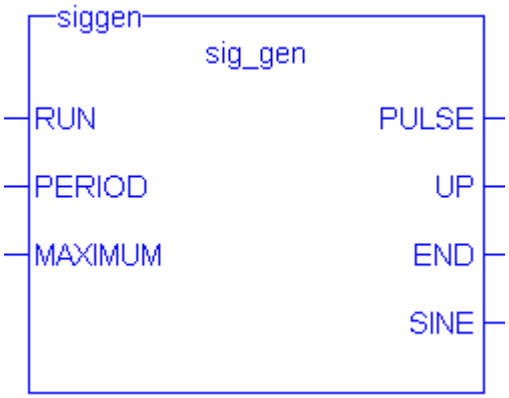
`MAXIMUM : DINT` Maximum growth during the signal period

#### 3.9.4.2 Outputs

This FB generates signals of the four following types:

- `PULSE:` blinking at each period
- `UP :` growing according max * period
- `END :` pulse after max * period
- `SINE :` sine curve

#### 3.9.4.3 FFLD Language




### 3.9.5 TMD PLCopen

*Function Block* - Down-counting stop watch.

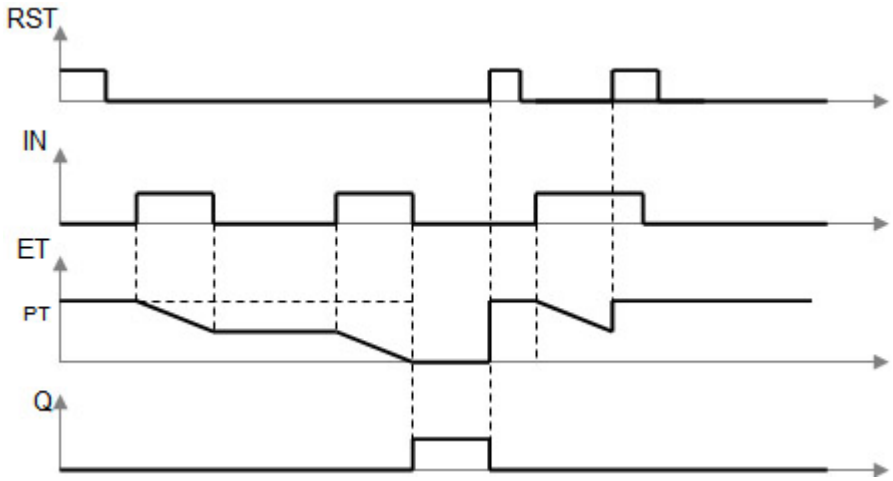
#### 3.9.5.1 Inputs

`IN : BOOL` The time counts when this input is TRUE
`RST : BOOL` Timer is reset to PT when this input is TRUE
`PT : TIME` Programmed time

#### 3.9.5.2 Outputs

`Q : BOOL` Timer elapsed output signal
`ET : TIME` Elapsed time

#### 3.9.5.3 Time diagram



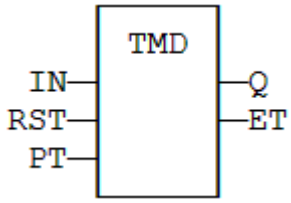

#### 3.9.5.4 Remarks

The timer counts up when the IN input is TRUE. It stops when the programmed time is elapsed. The timer is reset when the RST input is TRUE. It is not reset when IN is false.

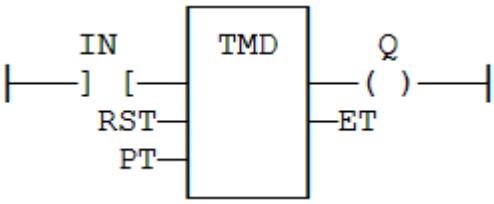
#### 3.9.5.5 ST Language

```
(* MyTimer is a declared instance of TMD function block *)
MyTimer (IN, RST, PT);
Q := MyTimer.Q;
ET := MyTimer.ET;
```

#### 3.9.5.6 FBD Language




#### 3.9.5.7 FFLD Language




#### 3.9.5.8 IL Language

```
(* MyTimer is a declared instance of TMD function block *)
Op1: CAL MyTimer (IN, RST, PT)
 FFLD: MyTimer.Q
 ST: Q
```

FFLD: MyTimer.ET
ST: ET

**See also**

TMU

## 3.9.6 TMU / TMUsec

*Function Block* - Up-counting stop watch. TMUsec is identical to TMU except that the parameter is a number of seconds.

### 3.9.6.1 Inputs

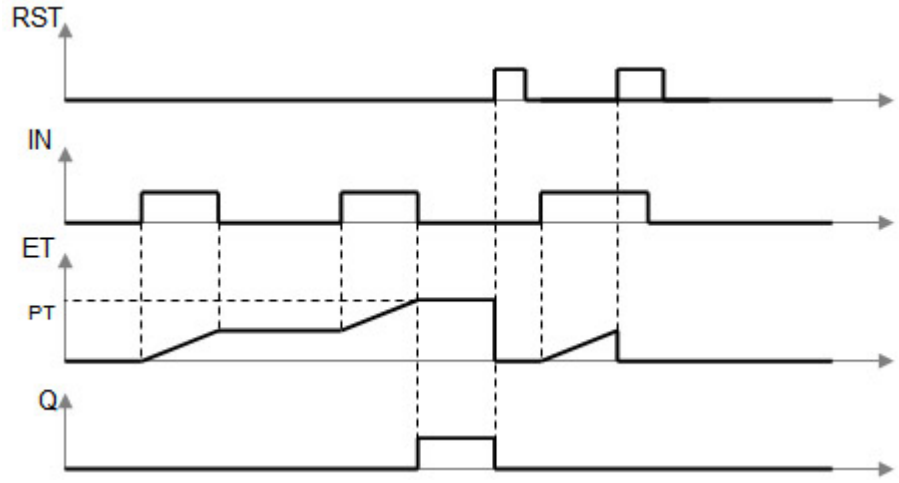
| | | |
|---|---|---|
| **IN** | BOOL | The time counts when this input is TRUE |
| **RST** | BOOL | Timer is reset to 0 when this input is TRUE |
| **PT** | TIME | Programmed time |
| **PTsec** | UDINT | Programmed time. (TMUsec - seconds) |

### 3.9.6.2 Outputs

`Q : BOOL` Timer elapsed output signal
`ET : TIME` Elapsed time

| | | |
|---|---|---|
| **Q** | BOOL | Timer elapsed output signal |
| **ET** | TIME | Elapsed time |
| **ETsec** | UDINT | Elapsed time. (TMU - seconds) |

### 3.9.6.3 Time diagram




### 3.9.6.4 Remarks

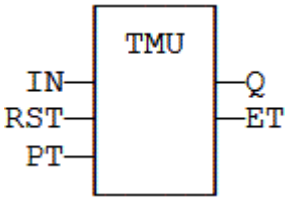
The timer counts up when the IN input is TRUE. It stops when the programmed time is elapsed. The timer is reset when the RST input is TRUE. It is not reset when IN is false.

### 3.9.6.5 ST Language

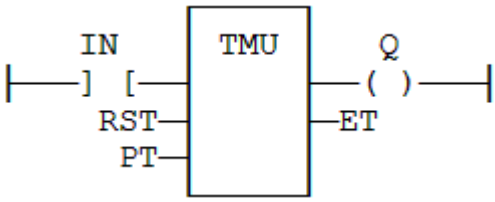
```
(* MyTimer is a declared instance of TMU function block *)
```

```
MyTimer (IN, RST, PT);
Q := MyTimer.Q;
ET := MyTimer.ET;
```

#### 3.9.6.6 FBD Language




#### 3.9.6.7 FFLD Language




#### 3.9.6.8 IL Language:

(* MyTimer is a declared instance of TMU function block *)

```
Op1: CAL   MyTimer (IN, RST, PT)
     FFLD  MyTimer.Q
     ST    Q
     FFLD  MyTimer.ET
     ST    ET
```

See also

TMD

### 3.9.7 TOF / TOFR PLCopen

*Function Block* - Off timer.

#### 3.9.7.1 Inputs

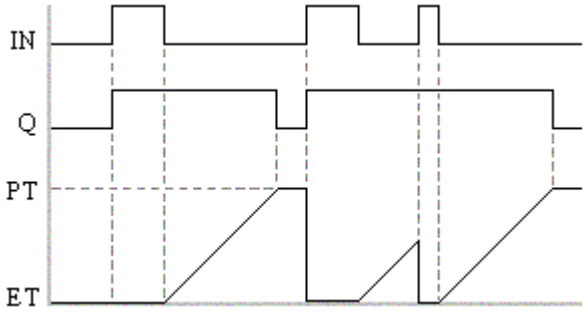
`IN : BOOL` Timer command
`PT : TIME` Programmed time

`RST : BOOL` Reset (TOFR only)

#### 3.9.7.2 Outputs

`Q : BOOL` Timer elapsed output signal
`ET : TIME` Elapsed time

#### 3.9.7.3 Time diagram




#### 3.9.7.4 Remarks

The timer starts on a falling pulse of IN input. It stops when the elapsed time is equal to the programmed time. A rising pulse of IN input resets the timer to 0. The output signal is set to TRUE on when the IN input rises to TRUE, reset to FALSE when programmed time is elapsed..

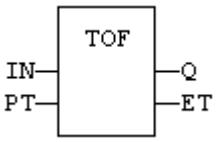
TOFR is same as TOF but has an extra input for resetting the timer

In FFLD language, the input rung is the IN command. The output rung is Q the output signal.

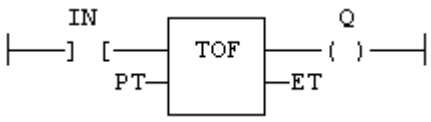
#### 3.9.7.5 ST Language

```
(* MyTimer is a declared instance of TOF function block *)
MyTimer (IN, PT);
Q := MyTimer.Q;
ET := MyTimer.ET;
```

#### 3.9.7.6 FBD Language




#### 3.9.7.7 FFLD Language




#### 3.9.7.8 IL Language:

```
(* MyTimer is a declared instance of TOF function block *)
Op1: CAL MyTimer (IN, PT)
    FFLD  MyTimer.Q
    ST  Q
    FFLD  MyTimer.ET
    ST  ET
```

See also

TON TP BLINK

### 3.9.8 TON PLCopen

*Function Block* - On timer.

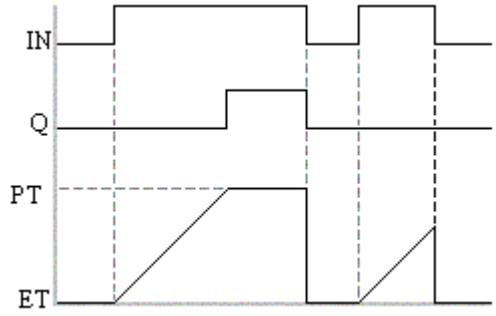
#### 3.9.8.1 Inputs

`IN : BOOL` Timer command
`PT : TIME` Programmed time

#### 3.9.8.2 Outputs

`Q : BOOL` Timer elapsed output signal
`ET : TIME` Elapsed time

#### 3.9.8.3 Time diagram



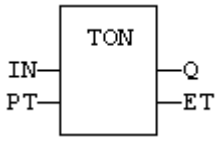

#### 3.9.8.4 Remarks

- The timer starts on a rising pulse of IN input. It stops when the elapsed time is equal to the programmed time. A falling pulse of IN input resets the timer to 0. The output signal is set to TRUE when programmed time is elapsed, and reset to FALSE when the input command falls.
- In FFLD language, the input rung is the IN command. The output rung is Q the output signal.

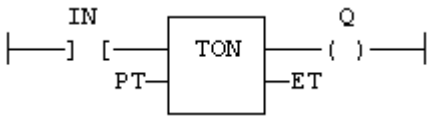
#### 3.9.8.5 ST Language

```
(* Inst_TON is a declared instance of TON function block *)
Inst_TON( FALSE, T#2s );
Q := Inst_TON.Q;
ET := Inst_TON.ET;
```

#### 3.9.8.6 FBD Language



#### 3.9.8.7 FFLD Language




#### 3.9.8.8 IL Language:

```
(* MyTimer is a declared instance of TON function block *)
Op1: CAL MyTimer (IN, PT)
     FFLD  MyTimer.Q
     ST  Q
     FFLD  MyTimer.ET
     ST  ET
```

See also

TOF TP BLINK

### 3.9.9 TP / TPR PLCopen

*Function Block* - Pulse timer.

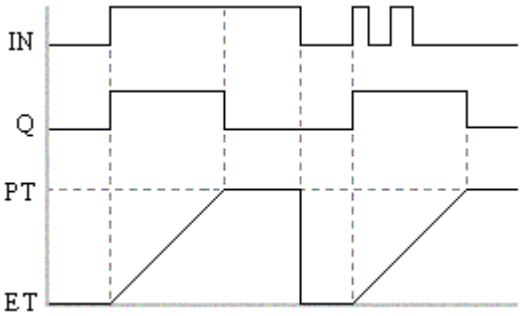
#### 3.9.9.1 Inputs

`IN : BOOL` Timer command
`PT : TIME` Programmed time
`RST : BOOL` Reset (TPR only)

#### 3.9.9.2 Outputs

`Q : BOOL` Timer elapsed output signal
`ET : TIME` Elapsed time

#### 3.9.9.3 Time diagram




#### 3.9.9.4 Remarks

The timer starts on a rising pulse of IN input. It stops when the elapsed time is equal to the programmed time. A falling pulse of IN input resets the timer to 0, only if the programmed time is elapsed. All pulses of IN while the timer is running are ignored. The output signal is set to TRUE while the timer is running.

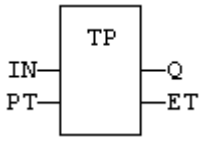
TPR is same as TP but has an extra input for resetting the timer

In FFLD language, the input rung is the IN command. The output rung is Q the output signal.

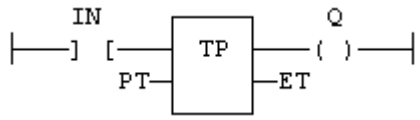
#### 3.9.9.5 ST Language

```
(* MyTimer is a declared instance of TP function block *)
MyTimer (IN, PT);
Q := MyTimer.Q;
ET := MyTimer.ET;
```

#### 3.9.9.6 FBD Language




#### 3.9.9.7 FFLD Language




#### 3.9.9.8 IL Language:

```
(* MyTimer is a declared instance of TP function block *)
Op1: CAL MyTimer (IN, PT)
    FFLD  MyTimer.Q
    ST  Q
    FFLD  MyTimer.ET
    ST  ET
```

**See also**

TON TOF BLINK

## 3.10 Mathematic operations PLCopen

Below are the standard functions that perform mathematic calculation:

| | |
|---|---|
| ABS | absolute value |
| TRUNC | integer part |
| LOG, LN / LNL | logarithm, natural logarithm |
| POW,EXPT, EXP / EXPL | power |
| SQRT, ROOT | square root, root extraction |
| SCALELIN | scaling - linear conversion |

### 3.10.1 ABS / ABSL PLCopen

*Function* - Returns the absolute value of the input.

#### 3.10.1.1 Inputs

`IN : REAL/LREAL` ANY value

#### 3.10.1.2 Outputs

`Q : REAL/LREAL` Result: absolute value of IN

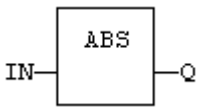
#### 3.10.1.3 Remarks

In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung. In IL, the input must be loaded in the current result before calling the function.

#### 3.10.1.4 ST Language

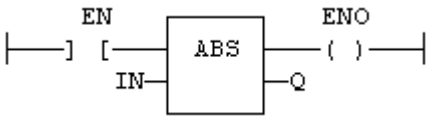
```
Q := ABS (IN);
```

#### 3.10.1.5 FBD Language




#### 3.10.1.6 FFLD Language

The function is executed only if EN is TRUE.
ENO keeps the same value as EN.




#### 3.10.1.7 IL Language

```
Op1: LD  IN
     ABS
     ST  Q   (* Q is: ABS (IN) *)
```

See also

TRUNC LOG POW SQRT

## 3.10.2 EXPT PLCopen

*Function* - Calculates a power.

### 3.10.2.1 Inputs

`IN : REAL` Real value
`EXP : DINT` Exponent

### 3.10.2.2 Outputs

`Q : REAL` Result: IN at the 'EXP' power

### 3.10.2.3 Remarks

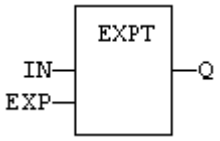
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function. The exponent (second input of the function) must be the operand of the function.

### 3.10.2.4 ST Language

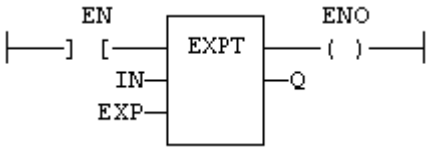
```
Q := EXPT (IN, EXP);
```

### 3.10.2.5 FBD Language




### 3.10.2.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




### 3.10.2.7 IL Language:

```
Op1: LD   IN
     EXPT EXP
     ST   Q    (* Q is: (IN ** EXP) *)
```

**See also**

ABS TRUNC LOG SQRT

## 3.10.3 EXP / EXPL PLCopen

*Function* - Calculates the natural exponential of the input.

### 3.10.3.1 Inputs

`IN : REAL/LREAL` Real value

#### 3.10.3.2 Outputs

`Q : REAL/LREAL` Result: natural exponential of IN.

#### 3.10.3.3 Remarks

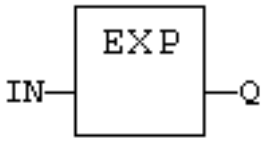
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.10.3.4 ST Language

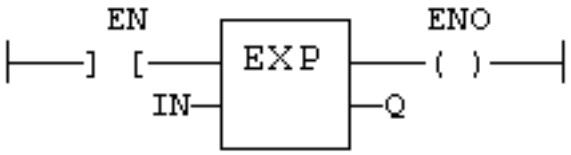
```
Q := EXP (IN);
```

#### 3.10.3.5 FBD Language




#### 3.10.3.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.10.3.7 IL Language:

```
Op1: LD  IN
     EXP
     ST    Q     (* Q is: EXP (IN) *)
```

### 3.10.4 LOG / LOGL PLCopen

*Function* - Calculates the logarithm (base 10) of the input.

#### 3.10.4.1 Inputs

`IN : REAL/LREAL` Real value

#### 3.10.4.2 Outputs

`Q : REAL/LREAL` Result: logarithm (base 10) of IN

#### 3.10.4.3 Remarks

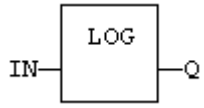
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.10.4.4 ST Language

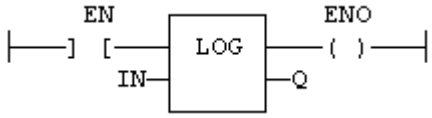
```
Q := LOG (IN);
```

#### 3.10.4.5 FBD Language




#### 3.10.4.6 FFLD Language

(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)




#### 3.10.4.7 IL Language:

```
Op1: LD  IN
     LOG
     ST  Q   (* Q is: LOG (IN) *)
```

**See also**

ABS TRUNC POW SQRT

### 3.10.5 LN / LNL PLCopen

*Function* - Calculates the natural logarithm of the input.

#### 3.10.5.1 Inputs

| | |
|---|---|
| `IN : REAL/LREAL` | Real value |

#### 3.10.5.2 Outputs

| | |
|---|---|
| `Q : REAL/LREAL` | Result: natural logarithm of IN |

#### 3.10.5.3 Remarks

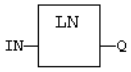
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.10.5.4 ST Language

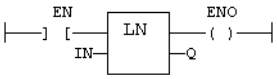
```
Q := LN (IN);
```

#### 3.10.5.5 FBD Language




#### 3.10.5.6 FFLD Language

(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)




#### 3.10.5.7 IL Language:

```
Op1: LD  IN
     LN
     ST    Q     (* Q is: LN (IN) *)
```

### 3.10.6 POW ** POWL PLCopen

*Function* - Calculates a power.

#### 3.10.6.1 Inputs

| | |
|---|---|
| `IN : REAL/LREAL` | Real value |
| `EXP : REAL/LREAL` | Exponent |

#### 3.10.6.2 Outputs

| | |
|---|---|
| `Q : REAL/LREAL` | Result: IN at the 'EXP' power |

#### 3.10.6.3 Remarks

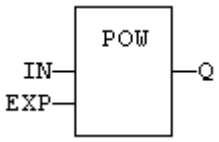
Alternatively, in ST language, the "**" operator can be used. In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function. The exponent (second input of the function) must be the operand of the function.

#### 3.10.6.4 ST Language

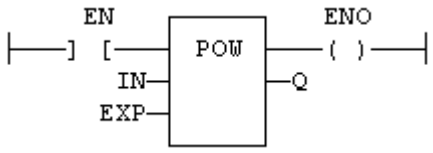
Q := POW (IN, EXP);
Q := IN ** EXP;

#### 3.10.6.5 FBD Language




#### 3.10.6.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.10.6.7 IL Language:

```
Op1: LD  IN
     POW EXP
     ST  Q   (* Q is: (IN ** EXP) *)
```

**See also**

ABS TRUNC LOG SQRT

### 3.10.7 ROOT PLCopen

*Function* - Calculates the Nth root of the input.

#### 3.10.7.1 Inputs

`IN : REAL` Real value
`N : DINT` Root level

#### 3.10.7.2 Outputs

`Q : REAL` Result: Nth root of IN

#### 3.10.7.3 Remarks

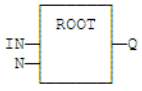
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.10.7.4 ST Language

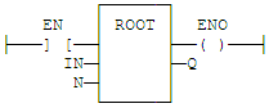
```
Q := ROOT (IN, N);
```

#### 3.10.7.5 FBD Language




#### 3.10.7.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.10.7.7 IL Language:

```
Op1: LD   IN
     ROOT  N
     ST    Q     (* Q is: ROOT (IN) *)
```

## 3.10.8 ScaleLin

PLCopen

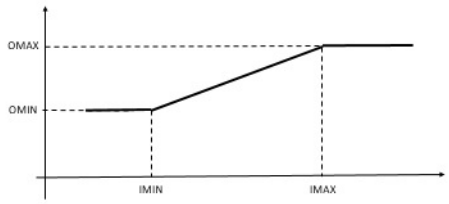
*Function* - Scaling - linear conversion.

### 3.10.8.1 Inputs

| | | |
|---|---|---|
| IN | : REAL | Real value |
| IMIN | : REAL | Minimum input value |
| IMAX | : REAL | Maximum input value |
| OMIN | : REAL | Minimum output value |
| OMAX | : REAL | Maximum output value |

### 3.10.8.2 Outputs

OUT : REAL Result: OMIN + IN * (OMAX - OMIN) / (IMAX - IMIN)




### 3.10.8.3 Truth table

| Inputs | OUT |
|---|---|
| IMIN >=IMAX | = IN |
| IN < IMIN | = IMIN |
| IN >IMAX | = IMAX |
| other | = OMIN + IN * (OMAX - OMIN) / (IMAX - IMIN) |

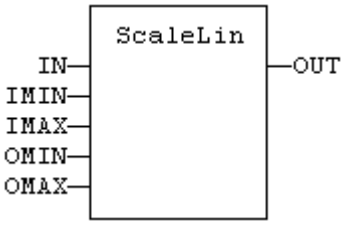
### 3.10.8.4 Remarks

- In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.
- In IL, the input must be loaded in the current result before calling the function.

### 3.10.8.5 ST Language

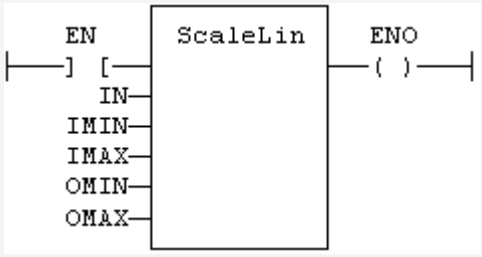
```
OUT := ScaleLin (IN, IMIN, IMAX, OMIN, OMAX);
```

### 3.10.8.6 FBD Language



#### 3.10.8.7 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.10.8.8 IL Language

```
Op1: LD      IN
     ScaleLin IMAX, IMIN, OMAX, OMIN
     ST      OUT
```

### 3.10.9 SQRT / SQRTL PLCopen

*Function* - Calculates the square root of the input.

#### 3.10.9.1 Inputs

`IN : REAL/LREAL` Real value

#### 3.10.9.2 Outputs

`Q : REAL/LREAL` Result: square root of IN

#### 3.10.9.3 Remarks

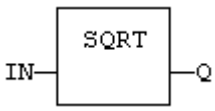
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.10.9.4 ST Language

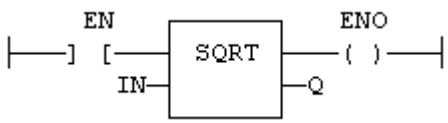
```
Q := SQRT (IN);
```

#### 3.10.9.5 FBD Language




#### 3.10.9.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```



#### 3.10.9.7 IL Language:

```
Op1: LD  IN
     SQRT
     ST  Q   (* Q is: SQRT (IN) *)
```

**See also**

ABS TRUNC LOG POW

### 3.10.10 TRUNC / TRUNCL PLCopen

*Function* - Truncates the decimal part of the input.

#### 3.10.10.1 Inputs

`IN : REAL/LREAL` Real value

#### 3.10.10.2 Outputs

`Q : REAL/LREAL` Result: integer part of IN

#### 3.10.10.3 Remarks

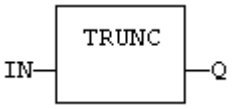
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.10.10.4 ST Language

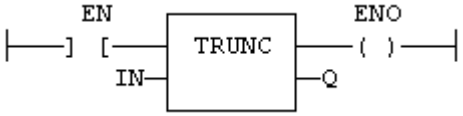
```
Q := TRUNC (IN);
```

#### 3.10.10.5 FBD Language




#### 3.10.10.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.10.10.7 IL Language:

```
Op1: LD  IN
     TRUNC
     ST  Q   (* Q is the integer part of IN *)
```

**See also**

ABS LOG POW SQRT

## 3.11 Trigonometric functions

Below are the standard functions for trigonometric calculation:

| | |
|---|---|
| SIN | sine |
| COS | cosine |
| TAN | tangent |
| ASIN | arc-sine |
| ACOS | arc-cosine |
| ATAN | arc-tangent |
| ATAN2 | arc-tangent of Y / X |

See Also:

UseDegrees

### 3.11.1 ACOS / ACOSL PLCopen

*Function* - Calculate an arc-cosine.

#### 3.11.1.1 Inputs

`IN : REAL/LREAL` Real value

#### 3.11.1.2 Outputs

`Q : REAL/LREAL` Result: arc-cosine of IN

#### 3.11.1.3 Remarks

In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.11.1.4 ST Language

```
Q := ACOS (IN);
```

#### 3.11.1.5 FBD Language


#### 3.11.1.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```


#### 3.11.1.7 IL Language:

```
Op1: LD  IN
     ACOS
     ST  Q   (* Q is: ACOS (IN) *)
```

**See also**

SIN COS TAN ASIN ATAN ATAN2

## 3.11.2 ASIN / ASINL PLCopen

*Function* - Calculate an arc-sine.

### 3.11.2.1 Inputs

`IN : REAL/LREAL` Real value

### 3.11.2.2 Outputs

`Q : REAL/LREAL` Result: arc-sine of IN

### 3.11.2.3 Remarks

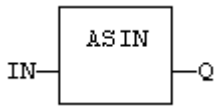
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

### 3.11.2.4 ST Language

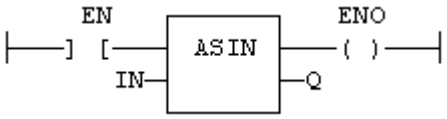
```
Q := ASIN (IN);
```

### 3.11.2.5 FBD Language




### 3.11.2.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




### 3.11.2.7 IL Language:

```
Op1: LD  IN
     ASIN
     ST  Q   (* Q is: ASIN (IN) *)
```

**See also**

SIN COS TAN ACOS ATAN ATAN2

## 3.11.3 ATAN / ATANL PLCopen

*Function* - Calculate an arc-tangent.

#### 3.11.3.1 Inputs

`IN : REAL/LREAL` Real value

#### 3.11.3.2 Outputs

`Q : REAL/LREAL` Result: arc-tangent of IN

#### 3.11.3.3 Remarks

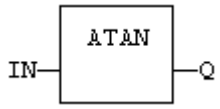
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.11.3.4 ST Language

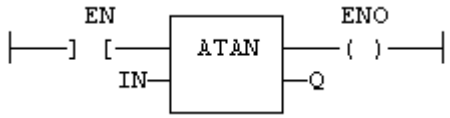
```
Q := ATAN (IN);
```

#### 3.11.3.5 FBD Language




#### 3.11.3.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.11.3.7 IL Language:

```
Op1: LD  IN
     ATAN
     ST  Q   (* Q is: ATAN (IN) *)
```

**See also**

SIN COS TAN ASIN ACOS ATAN2

### 3.11.4 ATAN2 / ATAN2L

*Function* - Calculate arc-tangent of Y/X

#### 3.11.4.1 Inputs

`Y : REAL/LREAL` Real value
`X : REAL/LREAL` Real value

#### 3.11.4.2 Outputs

`Q : REAL/LREAL` Result: arc-tangent of Y / X

#### 3.11.4.3 Remarks

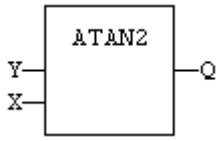
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.11.4.4 ST Language

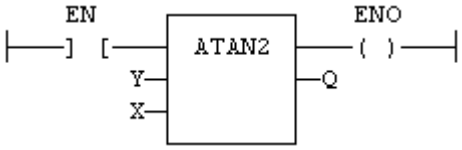
```
Q := ATAN2 (IN);
```

#### 3.11.4.5 FBD Language




#### 3.11.4.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.11.4.7 IL Language

```
Op1: LD  Y
     ATAN2 X
     ST  Q    (* Q is: ATAN2 (Y / X) *)
```

**See also**

SIN COS TAN ASIN ACOS ATAN

### 3.11.5 COS / COSL PLCopen

*Function* - Calculate a cosine.

#### 3.11.5.1 Inputs

`IN : REAL/LREAL` Real value

#### 3.11.5.2 Outputs

`Q : REAL/LREAL` Result: cosine of IN

#### 3.11.5.3 Remarks

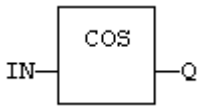
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.11.5.4 ST Language

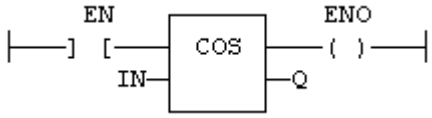
```
Q := COS (IN);
```

#### 3.11.5.5 FBD Language




#### 3.11.5.6 FFLD Language

(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)




#### 3.11.5.7 IL Language:

```
Op1: LD  IN
     COS
     ST  Q   (* Q is: COS (IN) *)
```

**See also**

SIN TAN ASIN ACOS ATAN ATAN2

### 3.11.6 SIN / SINL PLCopen

*Function* - Calculate a sine.

#### 3.11.6.1 Inputs

`IN : REAL/LREAL` Real value

#### 3.11.6.2 Outputs

`Q : REAL/LREAL` Result: sine of IN

#### 3.11.6.3 Remarks

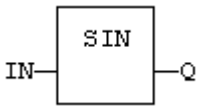
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.11.6.4 ST Language

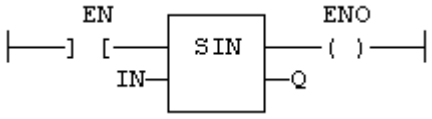
```
Q := SIN (IN);
```

#### 3.11.6.5 FBD Language




#### 3.11.6.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.11.6.7 IL Language:

```
Op1: LD  IN
     SIN
     ST  Q   (* Q is: SIN (IN) *)
```

**See also**

COS TAN ASIN ACOS ATAN ATAN2

### 3.11.7 TAN / TANL PLCopen

*Function* - Calculate a tangent.

#### 3.11.7.1 Inputs

`IN : REAL/LREAL` Real value

#### 3.11.7.2 Outputs

`Q : REAL/LREAL` Result: tangent of IN

#### 3.11.7.3 Remarks

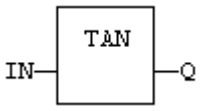
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.11.7.4 ST Language

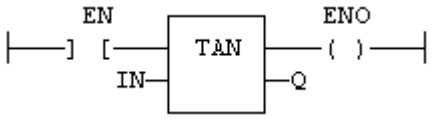
```
Q := TAN (IN);
```

#### 3.11.7.5 FBD Language




#### 3.11.7.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.11.7.7 IL Language:

```
Op1: LD  IN
     TAN
     ST  Q   (* Q is: TAN (IN) *)
```

**See also**

SIN COS ASIN ACOS ATAN ATAN2

## 3.11.8 UseDegrees PLCopen

*Function* - Sets the unit for angles in all trigonometric functions.

### 3.11.8.1 Inputs

| | |
|---|---|
| `IN : BOOL` | If TRUE, turn all trigonometric functions to use degrees<br>If FALSE, turn all trigonometric functions to use radians (default) |

### 3.11.8.2 Outputs

| | |
|---|---|
| `Q : BOOL` | TRUE if functions use degrees before the call |

### 3.11.8.3 Remarks

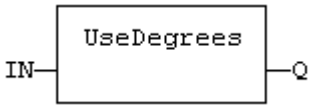
This function sets the working unit for the following functions:

| | |
|---|---|
| SIN | sine |
| COS | cosine |
| TAN | tangent |
| ASIN | arc-sine |
| ACOS | arc-cosine |
| ATAN | arc-tangent |
| ATAN2 | arc-tangent of Y / X |

### 3.11.8.4 ST Language

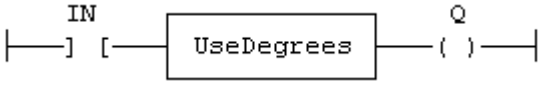
```
Q := UseDegrees (IN);
```

### 3.11.8.5 FBD Language




### 3.11.8.6 FFLD Language

```
(* Input is the rung. The rung is the output *)
```




### 3.11.8.7 IL Language

```
Op1: LD   IN
     UseDegrees
     ST   Q
```

## 3.12 String operations

Below are the standard operators and functions that manage character strings:

| | |
|---|---|
| + | concatenation of strings |
| CONCAT | concatenation of strings |
| MLEN | get string length |
| DELETE | delete characters in a string |
| INSERT | insert characters in a string |
| FIND | find characters in a string |
| REPLACE | replace characters in a string |
| LEFT | extract a part of a string on the left |
| RIGHT | extract a part of a string on the right |
| MID | extract a part of a string |
| CHAR | build a single character string |
| ASCII | get the ASCII code of a character within a string |
| ATOH | converts string to integer using hexadecimal basis |
| HTOA | converts integer to string using hexadecimal basis |
| CRC16 | CRC16 calculation |
| ArrayToString | copies elements of an SINT array to a STRING |
| StringToArray | copies characters of a STRING to an SINT array |

Other functions are available for managing string tables as resources:

| | |
|---|---|
| StringTable | Select the active string table resource |
| LoadString | Load a string from the active string table |

### 3.12.1 ArrayToString / ArrayToStringU


*Function* - Copy an array of SINT to a STRING.

#### 3.12.1.1 Inputs

| | |
|---|---|
| `SRC : SINT` | Source array of SINT small integers (USINT for ArrayToStringU) |
| `DST : STRING` | Destination STRING |
| `COUNT : DINT` | Numbers of characters to be copied |

#### 3.12.1.2 Outputs

| | |
|---|---|
| `Q : DINT` | Number of characters copied |

#### 3.12.1.3 Remarks

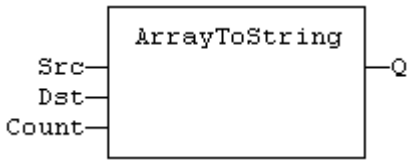
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

This function copies the COUNT first elements of the SRC array to the characters of the DST string. The function checks the maximum size of the destination string and adjust the COUNT number if necessary.

#### 3.12.1.4 ST Language

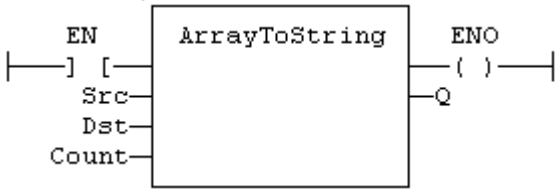
Q := ArrayToString (SRC, DST, COUNT);

#### 3.12.1.5 FBD Language




#### 3.12.1.6 FFLD Language

(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)




#### 3.12.1.7 IL Language

*Not available*

**See also**

StringToArray

### 3.12.2 ASCII

PLCopen

*Function* - Get the ASCII code of a character within a string

#### 3.12.2.1 Inputs

| | | |
|---|---|---|
| IN | : STRING | Input string |
| POS | : DINT | Position of the character within the string (The first valid position is 1) |

#### 3.12.2.2 Outputs

| | | |
|---|---|---|
| CODE | : DINT | ASCII code of the selected character or 0 if position is invalid |

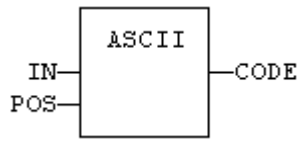
#### 3.12.2.3 Remarks

In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the same value as the input rung. In IL language, the first parameter (IN) must be loaded in the current result before calling the function. The other input is the operand of the function.

#### 3.12.2.4 ST Language

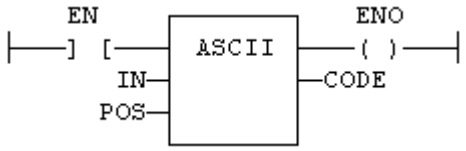
CODE := ASCII (IN, POS);

#### 3.12.2.5 FBD Language




#### 3.12.2.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO is equal to EN *)
```




#### 3.12.2.7 IL Language:

```
Op1: LD      IN
     AND_MASK MSK
     ST      CODE
```

**See also**

CHAR

### 3.12.3 ATOH PLCopen

*Function* - Converts string to integer using hexadecimal basis

#### 3.12.3.1 Inputs

`IN : STRING` String representing an integer in hexadecimal format

#### 3.12.3.2 Outputs

`Q : DINT` Integer represented by the string

#### 3.12.3.3 Truth table (examples)

| IN | Q |
|---|---|
| '' | 0 |
| '12' | 18 |
| 'a0' | 160 |
| 'A0zzz' | 160 |

#### 3.12.3.4 Remarks

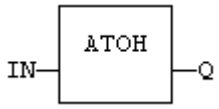
The function is case insensitive. The result is 0 for an empty string. The conversion stops before the first invalid character. In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.12.3.5 ST Language

```
Q := ATOH (IN);
```

#### 3.12.3.6 FBD Language




#### 3.12.3.7 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```


#### 3.12.3.8 IL Language:

```
Op1: LD  IN
     ATOH
     ST  Q
```

**See also**

HTOA

### 3.12.4 CHAR PLCopen

*Function* - Builds a single character string

#### 3.12.4.1 Inputs

`CODE : DINT` ASCII code of the wished character

#### 3.12.4.2 Outputs

`Q : STRING` STRING containing only the specified character

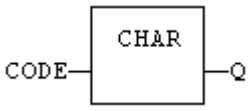
#### 3.12.4.3 Remarks

In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the same value as the input rung. In IL language, the input parameter (CODE) must be loaded in the current result before calling the function.

#### 3.12.4.4 ST Language

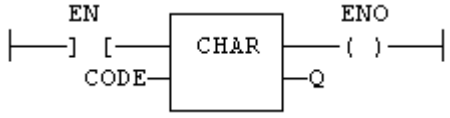
```
Q := CHAR (CODE);
```

#### 3.12.4.5 FBD Language




#### 3.12.4.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO is equal to EN *)
```




#### 3.12.4.7 IL Language:

```
Op1: LD   CODE
     CHAR
     ST   Q
```

**See also**

ASCII

## 3.12.5 CONCAT PLCopen

*Function* - Concatenate strings.

### 3.12.5.1 Inputs

`IN_1 : STRING` Any string variable or constant expression
...
`IN_N : STRING` Any string variable or constant expression

### 3.12.5.2 Outputs

`Q : STRING` Concatenation of all inputs

### 3.12.5.3 Remarks

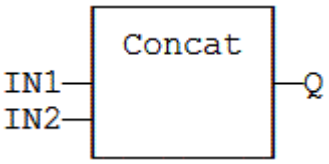
In FBD or FFLD language, the block can have up to 16 inputs. In IL or ST, the function accepts a variable number of inputs (at least 2).

Note that you also can use the "**+**" operator to concatenate strings.

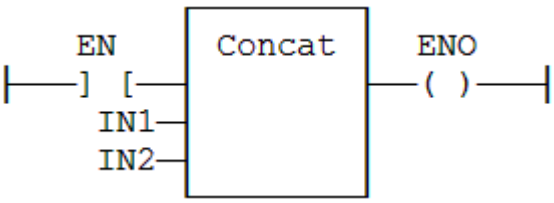
### 3.12.5.4 ST Language

```
Q := CONCAT ('AB', 'CD', 'E');
(* now Q is 'ABCDE' *)
```

### 3.12.5.5 FBD Language




### 3.12.5.6 FFLD Language




### 3.12.5.7 IL Language

```
Op1: FFLD    'AB'
   CONCAT 'CD', 'E'
   ST  Q   (* Q is now 'ABCDE' *)
```

## 3.12.6 CRC16 PLCopen

*Function* - calculates a CRC16 on the characters of a string

### 3.12.6.1 Inputs

`IN : STRING` character string

#### 3.12.6.2 Outputs

`Q : INT` CRC16 calculated on all the characters of the string.

#### 3.12.6.3 Remarks

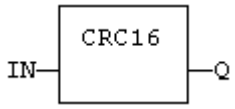
In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the same value as the input rung. In IL language, the input parameter (IN) must be loaded in the current result before calling the function.

The function calculates a Modbus CRC16, initialized at 16#FFFF value.

#### 3.12.6.4 ST Language

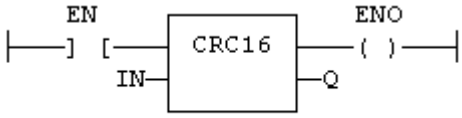
```
Q := CRC16 (IN);
```

#### 3.12.6.5 FBD Language




#### 3.12.6.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO is equal to EN *)
```




#### 3.12.6.7 IL Language:

```
Op1: LD   IN
     CRC16
     ST   Q
```

### 3.12.7 DELETE PLCopen

*Function* - Delete characters in a string.

#### 3.12.7.1 Inputs

`IN : STRING` Character string
`NBC : DINT` Number of characters to be deleted
`POS : DINT` Position of the first deleted character (first character position is 1)

#### 3.12.7.2 Outputs

`Q : STRING` Modified string.

#### 3.12.7.3 Remarks

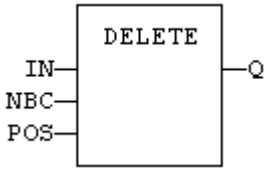
The first valid character position is 1. In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the first input (the string) must be loaded in the current result before calling the function. Other arguments are operands of the function, separated by comas.

#### 3.12.7.4 ST Language

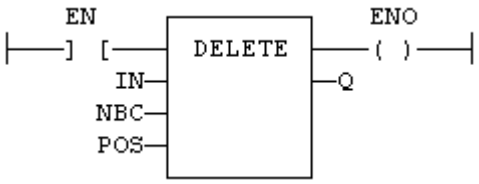
```
Q := DELETE (IN, NBC, POS);
```

#### 3.12.7.5 FBD Language




#### 3.12.7.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.12.7.7 IL Language:

```
Op1: LD    IN
     DELETE NBC, POS
     ST    Q
```

See also

+ MLEN INSERT FIND REPLACE LEFT RIGHT MID

### 3.12.8 FIND PLCopen

*Function* - Find position of characters in a string.

#### 3.12.8.1 Inputs

| | | |
|---|---|---|
| IN | STRING | Character string |
| STR | STRING | Specific characters to search for within the STRING |

#### 3.12.8.2 Outputs

| | | |
|---|---|---|
| POS | DINT | Position of the first character of STR in IN, or 0 if not found. |

#### 3.12.8.3 Remarks

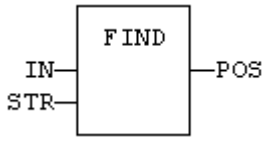
The first valid character position is 1. A return value of 0 means that the STR string has not been found. Search is case sensitive. In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the first input (the string) must be loaded in the current result before calling the function. The second argument is the operand of the function.

#### 3.12.8.4 ST Language

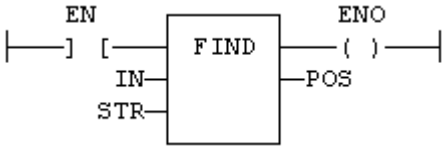
POS := FIND (IN, STR);

#### 3.12.8.5 FBD Language




#### 3.12.8.6 FFLD Language

(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)




#### 3.12.8.7 IL Language:

```
Op1: LD   IN
     FIND STR
     ST   POS
```

See also

+ MLEN DELETE INSERT REPLACE LEFT RIGHT MID

### 3.12.9 HTOA PLCopen

*Function* - Converts integer to string using hexadecimal basis

#### 3.12.9.1 Inputs

`IN : DINT` Integer value

#### 3.12.9.2 Outputs

`Q : STRING` String representing the integer in hexadecimal format

#### 3.12.9.3 Truth table (examples)

| IN | Q |
|---|---|
| 0 | '0' |
| 18 | '12' |
| 160 | 'A0' |

#### 3.12.9.4 Remarks

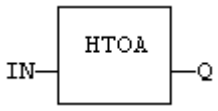
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

#### 3.12.9.5 ST Language

Q := HTOA (IN);

#### 3.12.9.6 FBD Language




#### 3.12.9.7 FFLD Language

(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)


#### 3.12.9.8 IL Language:

```
Op1: LD  IN
     HTOA
     ST  Q
```

See also

ATOH

### 3.12.10 INSERT PLCopen

*Function* - Insert characters in a string.

#### 3.12.10.1 Inputs

`IN : STRING` Character string
`STR : STRING` String containing characters to be inserted
`POS : DINT` Position of the first inserted character (first character position is 1)

#### 3.12.10.2 Outputs

`Q : STRING` Modified string.

#### 3.12.10.3 Remarks

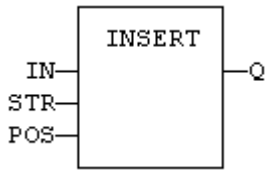
The first valid character position is 1. In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the first input (the string) must be loaded in the current result before calling the function. Other arguments are operands of the function, separated by comas.

#### 3.12.10.4 ST Language

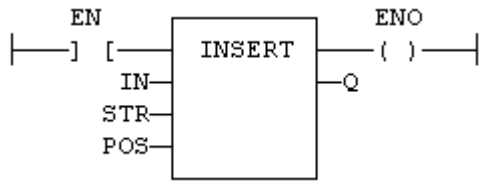
```
Q := INSERT (IN, STR, POS);
```

#### 3.12.10.5 FBD Language



#### 3.12.10.6 FFLD Language

(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)




#### 3.12.10.7 IL Language:

```
Op1: LD    IN
     INSERT STR, POS
     ST    Q
```

See also

+ MLEN DELETE FIND REPLACE LEFT RIGHT MID

### 3.12.11 LEFT PLCopen

*Function* - Extract characters of a string on the left.

#### 3.12.11.1 Inputs

`IN : STRING` Character string
`NBC : DINT` Number of characters to extract

#### 3.12.11.2 Outputs

`Q : STRING` String containing the first NBC characters of IN.

#### 3.12.11.3 Remarks

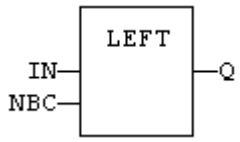
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the first input (the string) must be loaded in the current result before calling the function. The second argument is the operand of the function.

#### 3.12.11.4 ST Language

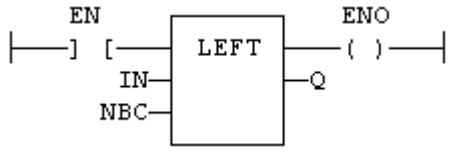
```
Q := LEFT (IN, NBC);
```

#### 3.12.11.5 FBD Language




#### 3.12.11.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.12.11.7 IL Language:

```
Op1: LD    IN
     LEFT  NBC
     ST    Q
```

**See also**

+ MLEN DELETE INSERT FIND REPLACE RIGHT MID

### 3.12.12 LoadString PLCopen

*Function* - Load a string from the active string table.

#### 3.12.12.1 Inputs

`ID: DINT` ID of the string as declared in the string table

#### 3.12.12.2 Outputs

`Q : STRING` Loaded string or empty string in case of error

#### 3.12.12.3 Remarks

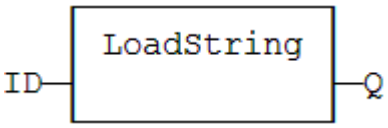
This function loads a string from the active string table and stores it into a STRING variable. The StringTable() function is used for selecting the active string table.

The "ID" input (the string item identifier) is an identifier such as declared within the string table resource. You don't need to "define" again this identifier. The system does it for you.

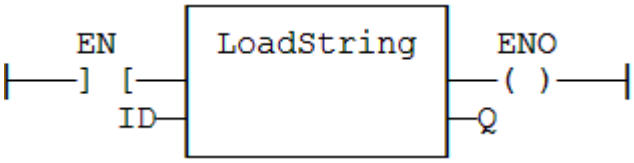
#### 3.12.12.4 ST Language

```
Q := LoadString (ID);
```

#### 3.12.12.5 FBD Language




#### 3.12.12.6 FFLD Language




#### 3.12.12.7 IL Language:

```
Op1: LD      ID
   LoadString
   ST       Q
```

**See also**

StringTable String tables

## 3.12.13 MID PLCopen

*Function* - Extract characters of a string at any position.

### 3.12.13.1 Inputs

`IN : STRING` Character string
`NBC : DINT` Number of characters to extract
`POS : DINT` Position of the first character to extract (first character of IN is at position 1)

### 3.12.13.2 Outputs

`Q : STRING` String containing the first NBC characters of IN.

### 3.12.13.3 Remarks

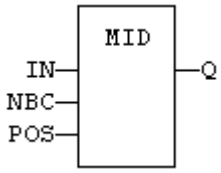
The first valid position is 1. In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the first input (the string) must be loaded in the current result before calling the function. Other arguments are operands of the function, separated by comas.

### 3.12.13.4 ST Language

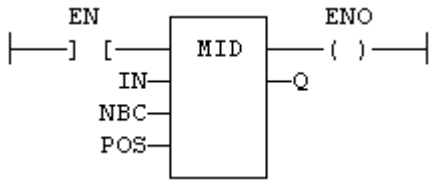
```
Q := MID (IN, NBC, POS);
```

### 3.12.13.5 FBD Language




### 3.12.13.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




### 3.12.13.7 IL Language:

```
Op1: LD   IN
   MID  NBC, POS
   ST   Q
```

**See also**

+ MLEN DELETE INSERT FIND REPLACE LEFT RIGHT

## 3.12.14 MLEN PLCopen

*Function* - Get the number of characters in a string.

### 3.12.14.1 Inputs

`IN : STRING` Character string

### 3.12.14.2 Outputs

`NBC : DINT` Number of characters currently in the string. 0 if string is empty.

### 3.12.14.3 Remarks

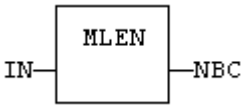
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

### 3.12.14.4 ST Language

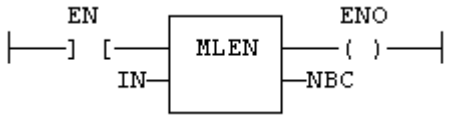
```
NBC := MLEN (IN);
```

### 3.12.14.5 FBD Language




### 3.12.14.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




### 3.12.14.7 IL Language:

```
Op1: LD   IN
     MLEN
     ST   NBC
```

See also

+ DELETE INSERT FIND REPLACE LEFT RIGHT MID

## 3.12.15 REPLACE PLCopen

*Function* - Replace characters in a string.

### 3.12.15.1 Inputs

`IN : STRING` Character string
`STR : STRING` String containing the characters to be inserted in place of NDEL removed characters

`NDEL : DINT` Number of characters to be deleted before insertion of STR
`POS : DINT` Position where characters are replaced (first character position is 1)

#### 3.12.15.2 Outputs

`Q : STRING` Modified string.

#### 3.12.15.3 Remarks

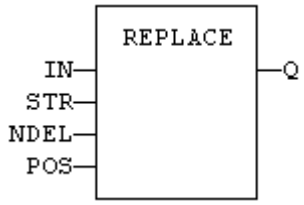
The first valid character position is 1. In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the first input (the string) must be loaded in the current result before calling the function. Other arguments are operands of the function, separated by comas.

#### 3.12.15.4 ST Language

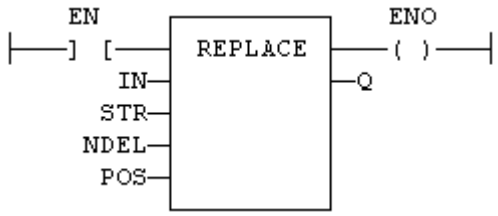
```
Q := REPLACE (IN, STR, NDEL, POS);
```

#### 3.12.15.5 FBD Language




#### 3.12.15.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.12.15.7 IL Language:

```
Op1: LD    IN
   REPLACE STR, NDEL, POS
   ST    Q
```

See also

+ MLEN DELETE INSERT FIND LEFT RIGHT MID

### 3.12.16 RIGHT PLCopen

*Function* - Extract characters of a string on the right.

#### 3.12.16.1 Inputs

`IN : STRING` Character string
`NBC : DINT` Number of characters to extract

#### 3.12.16.2 Outputs

`Q : STRING` String containing the last NBC characters of IN.

#### 3.12.16.3 Remarks

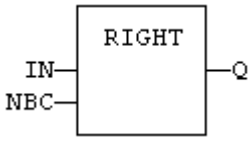
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the first input (the string) must be loaded in the current result before calling the function. The second argument is the operand of the function.

#### 3.12.16.4 ST Language

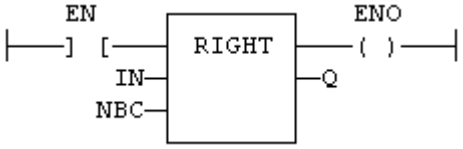
```
Q := RIGHT (IN, NBC);
```

#### 3.12.16.5 FBD Language




#### 3.12.16.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 3.12.16.7 IL Language:

```
Op1: LD    IN
     RIGHT NBC
     ST    Q
```

**See also**

+ MLEN DELETE INSERT FIND REPLACE LEFT MID

### 3.12.17 StringTable PLCopen

*Function* - Selects the active string table.

#### 3.12.17.1 Inputs

`TABLE : STRING` Name of the Sting Table resource - *must be a constant*
`COL : STRING` Name of the column in the table - *must be a constant*

#### 3.12.17.2 Outputs

`OK : BOOL` TRUE if OK

#### 3.12.17.3 Remarks

This function selects a column of a valid String Table resource to become the active string table. The LoadString() function always refers to the active string table.

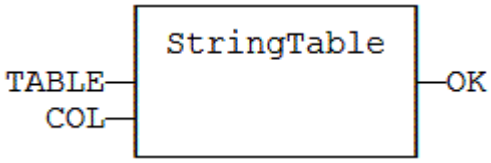
Arguments must be constant string expressions and must fit to a declared string table and a valid column name within this table.

If you have only one string table with only one column defined in your project, you do not need to call this function as it will be the default string table anyway.

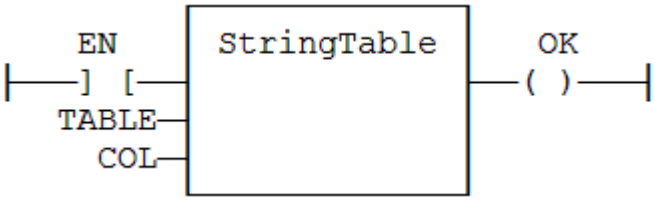
#### 3.12.17.4 ST Language

```
OK := StringTable ('MyTable', 'FirstColumn");
```

#### 3.12.17.5 FBD Language




#### 3.12.17.6 FFLD Language




#### 3.12.17.7 IL Language:

```
Op1: LD         'MyTable'
StringTable 'First Column'
ST          OK
```

See also

LoadString String tables

#### 3.12.17.8 String Table Resources

String tables are resources (embedded configuration data) edited with the Workbench. A string table is a list of items identified by a name and referring to one or more character strings.

String tables are typically used for defining static texts to be used in the application. The following functions can be used for getting access to string tables in the programs:

StringTable: selects the active string table.
LoadString: Load a string from the active table.

In addition, each string table may contain several columns of texts for each item, and thus ease the localization of application, simply by defining a column for each language. This way the language can be selected dynamically at runtime, simply by specifying the active language (as a column) in the StringTable() function.

The name entered in the string table as an "ID" is automatically declared for the compiler and can directly be passed to the LoadString() function without re-declaring it. The name must conform to IEC standard naming rules.

Obviously, you could do the same by declaring an array of STRING variables and enter some initial values for all items in the array. By the way, string tables provide significant advantages compared to arrays:

- The editor provides a comfortable view of multiple columns at editing.
- String tables are loaded in the application code and does not require any further RAM memory unlike declared arrays.
- The string table editor automatically declares for you readable IDs for any string item to be used in programs instead of working with hard-coded index values.

**TIP**

If the text is too long for the STRING variable when used at runtime, it is truncated.
You can use special '$' sequences in strings in order to specify non printable characters, according to the IEC standard:

| Code | Meaning |
|---|---|
| $$ | A "$" character |
| $' | A Single quote |
| $T | A tab stop (ASCII code 9) |
| $R | A carriage return character (ASCII code 13 |
| $L | A line feed character (ASCII code 10) |
| $N | Carriage return plus line feed characters (ASCII codes 13 and 10) |
| $P | A page break character (ASCII code 12) |
| $xx | Any character (xx is the ASCII code expressed on two hexadecimal digits |

## 3.12.18 StringToArray / StringToArrayU

PLCopen

*Function* - Copies the characters of a STRING to an array of SINT.

### 3.12.18.1 Inputs

`SRC : STRING` Source STRING
`DST : SINT` Destination array of SINT small integers (USINT for StringToArrayU)

### 3.12.18.2 Outputs

`Q : DINT` Number of characters copied

### 3.12.18.3 Remarks

In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

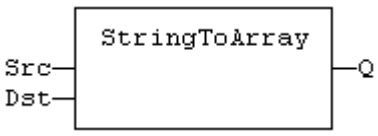
In IL, the input must be loaded in the current result before calling the function.

This function copies the characters of the SRC string to the first characters of the DST array. The function checks the maximum size destination arrays and reduces the number of copied characters if necessary.

### 3.12.18.4 ST Language

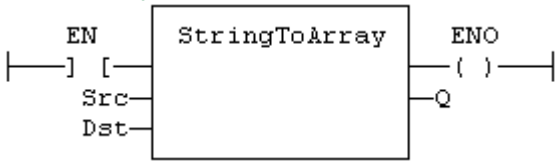
```
Q := StringToArray (SRC, DST);
```

### 3.12.18.5 FBD Language




### 3.12.18.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




### 3.12.18.7 IL Language:

```
Op1: LD          SRC
StringToArray DST
ST          Q
```

**See also**

ArrayToString

# 4 PLC Advanced Libraries

Below are the standard blocks that perform advanced operations.

**Analog signal processing:**

| Block | Description |
|---|---|
| Average / AverageL | Calculate the average of signal samples |
| Integral | Calculate the integral of a signal |
| Derivate | Derive a signal |
| PID | PID loop |
| Ramp | Ramp signal |
| Rand | Give a Random value modulo the input value |
| Lim_Alrm | Low / High level detection |
| Hyster, HysterAcc | Hysterisis calculation |
| SigPlay | Play an analog signal from a resource |
| SigScale | Get a point from a signal resource |
| CurveLin | Linear interpolation on a curve |
| SurfLin | Linear interpolation on a surface |

**Alarm management:**

| Block | Description |
|---|---|
| Lim_Alrm | Low / High level detection |
| Alarm_M | Alarm with manual reset |
| ALARM_A | Alarm with automatic reset |

**Data collections and serialization:**

| Block | Description |
|---|---|
| StackInt | Stack of integers |
| FIFO | "First in / first out" list |
| LIFO | "Last in / first out" stack |
| SerializeIn | Extract data from a binary frame. |
| SerializeOut | Write data to a binary frame. |

**Data Logging:**

| Block | Description |
|---|---|
| LogFileCSV | Log values of variables to a CSV file |

**Special operations:**

| Block | Description |
|---|---|
| GetSysInfo | Get system information |
| Printf | Trace messages |
| CycleStop | Sets the application in cycle stepping mode |
| FatalStop | Breaks the cycle and stop with fatal error |

| Block | Description |
|---|---|
| EnableEvents | Enable / disable produced events for binding |
| ApplyRecipeColumn | Apply the values of a column from a recipe file |
| VLID | Get the ID of an embedded list of variables |
| SigID | Get the ID of a signal resource |

**Communication:**

AS-interface

**Others:**

Real Time Clock

File Tools Function Blocks

## 4.1 ALARM_A PLCopen

*Function Block* - Alarm with automatic reset

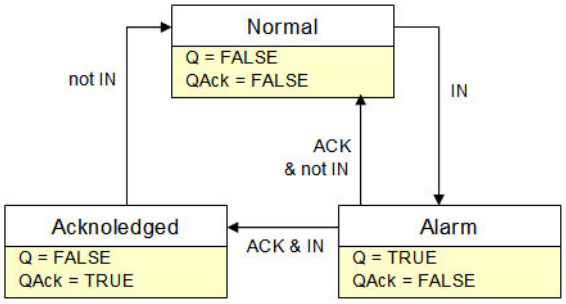
### 4.1.1 Inputs

```
IN    : BOOL  Process signal
ACK   : BOOL  Acknowledge command
```

### 4.1.2 Outputs

```
Q     : BOOL  TRUE if alarm is active
QACK  : BOOL  TRUE if alarm is acknowledged
```

### 4.1.3 Sequence



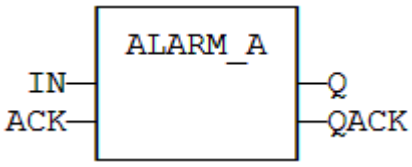

### 4.1.4 Remarks

Combine this block with the LIM_ALRM block for managing analog alarms.

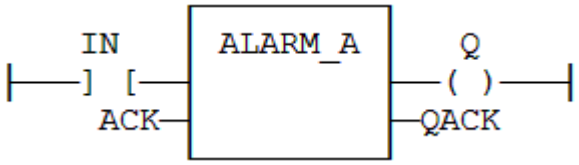
### 4.1.5 ST Language

```
(* MyALARM is declared as an instance of ALARM_A function block *)
MyALARM (IN, ACK);
Q := MyALARM.Q;
QACK := MyALARM.QACK;
```

### 4.1.6 FBD Language


ALARM_A
IN
ACK
Q
QACK

### 4.1.7 FFLD Language




### 4.1.8 IL Language

```
(* MyALARM is declared as an instance of ALARM_A function block *)
Op1: CAL
MyALARM (IN, ACK)
FFLD  MyALARM.Q
ST  Q
FFLD  MyALARM.QACK
ST
QACK
```

**See also**

ALARM_M LIM_ALRM

## 4.2 ALARM_M PLCopen

*Function Block* - Alarm with manual reset

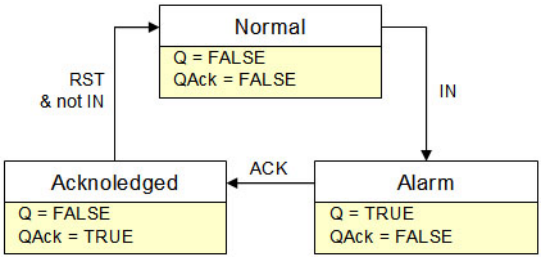
### 4.2.1 Inputs

| | | | |
|---|---|---|---|
| IN | : | BOOL | Process signal |
| ACK | : | BOOL | Acknowledge command |
| RST | : | BOOL | Reset command |

### 4.2.2 Outputs

| | | | |
|---|---|---|---|
| Q | : | BOOL | TRUE if alarm is active |
| QACK | : | BOOL | TRUE if alarm is acknowledged |

### 4.2.3 Sequence



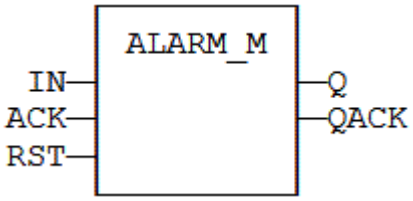

### 4.2.4 Remarks

Combine this block with the LIM_ALRM block for managing analog alarms.

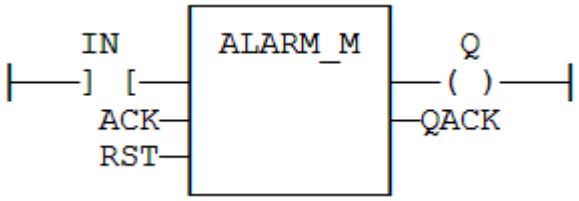
### 4.2.5 ST Language

```
(* MyALARM is declared as an instance of ALARM_M function block *)
MyALARM (IN, ACK, RST);
Q := MyALARM.Q;
QACK := MyALARM.QACK;
```

### 4.2.6 FBD Language



### 4.2.7 FFLD Language




### 4.2.8 IL Language

```
(* MyALARM is declared as an instance of ALARM_M function block *)
Op1: CAL
MyALARM (IN, ACK, RST)
FFLD  MyALARM.Q
ST  Q
FFLD  MyALARM.QACK
ST
QACK
```

**See also**

ALARM_A LIM_ALRM

## 4.3 ApplyRecipeColumn PLCopen

*Function* - Apply the values of a column from a recipe file

### 4.3.1 Inputs

| | |
|---|---|
| `FILE : STRING` | Path name of the recipe file (.RCP or .CSV) - *must be a constant value!* |
| `COL : DINT` | Index of the column in the recipe (0 based) |

*See an example of RCP file*

```
@COLNAME=Col3 Col4
@SIZECOL1=100
@SIZECOL2=100
@SIZECOL3=100
@SIZECOL4=100
bCommand
tPerio
bFast
Blink1
test_var
bOut
@EXPANDED=Blink1
```

*See an example of CSV file*

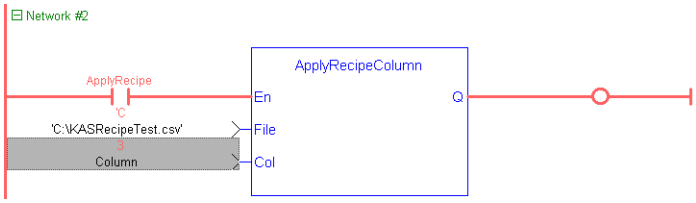
Example of CSV file with five variables and five set of values

```
comment lines here
TravelSpeed;100;200;300;400;500
MasterAbsPos;0;45;90;135;180
MasterDeltaPos;0;90;180;270;360
MachineSpeed;50;100;150;200;250
MachineState;0;0;1;1;2
```

**NOTE**

For your CSV file to be valid, ensure the data are separated with **semicolons** (and not commas).

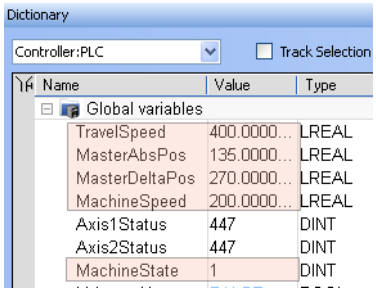
Usage in a FFLD program where column 3 is selected




Column 3 corresponds to column E in the Excel sheet because this parameter is 0 based

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | comment lines here | | | | | |
| 2 | TravelSpeed | 100 | 200 | 300 | 400 | 500 |
| 3 | MasterAbsPos | 0 | 45 | 90 | 135 | 180 |
| 4 | MasterDeltaPos | 0 | 90 | 180 | 270 | 360 |
| 5 | MachineSpeed | 50 | 100 | 150 | 200 | 250 |
| 6 | MachineState | 0 | 0 | 1 | 1 | 2 |

Result displayed in the Dictionary when the application is running




### 4.3.2 Outputs

`OK : BOOL` TRUE if OK - FALSE if parameters are invalid

### 4.3.3 Remarks

- The 'FILE' input is a constant string expression specifying the path name of a valid .RCP or .CSV file. If no path is specified, the file is assumed to be located in the project folder. RCP files are created using an external recipe editor. CSV files can be created using EXCEL or NOTEPAD.

- In CSV files, the first line must contain column headers, and is ignored during compiling. There is one variable per line. The first column contains the symbol of the variable. Other columns are values.
- If a cell is empty, it is assumed to be the same value as the previous (left side) cell. If it is the first cell of a row, it is assumed to be null (0 or FALSE or empty string).
- In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung is the result of the function.

**IMPORTANT**

Recipe files are read at compiling time and are embedded into the downloaded application code. This implies that a modification performed in the recipe file after downloading is not taken into account by the application.

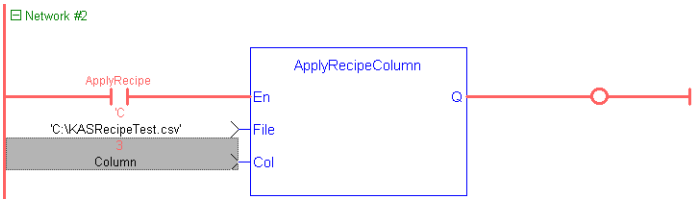
### 4.3.4 ST Language

```
OK := ApplyRecipeColumn ('MyFile.rcp', COL);
```

### 4.3.5 FBD Language


### 4.3.6 FFLD Language

(* The function is executed only if ApplyRecipe is TRUE *)




### 4.3.7 IL Language

```
Op1: LD
'MyFile.rcp'
ApplyRecipeColumn COL
ST
OK
```

## 4.4 AS-interface Functions

The following functions enable special operation on AS-i networks:

| | |
|---|---|
| ASiReadPP | read permanent parameters of an AS-i slave |
| ASiWritePP | write permanent parameters of an AS-i slave |
| ASiSendParam | send parameters to an AS-i slave |
| ASiReadPI | read actual parameters of an AS-i slave |
| ASiStorePI | store actual parameters as permanent parameters |

**IMPORTANT**

AS-i networking may be not available on some targets. Please refer to OEM instructions for further details about available features.

Interface

```
Params := ASiReadPP (Master, Slave);
bOK := ASiWritePP (Master, Slave, Params);
bOK := ASiSendParam (Master, Slave, Params);
Params := ASiReadPI (Master, Slave);
bOK := ASiStorePI (Master);
```

Arguments

| | |
|---|---|
| Master : DINT | Index of the AS-i master (1..N) such as shown in configuration |
| Slave : DINT | Address of the AS-i slave (1..32 / 33..63) |
| Params : DINT | Value of AS-i parameters |
| bOK : BOOL | TRUE if successful |

## 4.5 AVERAGE / AVERAGEL PLCopen

*Function Block* - Calculates the average of signal samples.

### 4.5.1 Inputs

`RUN : BOOL` Enabling command
`XIN : REAL` Input signal
`N : DINT` Number of samples stored for average calculation - Cannot exceed 128

### 4.5.2 Outputs

`XOUT : REAL` Average of the stored samples (*)

(*) AVERAGEL has LREAL arguments.

### 4.5.3 Remarks

The average is calculated according to the number of stored samples, which can be less than N when the block is enabled. By default the number of samples is 128.

The "N" input (or the number of samples) is taken into account *only* when the RUN input is FALSE.

**TIP**

The "RUN" needs to be reset after a change in the number of samples. You should cycle the RUN input when you first call this function, this will clear the default.

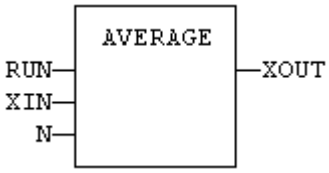
**NOTE**

In FFLD language, the input rung is the RUN command. The output rung keeps the state of the input rung.

### 4.5.4 ST Language

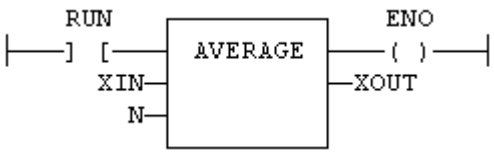
```
(* MyAve is a declared instance of AVERAGE function block *)
MyAve (RUN, XIN, N);
XOUT := MyAve.XOUT;
```

### 4.5.5 FBD Language




### 4.5.6 FFLD Language

(* ENO has the same state as RUN *)



### 4.5.7 IL Language:

```
(* MyAve is a declared instance of AVERAGE function block *)
Op1: CAL MyAve (RUN, XIN, N)
     FFLD  MyAve.XOUT
     ST  XOUT
```

See also

INTEGRAL DERIVATE LIM_ALRM HYSTER STACKINT

## 4.6 CurveLin PLCopen

*Function block*- Linear interpolation on a curve.

### 4.6.1 Inputs

X : REAL X coordinate of the point to be interpolated.

XAxis : REAL[] X coordinates of the known points of the X axis.

YVal : REAL[] Y coordinate of the points defined on the X axis.

### 4.6.2 Outputs

Y : REAL Interpolated Y value corresponding to the X input

OK : BOOL TRUE if successful.

ERR : DINT Error code if failed - 0 if OK.

### 4.6.3 Remarks

This function performs linear interpolation in between a list of points defined in the XAxis single dimension array. The output Y value is an interpolation of the Y values of the two rounding points defined in the X axis. Y values of defined points are passed in the YVal single dimension array.

Values in XAxis must be sorted from the smallest to the biggest. There must be at least two points defined in the X axis. YVal and XAxis input arrays must have the same dimension.

In case the X input is less than the smallest defined X point, the Y output takes the first value defined in YVal and an error is reported. In case the X input is greater than the biggest defined X point, the Y output takes the last value defined in YVal and an error is reported.

The ERR output gives the cause of the error if the function fails:

| Error Code | Meaning |
|---|---|
| 0 | OK |
| 1 | Invalid dimension of input arrays |
| 2 | Invalid points for the X axis |
| 4 | X is out of the defined X axis |

## 4.7 DERIVATE PLCopen

*Function Block* - Computes the derivative of a signal with respect to time.

The time unit is seconds. The output signal has the units of the input signal divided by seconds. The DERIVATE block samples the input signal at a maximum rate of 1 millisecond.

### 4.7.1 Inputs

`RUN : BOOL` Run command: TRUE=derivate / FALSE=hold
`XIN : REAL` Input signal
`CYCLE : TIME` Sampling period (must not be less than the target cycle timing)

### 4.7.2 Outputs

`XOUT : REAL` Output signal

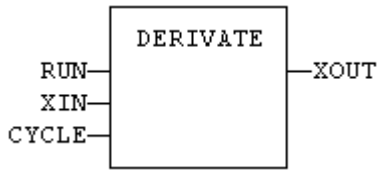
### 4.7.3 Remarks

In FFLD language, the input rung is the RUN command. The output rung keeps the state of the input rung.

### 4.7.4 ST Language

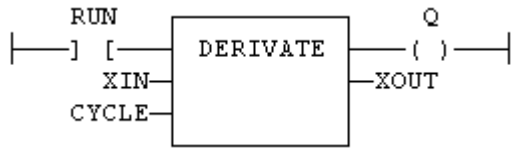
```
(* MyDerv is a declared instance of DERIVATE function block *)
MyDerv (RUN, XIN, CYCLE);
XOUT := MyDerv.XOUT;
```

### 4.7.5 FBD Language




### 4.7.6 FFLD Language

```
(* ENO has the same state as RUN *)
```




### 4.7.7 IL Language:

```
(* MyDerv is a declared instance of DERIVATE function block *)
Op1: CAL MyDerv (RUN, XIN, CYCLE)
FFLD  MyDerv.XOUT
ST  XOUT
```

See also

AVERAGE INTEGRAL LIM_ALRM HYSTERSTACKINT

## 4.8 EnableEvents PLCopen

*Function* - Enable or disable the production of events for binding(runtime to runtime variable exchange)

### 4.8.1 Inputs

`EN : BOOL` TRUE to enable events / FALSE to disable events

### 4.8.2 Outputs

`ENO : BOOL` Echo of EN input

### 4.8.3 Remarks

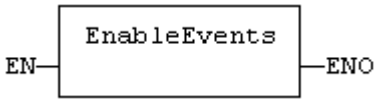
Production is enabled when the application starts. The first production will be operated after the first cycle. So to disable events since the beginning, you must call `EnableEvents (FALSE)` in the very first cycle.

In FFLD language, the input rung (EN) enables the event production, and the output rung keeps the state of the input rung. In IL language, the input must be loaded before the function call.

### 4.8.4 ST Language

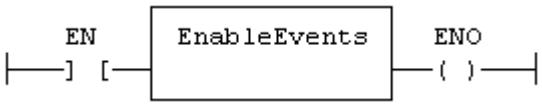
```
ENO := EnableEvents (EN);
```

### 4.8.5 FBD Language




### 4.8.6 FFLD Language

```
(* Events are enables if EN is TRUE *)
(* ENO has the same value as EN *)
```




### 4.8.7 IL Language:

```
Op1: LD  EN
     EnableEvents
     ST  ENO
```

## 4.9 FIFO PLCopen

*Function block* - Manages a "first in / first out" list

### 4.9.1 Inputs

| | | |
|---|---|---|
| PUSH | BOOL | Push a new value (on rising edge) |
| POP | BOOL | Pop a new value (on rising edge) |
| RST | BOOL | Reset the list |
| IN | ANY | Value to be pushed |

| | | |
|---|---|---|
| @Tail | ANY | Value of the oldest pushed value - *updated after call!* |
| Buf[] | ANY | Array for storing values |

### 4.9.2 Outputs

| | | |
|---|---|---|
| EMPTY | BOOL | TRUE if the list is empty |
| OFLO | BOOL | TRUE if overflow on a PUSH command |
| Count | DINT | Number of values in the list |
| pRead | DINT | Index in the buffer of the oldest pushed value |
| pWrite | DINT | Index in the buffer of the next push position |

### 4.9.3 Remarks

IN, @Tail and Buf[] must have the same data type *and cannot be STRING*.

The @Tail argument specifies a variable which is filled with the oldest push value after the block is called.

Values are stored in the "BUF" array. Data is arranged as a roll over buffer and is never shifted or reset. Only read and write pointers and pushed values are updated. The maximum size of the list is the dimension of the array.

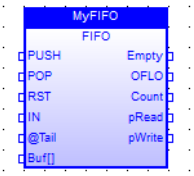
The first time the block is called, it remembers on which array it should work. If you call later the same instance with another BUF input, the call is considered as invalid and makes nothing. Outputs reports an empty list in this case.

In FFLD language, input rung is the PUSH input. The output rung is the EMPTY output.

### 4.9.4 ST Language

```
(* MyFIFO is a declared instance of FIFO function block *)
MyFIFO (PUSH, POP, RST, IN, @Tail , BUFFER);
EMPTY := MyFIFO.EMPTY;
OFLO := MyFIFO.OFLO;
COUNT := MyFIFO.COUNT;
PREAD := MyFIFO.PREAD;
PWRITE := MyFIFO.PWRITE;
```

### 4.9.5 FBD Language




### 4.9.6 FFLD Language


### 4.9.7 IL Language

```
(* MyFIFO is a declared instance of FIFO function block *)
Op1: CAL MyFIFO (PUSH, POP, RST, IN, @Tail , BUFF[])
FFLD  MyFIFO.EMPTY
ST  EMPTY
FFLD  MyFIFO.OFLO
ST  OFLO
FFLD  MyFIFO.COUNT
ST  COUNT
FFLD  MyFIFO.PREAD
ST  PREAD
FFLD  MyFIFO.PWRITE
ST  PWRITE
```

**See also**

LIFO

## 4.10 File Management

File Management functions provide the ability to do the following:

- Read machine recipes or other machine operational data into the .kas program from the SD card or a shared directory.
- Read cam tables into the program from the SD card or a shared directory.
- Store machine operational data in internal PxMM flash memory (retrievable through the web server), the SD card, or a shared directory.
  *Please note that a shared directory connection is setup through the web server.*

**TIP**

- If needed, functions to parse out information from a file using a string format can be found in String operations.
- If the file is in .CSV format, the following functions can be used: LogFileCSV, ApplyRecipeColumn

- You can create, store, and retrieve recipes and other data using:
  - the AKI Terminals. For more information see the KVB manual.
  - through an external bus connection to the PxMM with a supported fieldbus, such as UDP or HTTP.

The following function blocks enable sequential read / write operations in disk files:

| Function Block | Use |
|---|---|
| FileClose | Close an open file |
| FileCopy | Copy a file |
| FileDelete | Remove a file |
| FileEOF | Test if the end of the file is reached in a file that is open for reading |
| FileExists | Test if a file exists |
| FileOpenA | Create or open a file in append mode |
| FileOpenR | Open a file for reading |
| FileOpenW | Create or reset a file and open it for writing |
| FileReadBinData | Read binary data from a file |
| FileReadLine | Read a string value from a text file |
| FileRename | Rename a file |
| FileSeek | Set the current position of a file |
| FileSize | Get the size of a file |
| FileWriteBinData | Write binary data to a file |
| FileWriteLine | Write a string value to a text file |

The following functions handle mounting of SD cards.

| Name | Use |
|---|---|
| SD_ISREADY | Check that the SD card is ready for read/write |
| SD_MOUNT | Mount an SD card |
| SD_UNMOUNT | Unmount an SD card |

Each file is identified in the application by a unique handle manipulated as a DINT value. The file handles are allocated by the target system. Handles are returned by the Open function blocks and used by all other function blocks for identifying the file.

**Related function blocks:**

`LogFileCSV` log values of variables to a CSV file

**IMPORTANT**

- Files are opened and closed directly by the Operating System of the target. Opening some files can be dangerous for system safety and integrity. **The number of open files (from FileOpenA, FileOpenR, and FileOpenW) is limited by the resources available on the target system.**
- Ensure that each file successfully opened using FileOpenA, FileOpenR, and FileOpenW has a corresponding FileClose to close the file. Closing the file will release the file ID, making it available for operations on other files.
- The deprecated functions will stall the PLC, causing the controller to miss PLC cycles.
- The IDs used by the file function blocks are not compatible with the deprecated file functions.
- The IDs used by the deprecated file functions are not compatible with the file function blocks.

**NOTE**

- Opening a file with FileOpenA, FileOpenR, and FileOpenW can be unsuccessful (invalid path or file name, too many open files...) Your application must check the file ID for a NULL value. If the file ID is NULL (zero), then file read or write operations will fail.
- File management may be unavailable on some targets.
- Memory on the SD card is available in addition to the existing flash memory.
- Valid paths for storing files depend on the target implementation.
- Error messages are logged in the Controller log section of KAS Runtime where there is a failure in any related function block.
- Using the KAS Simulator, all pathnames are ignored, and files are stored in a reserved directory. Only the file name passed to the Open functions is taken into account.
- AKD PDMM / PCMM files are big endian.

**TIP**

Be sure to review File Path Conventions to understand hardware-based functional differences.

### 4.10.1 SD Card Access

Files may be written to and read from an SD card. This is typically used for storing a firmware image for Recovery Mode.

To use an SD card on the controller:

1. Ensure that the SD card is inserted.
2. Mount the card using SD_MOUNT.
3. Ensure the card is accessible using SD_ISREADY before performing a read or write action.
4. Unmount the card, if desired, using SD_UNMOUNT after performing read/write actions.

### 4.10.2 SD Card Mounting Functions

The following functions handle mounting of SD cards.

| Name | Use |
|---|---|
| SD_ISREADY | Check that the SD card is ready for read/write |
| SD_MOUNT | Mount an SD card |
| SD_UNMOUNT | Unmount an SD card |

#### 4.10.2.1 SD_MOUNT PLCopen

Mount the SDCard on the PDMM. This will not perform any action, and always return TRUE with a Simulator.

```
OK := SD_MOUNT();
```

| | |
|---|---|
| **OK : BOOL** | TRUE if mounting SDCard is successful |

**NOTE**

Before performing, make sure the SDCard is inserted.

**TIP**

It is recommended that SD_MOUNT be used only when motion is not started.

#### 4.10.2.2 SD_UNMOUNT PLCopen

Un-mount the SDCard from the PDMM. This will not perform any action, and always return TRUE with a Simulator.

```
OK := SD_UNMOUNT();
```

| | |
|---|---|
| **OK : BOOL** | TRUE if un-mounting SDCard is successful |

**TIP**

It is recommended that SD_UNMOUNT be used only when motion is not started.

#### 4.10.2.3 SD_ISREADY PLCopen

Verify if the SDCard is mounted on the PDMM. This will verify if the SDCard folder is available inside the userdata folder when using a Simulator.

```
OK := SD_ISREADY();
```

| | |
|---|---|
| **OK : BOOL** | TRUE if the SDCard is mounted (PDMM / PCMM) |

### 4.10.3 File Path Conventions

Depending upon the system used, paths to file locations may be defined as either absolute (`C://dir1/file1`) or relative paths (`/dir1/file1`). Not all systems handle all options, and the paths will vary depending upon the system.

| System | Absolute Paths | Relative Paths | Handling of Directories |
|---|---|---|---|
| **AKD PDMM or PCMM** | ✗ | ✓ | There is no support for creating directories on the controller. Any path provided to the function blocks, `file1` for example, will be appended with the default userdata folder which is: `/mount/flash/userdata` |
| **Simulator** | ✓ | ✓ | When a relative path is provided to the function blocks, the path is appended with the default userdata folder, which is: `<User Directory>/Kollmorgen/Kollmorgen Automation Suite/Sinope Simulator/Application/userdata` |

**See Also:**

- Shared Directory Path Conventions
- SD Card Path Conventions
- File Name Warning & Limitations

#### 4.10.3.1 Shared Directory Path Conventions

The AKD PDMM and PCMM support access to a shared directory on a remote computer. To access files in a shared directory from the AKD PDMM orPCMM use `'/mount/shared'` at the beginning of the path, before the shared directory's relative path and file name:

**`'/mount/shared/`***`directory/filename`*`'`

| Valid Paths | Notes |
|---|---|
| /mount/shared | The path is not case sensitive, so /MOUNT/SHARED, MOUNT/SHARED/, etc. are also valid. |
| mount/shared | |
| \mount\shared | |
| mount\shared | |

**Example 1:** Opening the file `example.txt` from a shared directory on a remote computer.

```
fileID := F_AOPEN('/mount/shared/example.txt');
```

**Example 2:** Opening the file `myfiles/example.txt` from a shared directory on a remote computer.

```
fileID := F_AOPEN('/mount/shared/myfiles/example.txt');
```

**See Also:**

- File Name Warning & Limitations

#### 4.10.3.2 SD Card Path Conventions

To access the SD card memory a valid SD card label must be used at the beginning of the path, followed by the relative path to the SD card. *(Valid SD Card Label)/(Relative Path)*

A valid SD card relative path starts with `//`, `/`, `\\`, or `\`. This is immediately followed by `SDCard` which is followed by `\` or `/`. Please note that this path label is case insensitive.

| Valid Paths | Notes |
|---|---|
| //SDCard/file1 | |
| \Sdcard/dir1/file1 | dir1 must have been already created |
| /sdcard/dir1/file1 | dir1 must have been already created |
| //sdCard\file1 | |

| Invalid Paths | Reason for being invalid |
|---|---|
| ///SDCard/file1 | Started with more than two forward or backward slashes |
| /\Sdcard/dir1/file1 | Started with one forward and one backward slash |
| /sdcarddir1/file1 | No forward or backward slash |
| /sdcard1/dir1/file1 | Invalid label |

In order to maintain compatibility with the Simulator, the `SDCard` folder is created inside the userdata folder . File access points to `userdata/SDCard` when a PDMM SDCard path is used on the Simulator.

**See Also:**

- File Name Warning & Limitations

#### 4.10.3.3 File Name Warning & Limitations

File names in the AKD PDMM or PCMM flash storage are case-sensitive and the SD card (FAT16 or FAT32) are not case-sensitive.

| Storage | File System | Case-Sensitive |
|---|---|---|
| AKD PDMM / PCMM embedded flash | FFS3 (POSIX-like) | Yes |
| AKD PDMM / PCMM SD card | FAT16 or FAT32 | No |

For example, two files (`MyFile.txt` and `myfile.txt`) can exist in the same directory of the PDMM flash, but cannot exist in the same directory on the PDMM's SD card. If you copy two files (via backup operation or function) with the same name, but different upper/lower case letters, from the PDMM flash to the SD card, one of the files will be lost. **To prevent conflicts and to keep your application compatible across all platforms, use unique filenames and do not rely on case-sensitive filenames.**

### 4.10.4 *DEPRECATED* File Management Functions

The file management functions have been deprecated in favor of the Function Blocks in the following table.

| Function Block | Deprecated Functions | Use |
|---|---|---|
| FileClose | F_CLOSE | Close an open file |
| FileCopy | F_COPY | Copy a file |
| FileDelete | F_DELETE | Remove a file |
| FileEOF | F_EOF | Test if the end of the file is reached in a file that is open for reading |
| FileExists | F_EXIST | Test if a file exists |
| FileOpenA | F_AOPEN | Create or open a file in append mode |
| FileOpenR | F_ROPEN | Open a file for reading |
| FileOpenW | F_WOPEN | Create or reset a file and open it for writing |
| FileReadBinData | FA_READ<br>FB_READ | Read binary data from a file |
| FileReadLine | FM_READ | Read a string value from a text file |
| FileRename | F_RENAME | Rename a file |
| FileSeek | F_SEEK | Set the current position of a file |
| FileSize | F_GETSIZE | Get the size of a file |
| FileWriteBinData | FA_WRITE<br>FB_WRITE | Write binary data to a file |
| FileWriteLine | FM_WRITE | Write a string value to a text file |

### 4.10.5 File Management Function Examples

Following are several examples of how File Management functions may be used. The functions used include FileOpenA, FileClose, FileOpenW, and FileWriteLine.

```
// Determine if this is a UNIX-based or Windows operating system and set
the directory.
ID:= FileOpenA('C:\Program Files\Kollmorgen\Kollmorgen Automation
Suite\Sinope Runtime\Resources\http.conf');
IF ID > 0 THEN
   OutputFile := '\' + FileName;
 FileClose(ID);
ELSE
   OutputFile := FileName;
```

```
END_IF;

IF (AddFileExt = true) THEN
   OutputFile := OutputFile + '.csv';
END_IF;
```

```
// Create a file for writing
FileID := FileOpenW (OutputFile);
IF (FileID = 0) THEN
    RETURN;
END_IF;
```

```
// Write header information to a file
HeaderStr := 'Time[ms],' + Header1 + '$R';
bStatus := FileWriteLine (FileID, HeaderStr);
IF (bStatus = false) THEN
    FileClose (FileID);
   FileID := 0;
    RETURN;
END_IF;
```

## 4.11 FilterOrder1

PLCopen

Function block - first order filter

### 4.11.1 Inputs

| | |
|---|---|
| XIN : REAL | Input analog value |
| GAIN : REAL | Transformation gain |

### 4.11.2 Outputs

| | |
|---|---|
| XOUT : REAL | Output signal |

### 4.11.3 Remarks

The operation performed is:

```
Output = (Input x Gain) + (OutputPrev * (1-Gain))
```

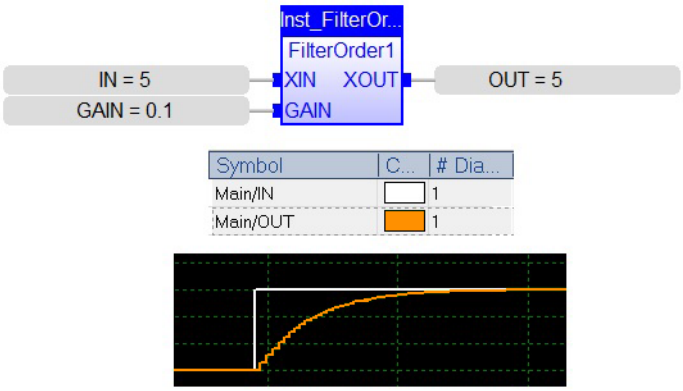
The allowed range for the gain is [0.05 .. 1.0]

### 4.11.4 ST Language

Filt1 is a declared instance of FilterOrder1 function block.

```
Filt1 (rIn, rGain);
Signal := Fillt1.Xout;
```

### 4.11.5 Example




## 4.12 GETSYSINFO

PLCopen

*Function* - Returns system information.

### 4.12.1 Inputs

`INFO : DINT` Identifier of the requested information

### 4.12.2 Outputs

`Q : DINT` Value of the requested information or 0 if error

### 4.12.3 Remarks

The INFO parameter can be one of the following predefined values:

| Value | Definition |
|---|---|
| _SYSINFO_TRIGGER_MICROS | programmed cycle time in micro-seconds |
| _SYSINFO_TRIGGER_MS | programmed cycle time in milliseconds |
| _SYSINFO_CYCLETIME_MICROS | duration of the previous cycle in micro-seconds |
| _SYSINFO_CYCLETIME_MS | duration of the previous cycle in milliseconds |
| _SYSINFO_CYCLEMAX_MICROS | maximum detected cycle time in micro-seconds |
| _SYSINFO_CYCLEMAX_MS | maximum detected cycle time in milliseconds |
| _SYSINFO_CYCLESTAMP_MS | time stamp of the current cycle in milliseconds (OEM dependent) |
| _SYSINFO_CYCLEOVERFLOWS | number of detected cycle time overflows |
| _SYSINFO_CYCLECOUNT | counter of cycles |
| _SYSINFO_APPSTAMP | compiling date stamp of the application |
| _SYSINFO_CODECRC | CRC of the application code. |
| _SYSINFO_DATACRC | CRC of the application symbols. |
| _SYSINFO_FREEHEAP | Available space in memory heap (bytes) |
| _SYSINFO_DBSIZE | Space used in RAM (bytes) |
| _SYSINFO_ELAPSED | Seconds elapsed since startup |
| _SYSINFO_CHANGE_CYCLE | Indicates a cycle just after an On Line Change |
| _SYSINFO_WARMSTART | Non zero if RETAIN variables were loaded at the last start |

| Value | Definition |
|---|---|
| _SYSINFO_NBLOCKED | Number of locked variables |
| _SYSINFO_NBBREAKPOINTS | Number of installed breakpoints |
| _SYSINFO_BIGENDIAN | Non zero if the runtime processor is big endian |
| _SYSINFO_DEMOAPP | Non zero if the application was compiled in DEMO mode |

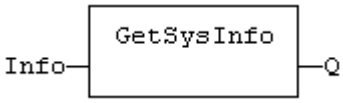
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

In IL, the input must be loaded in the current result before calling the function.

### 4.12.4 ST Language

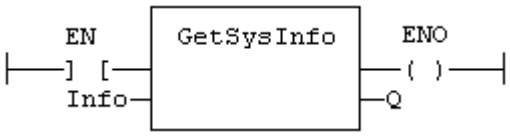
```
Q := GETSYSINFO (INFO);
```

### 4.12.5 FBD Language




### 4.12.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




### 4.12.7 IL Language:

```
Op1: LD  INFO
     GETSYSINFO
     ST  Q
```

## 4.13 HYSTER PLCopen

*Function Block* - Hysteresis detection.

### 4.13.1 Inputs

`XIN1 : REAL` First input signal
`XIN2 : REAL` Second input signal
`EPS : REAL` Hysterisis

### 4.13.2 Outputs

`Q : BOOL` Detected hysteresis: TRUE if XIN1 becomes greater than XIN2+EPS and is not yet below XIN2-EPS

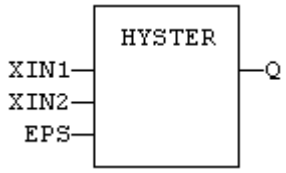
### 4.13.3 Remarks

The hysteresis is detected on the difference of XIN1 and XIN2 signals. In FFLD language, the input rung (EN) is used for enabling the block. The output rung is the Q output.

### 4.13.4 ST Language

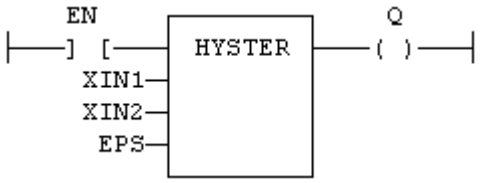
```
(* MyHyst is a declared instance of HYSTER function block *)
MyHyst (XIN1, XIN2, EPS);
Q := MyHyst.Q;
```

### 4.13.5 FBD Language




### 4.13.6 FFLD Language

```
(* The block is not called if EN is FALSE *)
```




### 4.13.7 IL Language:

```
(* MyHyst is a declared instance of HYSTER function block *)
Op1: CAL MyHyst (XIN1, XIN2, EPS)
FFLD  MyHyst.Q
ST  Q
```

**See also**

AVERAGE INTEGRAL DERIVATE LIM_ALRM STACKINT

## 4.14 INTEGRAL PLCopen

*Function Block* - Calculates the integral of a signal with respect to time.

The time unit is seconds. The output signal has the units of the input signal multiplied by seconds. The INTEGRAL block samples the input signal at a maximum rate of 1 millisecond.

### 4.14.1 Inputs

| | |
|---|---|
| `RUN : BOOL` | Run command: TRUE=integrate / FALSE=hold |
| `R1 : BOOL` | Overriding reset |
| `XIN : REAL` | Input signal |
| `X0 : REAL` | Initial value |
| `CYCLE : TIME` | Sampling period (must not be less than the target cycle timing) |

### 4.14.2 Outputs

| | |
|---|---|
| `Q : DINT` | Running mode report: NOT (R1) |
| `XOUT : REAL` | Output signal |

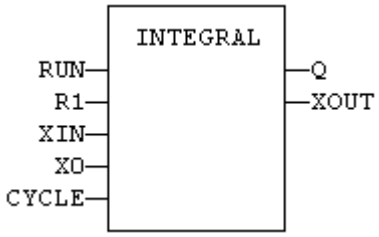
### 4.14.3 Remarks

In FFLD language, the input rung is the RUN command. The output rung is the Q report status.

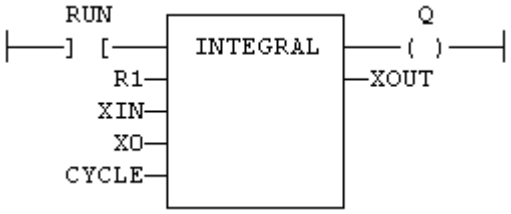
### 4.14.4 ST Language

```
(* MyIntg is a declared instance of INTEGRAL function block *)
MyIntg (RUN, R1, XIN, X0, CYCLE);
Q := MyIntg.Q;
XOUT := MyIntg.XOUT;
```

### 4.14.5 FBD Language




### 4.14.6 FFLD Language




### 4.14.7 IL Language:

```
(* MyIntg is a declared instance of INTEGRAL function block *)
Op1: CAL MyIntg (RUN, R1, XIN, X0, CYCLE)
     FFLD  MyIntg.Q
     ST  Q
     FFLD  MyIntg.XOUT
     ST  XOUT
```

See also

AVERAGE DERIVATE LIM_ALRM HYSTER STACKINT

## 4.15 LIFO PLCopen

*Function block* - Manages a "last in / first out" stack

### 4.15.1 Inputs

| | | |
|---|---|---|
| PUSH | BOOL | Push a new value (on rising edge) |
| POP | BOOL | Pop a new value (on rising edge) |
| RST | BOOL | Reset the list |
| NEXTIN | ANY | Value to be pushed |

| | | |
|---|---|---|
| NEXTOUT | ANY | Value at the top of the stack - *updated after call!* |
| BUFFER | ANY | Array for storing values |

### 4.15.2 Outputs

| | | |
|---|---|---|
| EMPTY | BOOL | TRUE if the stack is empty |
| OFLO | BOOL | TRUE if overflow on a PUSH command |
| COUNT | DINT | Number of values in the stack |
| PREAD | DINT | Index in the buffer of the top of the stack |
| PWRITE | DINT | Index in the buffer of the next push position |

### 4.15.3 Remarks

NEXTIN, NEXTOUT and BUFFER must have the same data type *and cannot be STRING*. The NEXTOUT argument specifies a variable which is filled with the value at the top of the stack after the block is called.

Values are stored in the "BUFFER" array. Data is never shifted or reset. Only read and write pointers and pushed values are updated. The maximum size of the stack is the dimension of the array.

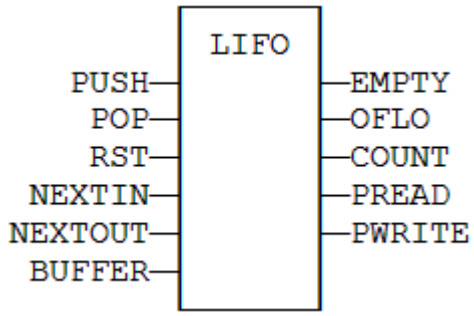
The first time the block is called, it remembers on which array it should work. If you call later the same instance with another BUFFER input, the call is considered as invalid and makes nothing. Outputs reports an empty stack in this case.

In FFLD language, input rung is the PUSH input. The output rung is the EMPTY output.

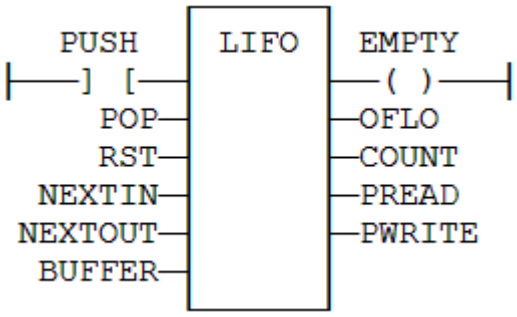
### 4.15.4 ST Language

```
(* MyLIFO is a declared instance of LIFO function block *)
MyLIFO (PUSH, POP, RST, NEXTIN, NEXTOUT, BUFFER);
EMPTY := MyLIFO.EMPTY;
OFLO := MyLIFO.OFLO;
COUNT := MyLIFO.COUNT;
PREAD := MyLIFO.PREAD;
PWRITE := MyLIFO.PWRITE;
```

### 4.15.5 FBD Language




### 4.15.6 FFLD Language




### 4.15.7 IL Language

```
(* MyLIFO is a declared instance of LIFO function block *)
Op1: CAL MyLIFO (PUSH, POP, RST, NEXTIN, NEXTOUT, BUFFER)
FFLD  MyLIFO.EMPTY
ST  EMPTY
FFLD  MyLIFO.OFLO
ST  OFLO
FFLD  MyLIFO.COUNT
ST  COUNT
FFLD  MyLIFO.PREAD
ST  PREAD
FFLD  MyLIFO.PWRITE
ST  PWRITE
```

**See also**

FIFO

## 4.16 LIM_ALRM PLCopen

*Function Block* - Detects High and Low limits of a signal with hysteresis.

### 4.16.1 Inputs

`H : REAL` Value of the High limit
`X : REAL` Input signal
`L : REAL` Value of the Low limit
`EPS : REAL` Value of the hysteresis

### 4.16.2 Outputs

`QH : BOOL` TRUE if the signal exceeds the High limit
`Q : BOOL` TRUE if the signal exceeds one of the limits (equals to QH OR QL)
`QL : BOOL` TRUE if the signal exceeds the Low limit

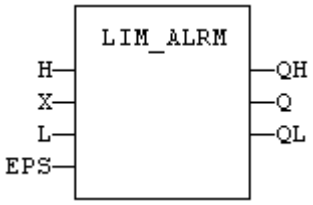
### 4.16.3 Remarks

In FFLD language, the input rung (EN) is used for enabling the block. The output rung is the QH output.

### 4.16.4 ST Language

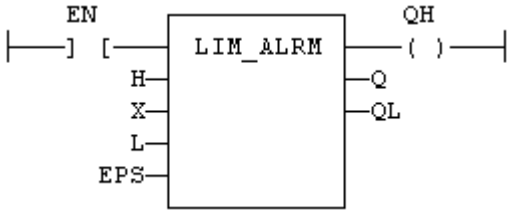
```
(* MyAlarm is a declared instance of LIM_ALRM function block *)
MyAlarm (H, X, L, EPS);
QH := MyAlarm.QH;
Q := MyAlarm.Q;
QL := MyAlarm.QL;
```

### 4.16.5 FBD Language




### 4.16.6 FFLD Language

(* The block is not called if EN is FALSE *)




### 4.16.7 IL Language:

```
(* MyAlarm is a declared instance of LIM_ALRM function block *)
Op1: CAL MyAlarm (H, X, L, EPS)
     FFLD  MyAlarm.QH
     ST  QH
     FFLD  MyAlarm.Q
     ST  Q
     FFLD  MyAlarm.QL
     ST  QL
```

**See also**

ALARM_A ALARM_M

## 4.17 LogFileCSV PLCopen

*Function block* - Generate a log file in CSV format for a list of variables

### 4.17.1 Inputs

| | |
|---|---|
| `LOG : BOOL` | Variables are saved on any rising edge of this input |
| `RST : BOOL` | Reset the contents of the CSV file |
| `LIST : DINT` | ID of the list of variables to log (use VLID function) |
| `PATH : STRING` | Path name of the CSV file (PxMM flash memory, SD card, or Shared Directory) |

### 4.17.2 Outputs

| | |
|---|---|
| `Q : BOOL` | TRUE if the requested operation has been performed without error |
| `ERR : DINT` | Error report for the last requested operation (0 is OK) |

**IMPORTANT**

Calling this function can lead to missing several PLC cycles. Files are opened and closed directly by the target's Operating System. Opening some files may be dangerous for system safety and integrity. The number of open files may be limited by the target system.

**NOTE**

- Opening a file may be unsuccessful (invalid path or file name, too many open files...) Your application has to process such error cases in a safe way.
- File management may be not available on some targets. Please refer to OEM instructions for further details about available features.
- Valid paths for storing files depend on the target implementation. Please refer to OEM instructions for further details about available paths.

### 4.17.3 Remarks

This function enables to log values of a list of variables in a CSV file. On each rising edge of the LOG input, one more line of values is added to the file. There is one column for each variable, as they are defined in the list.

The list of variables is prepared using the KAS IDE or a text editor. Use the VLID function to get the identifier of the list.

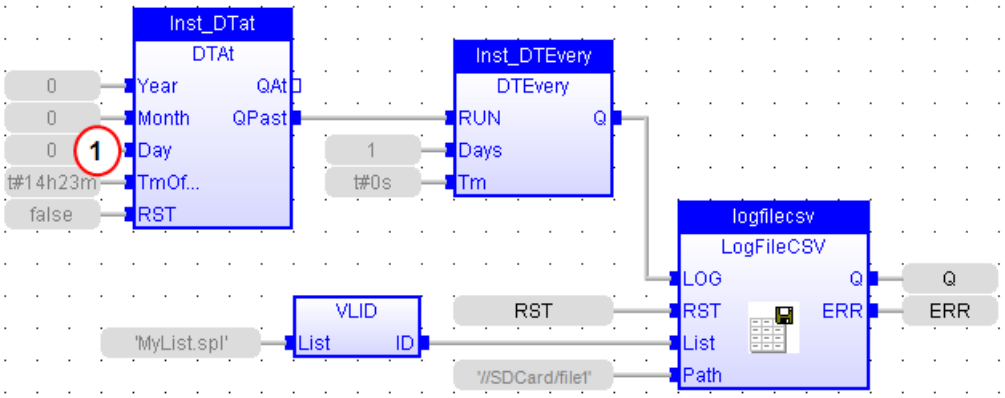
On a rising edge of the RST command, the file is emptied.

When a LOG or RST command is requested, the Q output is set to TRUE if successful.

In case of error, a report is given in the ERR output. Possible error values are:

1 = Cannot reset file on a RST command
2 = Cannot open file for data storing on a LOG command
3 = Embedded lists are not supported by the runtime
4 = Invalid list ID
5 = Error while writing to file

Combined with real time clock management functions, this block provides a very easy way to generate a periodical log file. The following example shows a list and a program that log values everyday at 14h23m (2:23 pm) (see call out 1)

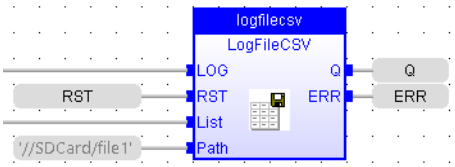



### 4.17.4 ST Language

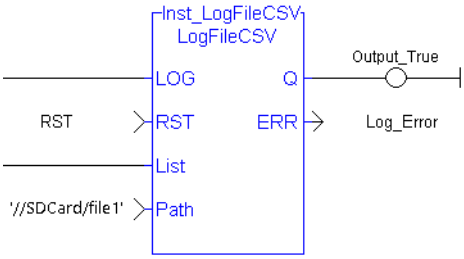
```
(* MyLOG is a declared instance of LogFileCSV function block *)
MyLOG (b_LOG, RST, LIST, PATH);
```

```
Q := MyLOG.Q;
ERR := MyLog.ERR;
```

### 4.17.5 FBD Language




### 4.17.6 FFLD Language



### 4.17.7 IL Language

```
(* MyLOG is a declared instance of LogFileCSV function block *)
Op1: CAL MyLOG (b_LOG, RST, LIST, PATH);
FFLD  MyLOG.Q
ST  Q
FFLD  MyLog.ERR
ST  ERR
```

See also

VLID

## 4.18 PID

*Function Block* - PID loop

### 4.18.1 Inputs

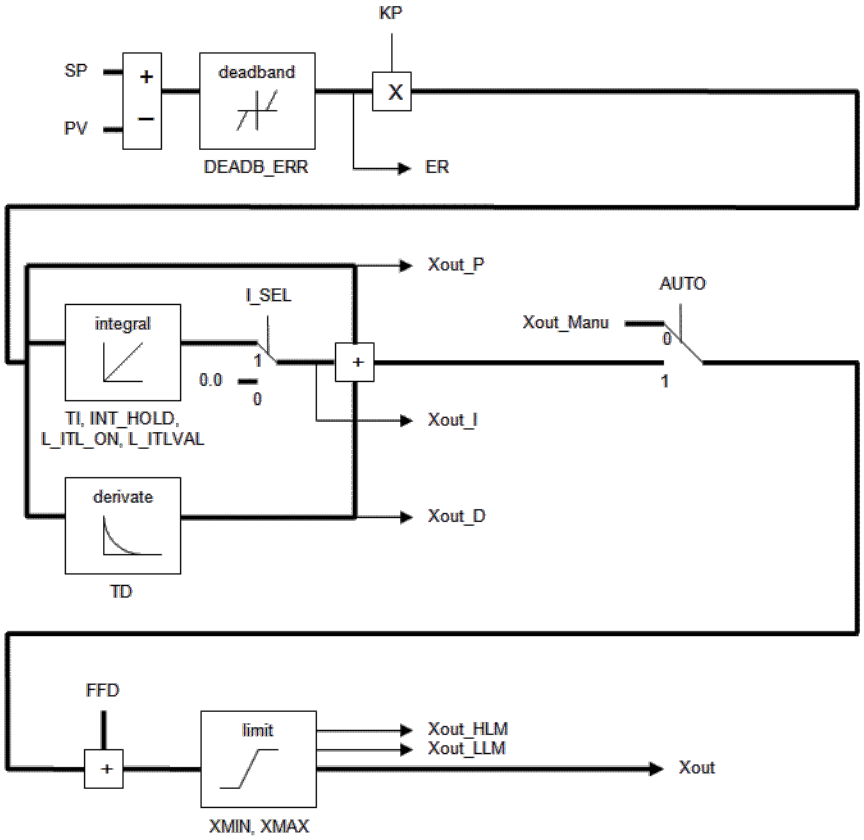
| Input | Type | Description |
|---|---|---|
| **AUTO** | BOOL | TRUE = normal mode , FALSE = manual mode. |
| **PV** | REAL | Process value. |
| **SP** | REAL | Set point. |
| **Xout_Manu** | REAL | Output value in manual mode. |
| **KP** | REAL | Gain. |
| **TI** | REAL | Integration time factor. A value of zero will stop the integrator and freeze Xout I at the previous sample's calculated value. |
| **TD** | REAL | Derivation factor. |
| **TS** | TIME | Sampling period. |
| **XMIN** | REAL | Minimum allowed output value. |
| **XMAX** | REAL | Maximum output value. |
| **I_SEL** | BOOL | If FALSE, the integrated value is ignored. |
| **INT_HOLD** | BOOL | If TRUE, the integrated value is frozen. |
| **I_ITL_ON** | BOOL | If TRUE, the integrated value is reset to I_ITLVAL. |
| **I_ITLVAL** | REAL | Reset value for integration when I_ITL_ON is TRUE. |
| **DEADB_ERR** | REAL | Hysteresis on PV. PV will be considered as unchanged if greater than (PVprev - DEADBAND_W) and less that (PRprev + DEADBAND_W). |
| **FFD** | REAL | Disturbance value on output. |

### 4.18.2 Outputs

| Output | Type | Description |
|---|---|---|
| **Xout** | REAL | Output command value. |
| **ER** | REAL | Last calculated error. |
| **Xout_P** | REAL | Last calculated proportional value. |
| **Xout_I** | REAL | Last calculated integrated value. |

| Output | Type | Description |
|---|---|---|
| **Xout_D** | REAL | Last calculated derivated value. |
| **Xout_HLM** | BOOL | TRUE if the output value is saturated to XMIN. |
| **Xout_LLM** | BOOL | TRUE if the output value is saturated to XMAX. |

### 4.18.3 Diagram



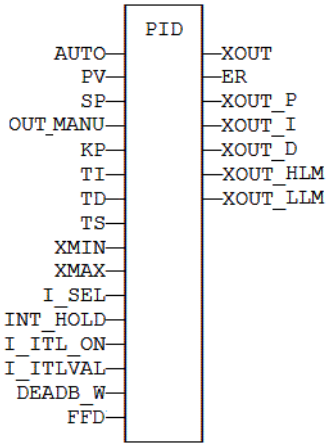

### 4.18.4 Remarks

- It is important for the stability of the control that the TS sampling period is much bigger than the cycle time.
- Output of the PID block always starts with zero. The value will vary per the inputs provided upon further cycle executions.
- The output rung has the same value as the AUTO input, corresponding to the input rung, in the FFLD language.

### 4.18.5 ST Language

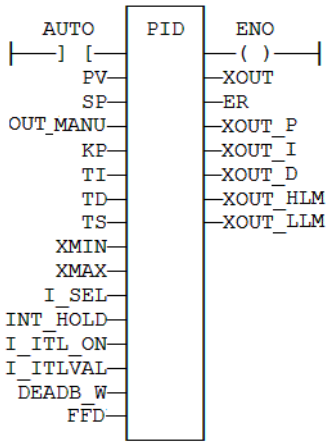
```
(* MyPID is a declared instance of PID function block *)
MyPID (AUTO, PV, SP, XOUT_MANU, KP, TI, TD, TS, XMIN, XMAX, I_SEL, I_ITL_
ON, I_ITLVAL, DEADB_ERR, FFD);
XOUT := MyPID.XOUT;
ER := MyPID.ER;
XOUT_P := MyPID.XOUT_P;
XOUT_I := MyPID.XOUT_I;
XOUT_D := MyPID.XOUT_D;
XOUT_HLM := MyPID.XOUT_HLM;
XOUT_LLM := MyPID.XOUT_LLM;
```

### 4.18.6 FBD Language




### 4.18.7 FFLD Language

(* ENO has the same state as the input rung *)




### 4.18.8 IL Language

```
(* MyPID is a declared instance of PID function block *)
Op1:    CAL MyPID (AUTO, PV, SP, XOUT_MANU, KP, TI, TD, TS, XMIN, XMAX, I_
SEL, I_ITL_ON, I_ITLVAL, DEADB_ERR, FFD)
        FFLD  MyPID.XOUT
        ST  XOUT
        FFLD  MyPID.ER
        ST  ER
        FFLD  MyPID.XOUT_P
        ST  XOUT_P
        FFLD  MyPID.XOUT_I
        ST  XOUT_I
        FFLD  MyPID.XOUT_D
        ST  XOUT_D
        FFLD  MyPID.XOUT_HLM
        ST  XOUT_HLM
        FFLD  MyPID.XOUT_LLM
        ST  XOUT_LLM
```

## 4.19 PWM PLCopen

Function block - generate a PWM signal.

### 4.19.1 Inputs

| | |
|---|---|
| XIN : REAL | Input analog value |
| XinMin : REAL | Minimum input value |
| XinMax : REAL | Maximum input value |
| MinPulse : TIME | Minimum pulse time on output |
| Period : TIME | Period of the output signal |

### 4.19.2 Outputs

| | |
|---|---|
| Q : BOOL | Blinking PWM signal |

### 4.19.3 Remarks

The input value is truncated to [XinMin .. XinMax] interval. XinMax must be greater than XinMin.

The signal is TRUE during:

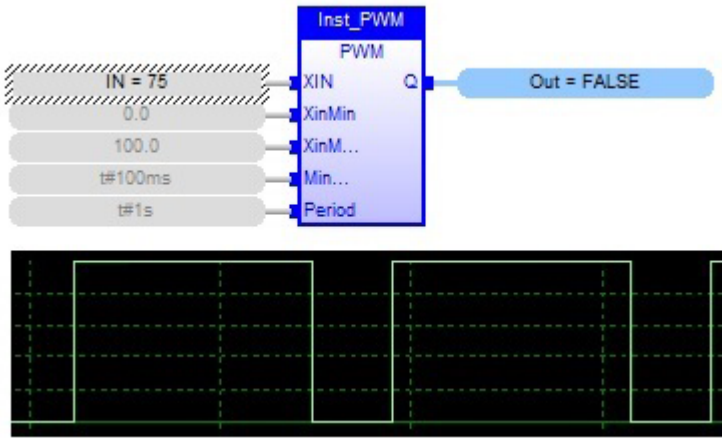
```
(Xin - XinMin) * Period / (XinMax - XinMin)
```

### 4.19.4 ST Language

PWM1 is a declared instance of PWM function block.

```
PWM1 (rIn, rInMin, rInMax, tMinPulse, tPeriod);
Signal := PWM1.Q;
```

### 4.19.5 Example




## 4.20 RAMP PLCopen

*Function block* - Limit the ascendance or descendance of a signal

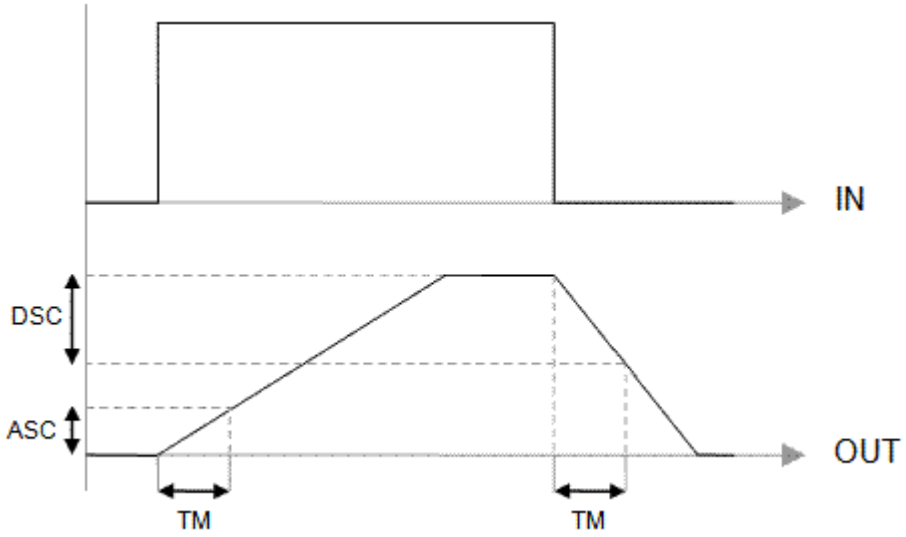
### 4.20.1 Inputs

```
IN : REAL      Input signal
ASC : REAL     Maximum ascendance during time base
DSC : REAL     Maximum descendance during time base
TM : TIME      Time base
RST : BOOL     Reset
```

### 4.20.2 Outputs

```
OUT : REAL     Ramp signal
```

### 4.20.3 Time diagram




### 4.20.4 Remarks

Parameters are not updated constantly. They are taken into account when only:

- the first time the block is called
- when the reset input (RST) is TRUE

In these two situations, the output is set to the value of IN input.

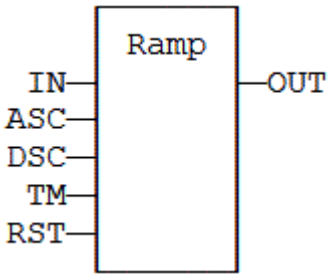
ASC and DSC give the maximum ascendant and descendant growth during the TB time base. Both must be expressed as positive numbers.

In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

### 4.20.5 ST Language

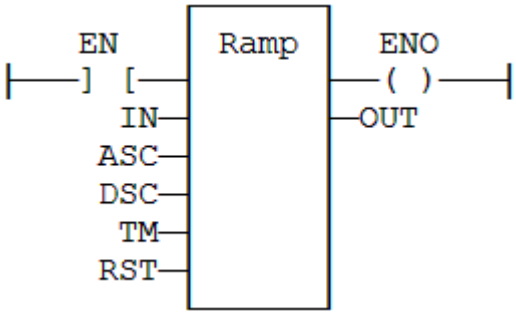
```
(* MyRamp is a declared instance of RAMP function block *)
MyRamp (IN, ASC, DSC, TM, RST);
OUT := MyBlinker.OUT;
```

### 4.20.6 FBD Language




### 4.20.7 FFLD Language

(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)




### 4.20.8 IL Language

```
(* MyRamp is a declared instance of RAMP function block *)
Op1: CAL
MyRamp (IN, ASC, DSC, TM, RST)
FFLD  MyBlinker.OUT
ST  OUT
```

## 4.21 Real Time Clock Management Functions

The following functions read the real time clock of the target system:

| | |
|---|---|
| **DTCurDate** | Get current date stamp |
| **DTCurTime** | Get current time stamp |
| **DTDay** | Get day from date stamp |
| **DTMonth** | Get month from date stamp |
| **DTYear** | Get year from date stamp |
| **DTSec** | Get seconds from time stamp |
| **DTMin** | Get minutes from time stamp |
| **DTHour** | Get hours from time stamp |
| **DTMs** | Get milliseconds from time stamp |

The following functions format the current date/time to a string:

| | |
|---|---|
| DAY_TIME | With predefined format |
| DTFORMAT | With custom format |

The following functions are used for triggering operations:

| | |
|---|---|
| DTAt | Pulse signal at the given date/time |
| DTEvery | Pulse signal with long period |

**IMPORTANT**

- The real-time clock may not be available on all controller hardware models. Please consult the controller hardware specifications for real-time clock availability.
- The date and time are reset to Jan 1, 1970 00:00:00 when the PCMM or AKD PDMM is powered-on, The elapsed time from device power-on can be determined from the Real Time Clock functions.

### DAY_TIME: get current date or time

```
Q := DAY_TIME (SEL);
```

`SEL : DINT` specifies the wished information (see below)
`Q : STRING` wished information formatted on a string

Possible values of SEL input

`1` current time - format: 'HH:MM:SS'
`2` day of the week
`0 (default)` current date - format: 'YYYY/MM/DD'

### DTCURDATE: get current date stamp

```
Q := DTCurDate ();
```

`Q : DINT` numerical stamp representing the current date

### DTCURTIME: get current time stamp

```
Q := DTCurTime ();
```

`Q : DINT` numerical stamp representing the current time of the day

### DTYEAR: extract the year from a date stamp

```
Q := DTYear (iDate);
```

`IDATE : DINT` numerical stamp representing a date. This is output of DTCURDATE.
`Q : DINT` year of the date (ex: 2004)

### DTMONTH: extract the month from a date stamp

```
Q := DTMonth (iDate);
```

`IDATE : DINT` numerical stamp representing a date. This is output of DTCURDATE.
`Q : DINT` month of the date (1..12)

### DTDAY: extract the day of the month from a date stamp

```
Q := DTDay (iDate);
```

`IDATE : DINT` numerical stamp representing a date. This is output of DTCURDATE.
`Q : DINT` day of the month of the date (1..31)

### DTHOUR: extract the hours from a time stamp

```
Q := DTHour (iTime);
```

`ITIME : DINT` numerical stamp representing a time. This is output of DTCURDATE.
`Q : DINT` Hours of the time (0..23)

### DTMIN: extract the minutes from a time stamp

```
Q := DTMin (iTime);
```

`ITIME : DINT` numerical stamp representing a time. This is output of DTCURDATE.
`Q : DINT` Minutes of the time (0..59)

### DTSEC: extract the seconds from a time stamp

```
Q := DTSec (iTime);
```

`ITIME : DINT` numerical stamp representing a time. This is output of DTCURDATE.
`Q : DINT` Seconds of the time (0..59)

**DTMS: extract the milliseconds from a time stamp**

```
Q := DTMs (iTime);
```

`ITIME : DINT` numerical stamp representing a time. This is output of DTCURDATE.
`Q : DINT` Milliseconds of the time (0..999)

## 4.21.1 DAY_TIME PLCopen

*Function* - Format the current date/time to a string.

### 4.21.1.1 Inputs

`SEL : DINT` Format selector

### 4.21.1.2 Outputs

`Q : STRING` String containing formatted date or time

**IMPORTANT**

The real-time clock may not be available on all controller hardware models. Please consult the controller hardware specifications for real-time clock availability.

### 4.21.1.3 Remarks

Possible values of the SEL input are:

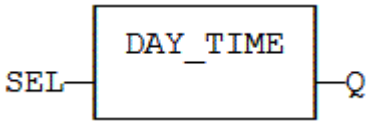
`1` current time - format: 'HH:MM:SS'
`2` day of the week
`0 (default)` current date - format: 'YYYY/MM/DD'

In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

### 4.21.1.4 ST Language

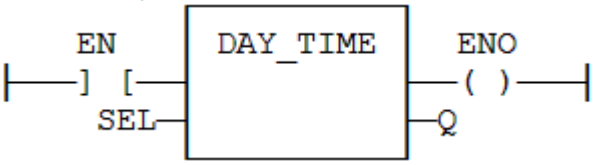
Q := DAY_TIME (SEL);

### 4.21.1.5 FBD Language




### 4.21.1.6 FFLD Language

(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)




### 4.21.1.7 IL Language

Op1: LD SEL
DAY_TIME
ST Q

**See also**

DTFORMAT

## 4.21.2 DTFORMAT PLCopen

*Function* - Format the current date/time to a string with a custom format.

### 4.21.2.1 Inputs

`FMT: STRING` Format string

### 4.21.2.2 Outputs

`Q : STRING` String containing formatted date or time

**IMPORTANT**

The real-time clock may not be available on all controller hardware models. Please consult the controller hardware specifications for real-time clock availability.

### 4.21.2.3 Remarks

The format string may contain any character. Some special markers beginning with the '%' character indicates a date/time information:

**%Y** Year including century (e.g. 2006)
**%y** Year without century (e.g. 06)
**%m** Month (1..12)
**%d** Day of the month (1..31)
**%H** Hours (0..23)
**%M** Minutes (0..59)
**%S** Seconds (0..59)

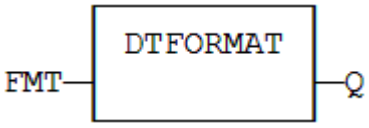
**Example**

```
(* let's say we are at July 04th 2006, 18:45:20 *)
   Q := DTFORMAT ('Today is %Y/%m/%d - %H:%M:%S');
(* Q is 'Today is 2006/07/04 - 18:45:20 *)
```

### 4.21.2.4 ST Language

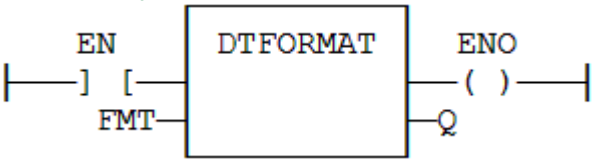
```
Q := DTFORMAT (FMT);
```

### 4.21.2.5 FBD Language




### 4.21.2.6 FFLD Language

```
(* The function is executed only if EN is TRUE *)
(* ENO keeps the same value as EN *)
```




#### 4.21.2.7 IL Language

```
Op1: LD FMT
DTFORMAT
ST Q
```

**See also**

DAY_TIME

### 4.21.3 DTAT PLCopen

*Function Block* - Generate a pulse at given date and time

#### 4.21.3.1 Inputs

| | |
|---|---|
| `YEAR : DINT` | Wished year (e.g. 2006) |
| `MONTH : DINT` | Wished month (1 = January) |
| `DAY : DINT` | Wished day (1 to 31) |
| `TMOFDAY : TIME` | Wished time |
| `RST : BOOL` | Reset command |

#### 4.21.3.2 Outputs

| | |
|---|---|
| `QAT : BOOL` | Pulse signal |
| `QPAST : BOOL` | True if elapsed |

**IMPORTANT**

The real-time clock may not be available on all controller hardware models. Please consult the controller hardware specifications for real-time clock availability.

#### 4.21.3.3 Remarks

Parameters are not updated constantly. They are taken into account when only:

- the first time the block is called
- when the reset input (RST) is TRUE

In these two situations, the outputs are reset to FALSE.

The first time the block is called with RST=FALSE and the specified date/stamp is passed, the output QPAST is set to TRUE, and the output QAT is set to TRUE for one cycle only (pulse signal).

Highest units are ignored if set to 0. For instance, if arguments are "year=0, month=0, day = 3, tmofday=t#10h" then the block will trigger on the next 3rd day of the month at 10h.

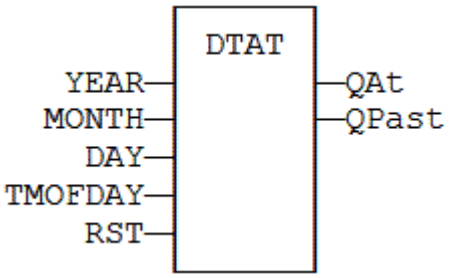
In FFLD language, the block is activated only if the input rung is TRUE..

#### 4.21.3.4 ST Language

```
(* MyDTAT is a declared instance of DTAT function block *)
MyDTAT (YEAR, MONTH, DAY, TMOFDAY, RST);
```

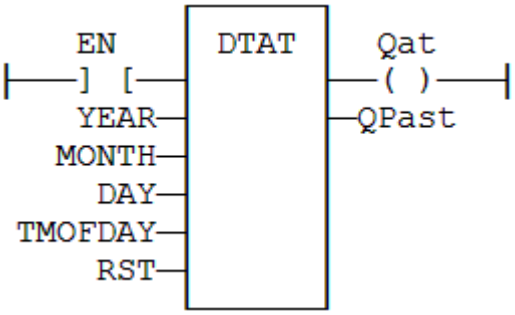
```
QAT := MyDTAT.QAT;
QPAST := MyDTATA.QPAST;
```

#### 4.21.3.5 FBD Language



#### 4.21.3.6 FFLD Language

```
(* Called only if EN is TRUE *)
```




#### 4.21.3.7 IL Language:

```
(* MyDTAT is a declared instance of DTAT function block *)
Op1: CAL MyDTAT (YEAR, MONTH, DAY, TMOFDAY, RST)
FFLD  MyDTAT.QAT
ST  QAT
FFLD  MyDTATA.QPAST
ST  QPAST
```

**See also**

DTEVERY Real time clock functions

### 4.21.4 DTEVERY PLCopen

*Function Block* - Generate a pulse signal with long period

#### 4.21.4.1 Inputs

| | |
|---|---|
| `RUN : DINT` | Enabling command |
| `DAYS : DINT` | Period : number of days |
| `TM : TIME` | Rest of the period (if not a multiple of 24h) |

#### 4.21.4.2 Outputs

| | |
|---|---|
| `Q : BOOL` | Pulse signal |

#### 4.21.4.3 Remarks

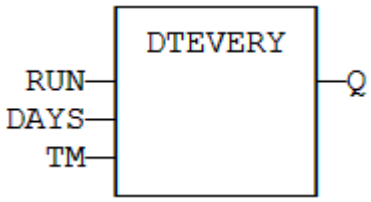
This block provides a pulse signal with a period of more than 24h. The period is expressed as:
DAYS * 24h + TM

For instance, specifying DAYS=1 and TM=6h means a period of 30 hours.

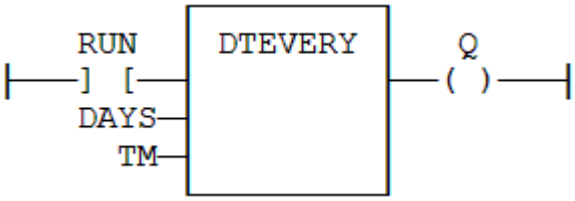
#### 4.21.4.4 ST Language

```
(* MyDTEVERY is a declared instance of DTEVERY function block *)
MyDTEVERY (RUN DAYS, TM);
Q := MyDTEVERY.Q;
```

#### 4.21.4.5 FBD Language




#### 4.21.4.6 FFLD Language




#### 4.21.4.7 IL Language:

```
(* MyDTEVERY is a declared instance of DTEVERY function block *)
Op1: CAL MyDTEVERY (RUN DAYS, TM)
    FFLD  MyDTEVERY.Q
    ST  Q
```

**See also**

DTAT Real time clock functions

## 4.22 SerializeIn PLCopen

### 4.22.1 Description

Extract the value of a variable from a binary frame. This function is commonly used for extracting data from a communication frame in binary format.

In LD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

**TIP**

This function is not available in IL language.

The FRAME input must fit the input position and data size. If the value cannot be safely extracted, the function returns 0. The DATA input must be directly connected to a variable, and cannot be a constant or complex expression. This variable will be forced with the extracted value.

The function extracts the following number of bytes from the source frame:

| | |
|---|---|
| **1 byte** | BOOL, SINT, USINT and BYTE variables |
| **2 bytes** | INT, UINT and WORD variables |
| **4 bytes** | DINT, UDINT, DWORD and REAL variables |
| **8 bytes** | LINT and LREAL variables |

**IMPORTANT**

The function cannot be used to serialize STRING variables.

The function returns the position in the source frame, after the extracted data. Thus the return value can be used as a position for the next serialization.

### 4.22.2 Arguments

#### 4.22.2.1 Input

| | | |
|---|---|---|
| En | Description | Execute the function |
| | Data type | BOOL |
| | Range | [0,1] |
| | Unit | N/A |
| | Default | — |
| Frame[] | Description | Source buffer - must be an array. |
| | Data type | USINT |
| | Range | [0,+65535] |
| | Unit | N/A |
| | Default | N/A |
| Data | Description | Destination variable to be copied |
| | Data type | any except STRING |
| | Range | — |
| | Unit | N/A |
| | Default | — |
| Pos | Description | Position in the source buffer |
| | Data type | DINT |
| | Range | [0,+65535] |
| | Unit | N/A |
| | Default | N/A |
| BigEndian | Description | TRUE if the frame is encoded with Big Endian format. |
| | Data type | BOOL |
| | Range | ? |
| | Unit | ? |
| | Default | ? |

#### 4.22.2.2 Output

| | | |
|---|---|---|
| OK | Description | Returns true when the function successfully executes. See Function - General rules. |
| | Data type | BOOL |

| | | |
|---|---|---|
| | **Unit** | N/A |
| **NextPos** | **Description** | Position in the source buffer after the extracted data. 0 in case or error (invalid position / buffer size). |
| | **Data type** | DINT |
| | **Unit** | N/A |

### 4.22.3 Examples

#### 4.22.3.1 Structured Text

```
NextPos := SerializeIn(Frame[](*USINT*), @Data(*ANY*), Pos(*DINT*),
BigEndian(*BOOL*)); //Read the position
```

## 4.23 SerializeOut PLCopen

### 4.23.1 Description

This function copies the value of a variable to a binary frame. This function is commonly used for building a communication frame in binary format.

In LD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

**NOTE**

This function is not available in IL language.

The FRAME input must be an array large enough to receive the data. If the data cannot be safely copied to the destination buffer, the function returns 0.

The function copies the following number of bytes to the destination frame:

| | |
|---|---|
| **1 byte** | BOOL, SINT, USINT and BYTE variables |
| **2 bytes** | INT, UINT and WORD variables |
| **4 bytes** | DINT, UDINT, DWORD and REAL variables |
| **8 bytes** | LINT and LREAL variables |

**IMPORTANT**

The function cannot be used to serialize STRING variables.

The function returns the position in the destination frame, after the copied data. Thus the return value can be used as a position for the next serialization.

### 4.23.2 Arguments

#### 4.23.2.1 Input

| | | |
|---|---|---|
| **En** | **Description** | Execute the function |
| | **Data type** | BOOL |
| | **Range** | [0,1] |

| | | |
|---|---|---|
| | **Unit** | N/A |
| | **Default** | — |
| **Frame[]** | **Description** | Destination buffer - must be an array. |
| | **Data type** | USINT |
| | **Range** | [0,+65535] |
| | **Unit** | N/A |
| | **Default** | — |
| **Data** | **Description** | Source variable to be copied |
| | **Data type** | any except STRING |
| | **Range** | — |
| | **Unit** | N/A |
| | **Default** | — |
| **Pos** | **Description** | Position in the destination buffer |
| | **Data type** | DINT |
| | **Range** | [0,+65535] |
| | **Unit** | N/A |
| | **Default** | — |
| **BigEndian** | **Description** | TRUE if the frame is encoded with Big Endian format. |
| | **Data type** | BOOL |
| | **Range** | [0,1] |
| | **Unit** | N/A |
| | **Default** | — |

#### 4.23.2.2 Output

| | | |
|---|---|---|
| **OK** | **Description** | Returns true when the function successfully executes. See Function - General rules. |
| | **Data type** | BOOL |
| | **Unit** | N/A |
| **NextPos** | **Description** | Position in the destination buffer after the copied data. 0 in case or error (invalid position / buffer size). |
| | **Data type** | DINT |
| | **Unit** | N/A |

### 4.23.3 Examples

#### 4.23.3.1 Structured Text

```
NextPos := SerializeOut(Frame[](*USINT*), Data(*ANY*), Pos(*DINT*),
BigEndian(*BOOL*)); //Read the position
```

## 4.24 SigID PLCopen

*Function* - Get the identifier of a "Signal" resource

### 4.24.1 Inputs

| | |
|---|---|
| `SIGNAL : STRING` | Name of the signal resource - *must be a constant value!* |
| `COL : STRING` | Name of the column within the signal resource - *must be a constant value!* |

### 4.24.2 Outputs

| | |
|---|---|
| `ID : DINT` | ID of the signal - to be passed to other blocks |

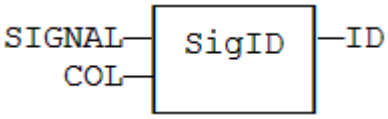
### 4.24.3 Remarks

Some blocks have arguments that refer to a "signal" resource. For all these blocks, the signal argument is materialized by a numerical identifier. This function enables you to get the identifier of a signal defined as a resource.

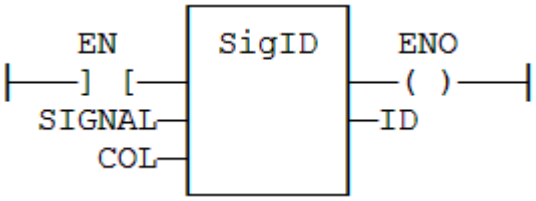
### 4.24.4 ST Language

ID := SigID ('MySignal', 'FirstColumn');

### 4.24.5 FBD Language




### 4.24.6 FFLD Language




### 4.24.7 IL Language

```
Op1: LD   'MySignal'
     SigID 'FirstColumn'
     ST   ID
```

**See also**

SigPlay SigScale

## 4.25 SigPlay PLCopen

*Function block* - Generate a signal defined in a resource

### 4.25.1 Inputs

| | |
|---|---|
| `IN : BOOL` | Triggering command |
| `ID : DINT` | ID of the signal resource, provided by SigID function |
| `RST : BOOL` | Reset command |
| `TM : TIME` | Minimum time in between two changes of the output |

### 4.25.2 Outputs

| | |
|---|---|
| `Q : BOOL` | TRUE when the signal is finished |
| `OUT : REAL` | Generated signal |
| `ET : TIME` | Elapsed time |

### 4.25.3 Remarks

The "ID" argument is the identifier of the "signal" resource. Use the SigID function to get this value.

The "IN" argument is used as a "Play / Pause" command to play the signal. The signal is not reset to the beginning when IN becomes FALSE. Instead, use the "RST" input that resets the signal and forces the OUT output to 0.

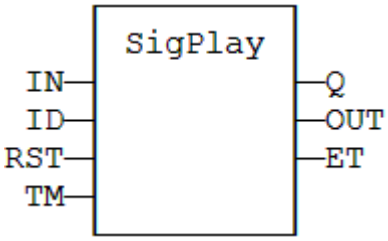
The "TM" input specifies the minimum amount of time in between two changes of the output signal. This parameter is ignored if less than the cycle scan time.

This function block includes its own timer. Alternatively, you can use the SigScale function if you want to trigger the signal using a specific timer.

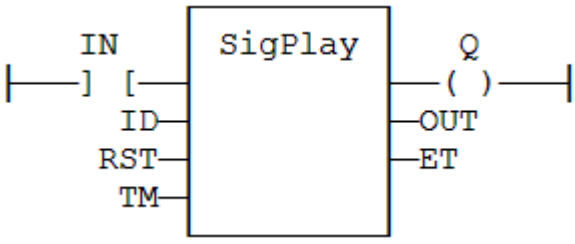
### 4.25.4 ST Language

```
Q := SigScale (ID, IN);
```

### 4.25.5 FBD Language




### 4.25.6 FFLD Language




### 4.25.7 IL Language

Op1: FFLD IN
SigScale ID
ST Q

**See also**

SigScale SigID

## 4.26 SigScale PLCopen

*Function* - Get a point from a "Signal" resource

### 4.26.1 Inputs

| | |
|---|---|
| `ID : DINT` | ID of the signal resource, provided by SigID function |
| `IN : TIME` | Time (X) coordinate of the wished point within the signal resource |

### 4.26.2 Outputs

| | |
|---|---|
| `Q : REAL` | Value (Y) coordinate of the point in the signal |

### 4.26.3 Remarks

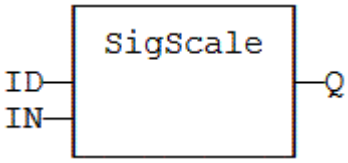
The "ID" argument is the identifier of the "signal" resource. Use the SigID function to get this value.

This function converts a time value to a analog value such as defined in the signal resource. This function can be used instead of SigPlay function block if you want to trigger the signal using a specific timer.

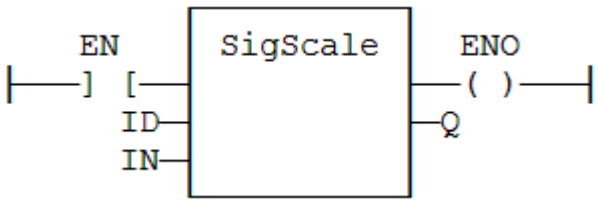
### 4.26.4 ST Language

Q := SigScale (ID, IN);

### 4.26.5 FBD Language




### 4.26.6 FFLD Language




### 4.26.7 IL Language

Op1: LD IN
SigScale ID
ST Q

**See also**

SigPlay SigID

## 4.27 STACKINT PLCopen

*Function Block* - Manages a stack of DINT integers.

### 4.27.1 Inputs

| | |
|---|---|
| `PUSH : BOOL` | Command: when changing from FALSE to TRUE, the value of IN is pushed on the stack |
| `POP : BOOL` | Pop command: when changing from FALSE to TRUE, deletes the top of the stack |
| `R1 : BOOL` | Reset command: if TRUE, the stack is emptied and its size is set to N |
| `IN : DINT` | Value to be pushed on a rising pulse of PUSH |
| `N : DINT` | maximum stack size - cannot exceed 128 |

### 4.27.2 Outputs

| | |
|---|---|
| `EMPTY : BOOL` | TRUE if the stack is empty |
| `OFLO : BOOL` | TRUE if the stack is full |
| `OUT : DINT` | value at the top of the stack |

### 4.27.3 Remarks

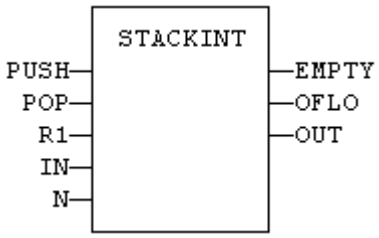
Push and pop operations are performed on rising pulse of PUSH and POP inputs. In FFLD language, the input rung is the PUSH command. The output rung is the EMPTY output.

The specified size (N) is taken into account only when the R1 (reset) input is TRUE.

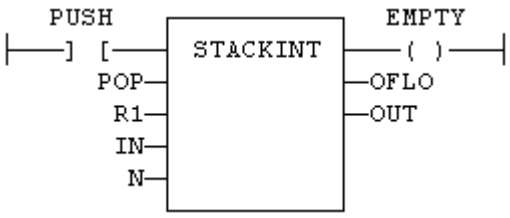
### 4.27.4 ST Language

```
(* MyStack is a declared instance of STACKINT function block *)
MyStack (PUSH, POP, R1, IN, N);
EMPTY := MyStack.EMPTY;
OFLO := MyStack.OFLO;
OUT := MyStack.OUT;
```

### 4.27.5 FBD Language




### 4.27.6 FFLD Language



### 4.27.7 IL Language

```
(* MyStack is a declared instance of STACKINT function block *)
Op1: CAL MyStack (PUSH, POP, R1, IN, N)
    FFLD  MyStack.EMPTY
    ST  EMPTY
    FFLD  MyStack.OFLO
    ST  OFLO
    FFLD  MyStack.OUT
    ST  OUT
```

**See also**

AVERAGE INTEGRAL DERIVATE LIM_ALRM HYSTER

## 4.28 SurfLin PLCopen

*Function block*- Linear interpolation on a surface.

### 4.28.1 Inputs

X : REAL X coordinate of the point to be interpolated.

Y : REAL Y coordinate of the point to be interpolated.

XAxis : REAL[] X coordinates of the known points of the X axis.

YAxis : REAL[] Y coordinates of the known points of the Y axis.

ZVal : REAL[,] Z coordinate of the points defined by the axis.

### 4.28.2 Outputs

Z : REAL Interpolated Z value corresponding to the X,Y input point

OK : BOOL TRUE if successful.

ERR : DINT Error code if failed - 0 if OK.

### 4.28.3 Remarks

This function performs linear surface interpolation in between a list of points defined in XAxis and YAxis single dimension arrays. The output Z value is an interpolation of the Z values of the four rounding points defined in the axis. Z values of defined points are passed in the ZVal matrix (two dimension array).

ZVal dimensions must be understood as: ZVal [ iX , iY ]

Values in X and Y axis must be sorted from the smallest to the biggest. There must be at least two points defined in each axis. ZVal must fit the dimension of XAxis and YAxis arrays. For instance:

XAxis : ARRAY [0..2] of REAL;

YAxis : ARRAY [0.3] of REAL;

ZVal : ARRAY [0..2,0..3] of REAL;

In case the input point is outside the rectangle defined by XAxis and YAxis limits, the Z output is bound to the corresponding value and an error is reported.

The ERR output gives the cause of the error if the function fails:

| Error Code | Meaning |
|---|---|
| 0 | OK |
| 1 | Invalid dimension of input arrays |
| 2 | Invalid points for the X axis |
| 3 | Invalid points for the Y axis |
| 4 | X,Y point is out of the defined axis |

## 4.29 VLID PLCopen

*Function* - Create the identifier (ID) of an embedded list of variables

### 4.29.1 Inputs

`FILE : STRING` Path name of the .TXT list file - *must be a constant value!*

### 4.29.2 Outputs

`ID : DINT` ID of the list - to be passed to other blocks

### 4.29.3 Remarks

This function is used to create an Identifier (ID) or ListID for a list of application variables that are typically stored on the development PC. The list of application variables:

- is a simple ".txt" text file.
- can contain only one variable name per line
- can be only global variables
- can only contain single variables. Items of arrays and structures must be specified one by one.
- the length of the list is not limited by the system.

This function's ID output can be used as an input to LogFileCSV. It defines the application variables whose present value will be recorded each time LogFileCSV is executed.

**IMPORTANT**

List files are read at compiling time and are embedded into the downloaded application code. This implies that a modification performed in the list file after downloading will not be taken into account by the application.

### 4.29.4 ST Language

```
ListID := VLID ('MyFile.txt');
```

### 4.29.5 FBD Language


### 4.29.6 FFLD Language

(* The function is executed only if EN is TRUE *)


### 4.29.7 IL Language

```
Op1: LD   'MyFile.txt'
VLID COL
ST   ListID
```

## About KOLLMORGEN

Kollmorgen is a leading provider of motion systems and components for machine builders. Through world-class knowledge in motion, industry-leading quality and deep expertise in linking and integrating standard and custom products, Kollmorgen delivers breakthrough solutions that are unmatched in performance, reliability and ease-of-use, giving machine builders an irrefutable marketplace advantage.


Join the Kollmorgen Developer Network for product support. Ask the community questions, search the knowledge base for answers, get downloads, and suggest improvements.


**North America**
**KOLLMORGEN**
201 West Rock Road
Radford, VA 24141, USA

| | |
|---|---|
| **Web:** | www.kollmorgen.com |
| **Mail:** | support@kollmorgen.com |
| **Tel.:** | +1 - 540 - 633 - 3545 |
| **Fax:** | +1 - 540 - 639 - 4162 |

**Europe**
**KOLLMORGEN Europe GmbH**
Pempelfurtstr. 1
40880 Ratingen, Germany

| | |
|---|---|
| **Web:** | www.kollmorgen.com |
| **Mail:** | technik@kollmorgen.com |
| **Tel.:** | +49 - 2102 - 9394 - 0 |
| **Fax:** | +49 - 2102 - 9394 - 3155 |

**South America**
**KOLLMORGEN**
Avenida João Paulo Ablas, 2970
Jardim da Glória, Cotia – SP
CEP 06711-250, Brazil

| | |
|---|---|
| **Web:** | www.kollmorgen.com |
| **Mail:** | contato@kollmorgen.com |
| **Tel.:** | +55 11 4615-6300 |

**China and SEA**
**KOLLMORGEN**
Room 302, Building 5, Lihpao Plaza,
88 Shenbin Road, Minhang District,
Shanghai, China.

| | |
|---|---|
| **Web:** | www.kollmorgen.cn |
| **Mail:** | sales.china@kollmorgen.com |
| **Tel.:** | +86 - 400 668 2802 |
| **Fax:** | +86 - 21 6248 5367 |