

# Kollmorgen Automation Suite

## KAS Reference Manual - PLC Library



**Document Edition: L, December 2018**

Valid for KAS Software Revision 3.01

Part Number: 959717



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

**KOLLMORGEN**<sup>®</sup>

*Because Motion Matters™*

## Trademarks and Copyrights

### Copyrights

Copyright © 2009-2018 Kollmorgen

Information in this document is subject to change without notice. The software package described in this document is furnished under a license agreement. The software package may be used or copied only in accordance with the terms of the license agreement.

This document is the intellectual property of Kollmorgen and contains proprietary and confidential information. The reproduction, modification, translation or disclosure to third parties of this document (in whole or in part) is strictly prohibited without the prior written permission of Kollmorgen.

### 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:

- [Qwt](#) project (distributed under the [terms](#) of the GNU Lesser General Public License - see also [GPL terms](#))
- [Zlib](#) software library
- [curl](#) software library
- [Mongoose](#) software (distributed under the MIT License - [see terms](#))
- [JsonCpp](#) software (distributed under the MIT License – [see terms](#))
- [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.

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 L published on 12/7/2018) 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

<b>1 Table of Contents</b> .....	<b>3</b>
<b>2 Programming Languages</b> .....	<b>33</b>
<b>2.1 Sequential Function Chart (SFC)</b> .....	<b>33</b>
2.1.1 SFC Execution at Runtime .....	33
2.1.2 Hierarchy of SFC programs .....	34
<b>2.2 Free Form Ladder Diagram (FFLD)</b> .....	<b>35</b>
2.2.1 Use of the "EN" input and the "ENO" output for blocks .....	35
2.2.2 Contacts and coils .....	36
2.2.2.1 FFLD Contacts .....	36
2.2.2.2 FFLD Coils .....	38
<b>3 PLC Standard Libraries</b> .....	<b>40</b>
<b>3.1 Basic Operations</b> .....	<b>41</b>
3.1.1 := .....	42
3.1.1.1 Inputs .....	42
3.1.1.2 Outputs .....	42
3.1.1.3 Remarks .....	42
3.1.1.4 ST Language .....	42
3.1.1.5 FBD Language .....	42
3.1.1.6 FFLD Language .....	42
3.1.1.7 IL Language: .....	42
3.1.2 Access to bits of an integer .....	43
3.1.3 Differences Between Functions and Function Blocks .....	44
3.1.4 Calling a sub-program .....	45
3.1.4.1 ST Language .....	45
3.1.4.2 FBD and FFLD Languages .....	45
3.1.4.3 IL Language .....	45
3.1.5 CASE OF ELSE END_CASE .....	46
3.1.5.1 Syntax .....	46
3.1.5.2 Remarks .....	46
3.1.5.3 ST Language .....	46
3.1.5.4 FBD Language .....	46
3.1.5.5 FFLD Language .....	46
3.1.5.6 IL Language .....	46
3.1.6 COUNTOF .....	47
3.1.6.1 Inputs .....	47
3.1.6.2 Outputs .....	47
3.1.6.3 Remarks .....	47
3.1.6.4 ST Language .....	47
3.1.6.5 FBD Language .....	47
3.1.6.6 FFLD Language .....	47
3.1.6.7 IL Language .....	47
3.1.7 DEC .....	48
3.1.7.1 Inputs .....	48
3.1.7.2 Outputs .....	48

---

3.1.7.3	Remarks	48
3.1.7.4	ST Language	48
3.1.7.5	FBD Language	48
3.1.7.6	FFLD Language	48
3.1.7.7	IL Language	48
3.1.8	EXIT	49
3.1.8.1	Remarks	49
3.1.8.2	ST Language	49
3.1.8.3	FBD Language	49
3.1.8.4	FFLD Language	49
3.1.8.5	IL Language	49
3.1.9	FOR TO BY END_FOR	50
3.1.9.1	Syntax	50
3.1.9.2	Remarks	50
3.1.9.3	ST Language	50
3.1.9.4	FBD Language	50
3.1.9.5	FFLD Language	50
3.1.9.6	IL Language	50
3.1.10	IF THEN ELSE ELSIF END_IF	51
3.1.10.1	Syntax	51
3.1.10.2	Remarks	51
3.1.10.3	ST Language	51
3.1.10.4	FBD Language	51
3.1.10.5	FFLD Language	51
3.1.10.6	IL Language	52
3.1.11	INC	53
3.1.11.1	Inputs	53
3.1.11.2	Outputs	53
3.1.11.3	Remarks	53
3.1.11.4	ST Language	53
3.1.11.5	FBD Language	53
3.1.11.6	FFLD Language	53
3.1.11.7	IL Language	53
3.1.12	MOVEBLOCK	54
3.1.12.1	Inputs	54
3.1.12.2	Outputs	54
3.1.12.3	Remarks	54
3.1.12.4	ST Language	54
3.1.12.5	FBD Language	54
3.1.12.6	FFLD Language	54
3.1.12.7	IL Language	55
3.1.13	NEG -	56
3.1.13.1	Inputs	56
3.1.13.2	Outputs	56
3.1.13.3	Truth table (examples)	56
3.1.13.4	Remarks	56
3.1.13.5	ST Language	56

3.1.13.6 FBD Language .....	56
3.1.13.7 FFLD Language .....	56
3.1.14 ON .....	57
3.1.14.1 Syntax .....	57
3.1.14.2 Remarks .....	57
3.1.14.3 ST Language .....	57
3.1.15 ( ) .....	58
3.1.15.1 Remarks .....	58
3.1.15.2 ST Language .....	58
3.1.15.3 FBD Language .....	58
3.1.15.4 FFLD Language .....	58
3.1.15.5 IL Language .....	58
3.1.16 REPEAT UNTIL END_REPEAT .....	59
3.1.16.1 Syntax .....	59
3.1.16.2 Remarks .....	59
3.1.16.3 ST Language .....	59
3.1.16.4 FBD Language .....	59
3.1.16.5 FFLD Language .....	59
3.1.16.6 IL Language .....	59
3.1.17 RETURN RET RETC RETNC RETCN .....	60
3.1.17.1 Remarks .....	60
3.1.17.2 ST Language .....	60
3.1.17.3 FBD Language .....	60
3.1.17.4 FFLD Language .....	60
3.1.17.5 IL Language .....	61
3.1.18 WAIT / WAIT_TIME .....	62
3.1.18.1 Syntax .....	62
3.1.18.2 Remarks .....	62
3.1.18.3 ST Language .....	62
3.1.19 WHILE DO END_WHILE .....	63
3.1.19.1 Syntax .....	63
3.1.19.2 Remarks .....	63
3.1.19.3 ST Language .....	63
3.1.19.4 FBD Language .....	63
3.1.19.5 FFLD Language .....	63
3.1.19.6 IL Language .....	63
<b>3.2 Boolean operations .....</b>	<b>64</b>
3.2.1 FLIPFLOP .....	65
3.2.1.1 Inputs .....	65
3.2.1.2 Outputs .....	65
3.2.1.3 Remarks .....	65
3.2.1.4 ST Language .....	65
3.2.1.5 FBD Language .....	65
3.2.1.6 FFLD Language .....	65
3.2.1.7 IL Language .....	65
3.2.2 F_TRIG .....	66
3.2.2.1 Inputs .....	66

---

3.2.2.2	Outputs	66
3.2.2.3	Truth table	66
3.2.2.4	Remarks	66
3.2.2.5	ST Language	66
3.2.2.6	FBD Language	66
3.2.2.7	FFLD Language	66
3.2.2.8	IL Language:	67
3.2.3	NOT	68
3.2.3.1	Inputs	68
3.2.3.2	Outputs	68
3.2.3.3	Truth table	68
3.2.3.4	Remarks	68
3.2.3.5	ST Language	68
3.2.3.6	FBD Language	68
3.2.3.7	FFLD Language	68
3.2.3.8	IL Language:	69
3.2.4	QOR	70
3.2.4.1	Inputs	70
3.2.4.2	Outputs	70
3.2.4.3	Remarks	70
3.2.4.4	ST Language	70
3.2.4.5	FBD Language	70
3.2.4.6	FFLD Language	70
3.2.4.7	IL Language	70
3.2.5	R	71
3.2.5.1	Inputs	71
3.2.5.2	Outputs	71
3.2.5.3	Truth table	71
3.2.5.4	Remarks	71
3.2.5.5	ST Language	71
3.2.5.6	FBD Language	71
3.2.5.7	FFLD Language	71
3.2.5.8	IL Language:	71
3.2.6	RS	72
3.2.6.1	Inputs	72
3.2.6.2	Outputs	72
3.2.6.3	Truth table	72
3.2.6.4	Remarks	72
3.2.6.5	ST Language	72
3.2.6.6	FBD Language	72
3.2.6.7	FFLD Language	73
3.2.6.8	IL Language:	73
3.2.7	R_TRIG	74
3.2.7.1	Inputs	74
3.2.7.2	Outputs	74
3.2.7.3	Truth table	74
3.2.7.4	Remarks	74

3.2.7.5	ST Language .....	74
3.2.7.6	FBD Language .....	74
3.2.7.7	FFLD Language .....	75
3.2.7.8	IL Language: .....	75
3.2.8	S .....	76
3.2.8.1	Inputs .....	76
3.2.8.2	Outputs .....	76
3.2.8.3	Truth table .....	76
3.2.8.4	Remarks .....	76
3.2.8.5	ST Language .....	76
3.2.8.6	FBD Language .....	76
3.2.8.7	FFLD Language .....	76
3.2.8.8	IL Language: .....	76
3.2.9	SEMA .....	77
3.2.9.1	Inputs .....	77
3.2.9.2	Outputs .....	77
3.2.9.3	Remarks .....	77
3.2.9.4	ST Language .....	77
3.2.9.5	FBD Language .....	77
3.2.9.6	FFLD Language .....	77
3.2.9.7	IL Language: .....	77
3.2.10	SR .....	78
3.2.10.1	Inputs .....	78
3.2.10.2	Outputs .....	78
3.2.10.3	Truth table .....	78
3.2.10.4	Remarks .....	78
3.2.10.5	ST Language .....	78
3.2.10.6	FBD Language .....	78
3.2.10.7	FFLD Language .....	79
3.2.10.8	IL Language: .....	79
3.2.11	XOR XORN .....	80
3.2.11.1	Inputs .....	80
3.2.11.2	Outputs .....	80
3.2.11.3	Truth table .....	80
3.2.11.4	Remarks .....	80
3.2.11.5	ST Language .....	80
3.2.11.6	FBD Language .....	80
3.2.11.7	FFLD Language .....	80
3.2.11.8	IL Language .....	81
<b>3.3</b>	<b>Arithmetic operations .....</b>	<b>82</b>
3.3.1	+ Addition .....	83
3.3.1.1	Inputs .....	83
3.3.1.2	Outputs .....	83
3.3.1.3	Remarks .....	83
3.3.1.4	ST Language .....	83
3.3.1.5	FBD Language .....	83
3.3.1.6	FFLD Language .....	83

---

3.3.1.7 IL Language .....	84
3.3.2/ Divide .....	85
3.3.2.1 Inputs .....	85
3.3.2.2 Outputs .....	85
3.3.2.3 Remarks .....	85
3.3.2.4 ST Language .....	85
3.3.2.5 FBD Language .....	85
3.3.2.6 FFLD Language .....	85
3.3.2.7 IL Language: .....	85
3.3.3 NEG - .....	86
3.3.3.1 Inputs .....	86
3.3.3.2 Outputs .....	86
3.3.3.3 Truth table (examples) .....	86
3.3.3.4 Remarks .....	86
3.3.3.5 ST Language .....	86
3.3.3.6 FBD Language .....	86
3.3.3.7 FFLD Language .....	86
3.3.4 LIMIT .....	87
3.3.4.1 Inputs .....	87
3.3.4.2 Outputs .....	87
3.3.4.3 Function diagram .....	87
3.3.4.4 Remarks .....	87
3.3.4.5 ST Language .....	87
3.3.4.6 FBD Language .....	87
3.3.4.7 FFLD Language .....	87
3.3.4.8 IL Language: .....	88
3.3.5 MAX .....	89
3.3.5.1 Inputs .....	89
3.3.5.2 Outputs .....	89
3.3.5.3 Remarks .....	89
3.3.5.4 ST Language .....	89
3.3.5.5 FBD Language .....	89
3.3.5.6 FFLD Language .....	89
3.3.5.7 IL Language: .....	89
3.3.6 MIN .....	90
3.3.6.1 Inputs .....	90
3.3.6.2 Outputs .....	90
3.3.6.3 Remarks .....	90
3.3.6.4 ST Language .....	90
3.3.6.5 FBD Language .....	90
3.3.6.6 FFLD Language .....	90
3.3.6.7 IL Language: .....	90
3.3.7 MOD / MODR / MODLR .....	91
3.3.7.1 Remarks .....	91
3.3.7.2 ST Language .....	91
3.3.7.3 FBD Language .....	91
3.3.7.4 FFLD Language .....	91



3.3.7.5 IL Language .....	91
3.3.8 * Multiply .....	92
3.3.8.1 Inputs .....	92
3.3.8.2 Outputs .....	92
3.3.8.3 Remarks .....	92
3.3.8.4 ST Language .....	92
3.3.8.5 FBD Language .....	92
3.3.8.6 FFLD Language .....	92
3.3.8.7 IL Language: .....	92
3.3.9 ODD .....	93
3.3.9.1 Inputs .....	93
3.3.9.2 Outputs .....	93
3.3.9.3 Remarks .....	93
3.3.9.4 ST Language .....	93
3.3.9.5 FBD Language .....	93
3.3.9.6 FFLD Language .....	93
3.3.9.7 IL Language: .....	93
3.3.10 SetWithin .....	94
3.3.10.1 Inputs .....	94
3.3.10.2 Outputs .....	94
3.3.10.3 Truth Table .....	94
3.3.10.4 Remarks .....	94
3.3.11 - Subtraction .....	95
3.3.11.1 Inputs .....	95
3.3.11.2 Outputs .....	95
3.3.11.3 Remarks .....	95
3.3.11.4 ST Language .....	95
3.3.11.5 FBD Language .....	95
3.3.11.6 FFLD Language .....	95
3.3.11.7 IL Language: .....	95
<b>3.4 Comparison Operations .....</b>	<b>96</b>
3.4.1 CMP .....	97
3.4.1.1 Inputs .....	97
3.4.1.2 Outputs .....	97
3.4.1.3 Remarks .....	97
3.4.1.4 ST Language .....	97
3.4.1.5 FBD Language .....	97
3.4.1.6 FFLD Language .....	97
3.4.1.7 IL Language: .....	97
3.4.2 >= GE .....	98
3.4.2.1 Inputs .....	98
3.4.2.2 Outputs .....	98
3.4.2.3 Remarks .....	98
3.4.2.4 ST Language .....	98
3.4.2.5 FBD Language .....	98
3.4.2.6 FFLD Language .....	98
3.4.2.7 IL Language: .....	98

---

3.4.3 > GT	99
3.4.3.1 Inputs	99
3.4.3.2 Outputs	99
3.4.3.3 Remarks	99
3.4.3.4 ST Language	99
3.4.3.5 FBD Language	99
3.4.3.6 FFLD Language	99
3.4.3.7 IL Language:	99
3.4.4 = EQ	100
3.4.4.1 Inputs	100
3.4.4.2 Outputs	100
3.4.4.3 Remarks	100
3.4.4.4 ST Language	100
3.4.4.5 FBD Language	100
3.4.4.6 FFLD Language	100
3.4.4.7 IL Language:	100
3.4.5 <> NE	101
3.4.5.1 Inputs	101
3.4.5.2 Outputs	101
3.4.5.3 Remarks	101
3.4.5.4 ST Language	101
3.4.5.5 FBD Language	101
3.4.5.6 FFLD Language	101
3.4.5.7 IL Language:	101
3.4.6 <= LE	102
3.4.6.1 Inputs	102
3.4.6.2 Outputs	102
3.4.6.3 Remarks	102
3.4.6.4 ST Language	102
3.4.6.5 FBD Language	102
3.4.6.6 FFLD Language	102
3.4.6.7 IL Language:	102
3.4.7 < LT	103
3.4.7.1 Inputs	103
3.4.7.2 Outputs	103
3.4.7.3 Remarks	103
3.4.7.4 ST Language	103
3.4.7.5 FBD Language	103
3.4.7.6 FFLD Language	103
3.4.7.7 IL Language:	103
<b>3.5 Type conversion functions</b>	<b>104</b>
3.5.1 ANY_TO_BOOL	105
3.5.1.1 Inputs	105
3.5.1.2 Outputs	105
3.5.1.3 Remarks	105
3.5.1.4 ST Language	105
3.5.1.5 FBD Language	105

3.5.1.6	FFLD Language .....	105
3.5.1.7	IL Language: .....	105
3.5.1.8	See also .....	105
3.5.2	ANY_TO_DINT / ANY_TO_UDINT .....	106
3.5.2.1	Inputs .....	106
3.5.2.2	Outputs .....	106
3.5.2.3	Remarks .....	106
3.5.2.4	ST Language .....	106
3.5.2.5	FBD Language .....	106
3.5.2.6	FFLD Language .....	106
3.5.2.7	IL Language: .....	106
3.5.2.8	See also .....	106
3.5.3	ANY_TO_INT / ANY_TO_UINT .....	107
3.5.3.1	Inputs .....	107
3.5.3.2	Outputs .....	107
3.5.3.3	Remarks .....	107
3.5.3.4	ST Language .....	107
3.5.3.5	FBD Language .....	107
3.5.3.6	FFLD Language .....	107
3.5.3.7	IL Language: .....	107
3.5.3.8	See also .....	107
3.5.4	ANY_TO_LINT / ANY_TO_ULINT .....	108
3.5.4.1	Inputs .....	108
3.5.4.2	Outputs .....	108
3.5.4.3	Remarks .....	108
3.5.4.4	ST Language .....	108
3.5.4.5	FBD Language .....	108
3.5.4.6	FFLD Language .....	108
3.5.4.7	IL Language: .....	108
3.5.4.8	See also .....	108
3.5.5	ANY_TO_LREAL .....	109
3.5.5.1	Inputs .....	109
3.5.5.2	Outputs .....	109
3.5.5.3	Remarks .....	109
3.5.5.4	ST Language .....	109
3.5.5.5	FBD Language .....	109
3.5.5.6	FFLD Language .....	109
3.5.5.7	IL Language: .....	109
3.5.5.8	See also .....	109
3.5.6	ANY_TO_REAL .....	110
3.5.6.1	Inputs .....	110
3.5.6.2	Outputs .....	110
3.5.6.3	Remarks .....	110
3.5.6.4	ST Language .....	110
3.5.6.5	FBD Language .....	110
3.5.6.6	FFLD Language .....	110
3.5.6.7	IL Language: .....	110

---

3.5.6.8 See also .....	110
3.5.7 ANY_TO_TIME .....	111
3.5.7.1 Inputs .....	111
3.5.7.2 Outputs .....	111
3.5.7.3 Remarks .....	111
3.5.7.4 ST Language .....	111
3.5.7.5 FBD Language .....	111
3.5.7.6 FFLD Language .....	111
3.5.7.7 IL Language: .....	111
3.5.7.8 See also .....	111
3.5.8 ANY_TO_SINT / ANY_TO_USINT .....	112
3.5.8.1 Inputs .....	112
3.5.8.2 Outputs .....	112
3.5.8.3 Remarks .....	112
3.5.8.4 ST Language .....	112
3.5.8.5 FBD Language .....	112
3.5.8.6 FFLD Language .....	112
3.5.8.7 IL Language .....	112
3.5.8.8 See also .....	112
3.5.9 ANY_TO_STRING .....	113
3.5.9.1 Inputs .....	113
3.5.9.2 Outputs .....	113
3.5.9.3 Remarks .....	113
3.5.9.4 ST Language .....	113
3.5.9.5 FBD Language .....	113
3.5.9.6 FFLD Language .....	113
3.5.9.7 IL Language: .....	113
3.5.9.8 See also .....	113
3.5.10 NUM_TO_STRING .....	114
3.5.10.1 Inputs .....	114
3.5.10.2 Outputs .....	114
3.5.10.3 Remarks .....	114
3.5.10.4 Examples .....	114
3.5.11 BCD_TO_BIN .....	115
3.5.11.1 Inputs .....	115
3.5.11.2 Outputs .....	115
3.5.11.4 Remarks .....	115
3.5.11.5 ST Language .....	115
3.5.11.6 FBD Language .....	115
3.5.11.7 FFLD Language .....	115
3.5.11.8 IL Language .....	115
3.5.12 BIN_TO_BCD .....	116
3.5.12.1 Inputs .....	116
3.5.12.2 Outputs .....	116
3.5.12.3 Truth table (examples) .....	116
3.5.12.4 Remarks .....	116
3.5.12.5 ST Language .....	116

3.5.12.6 FBD Language .....	116
3.5.12.7 FFLD Language .....	116
3.5.12.8 IL Language: .....	116
<b>3.6 Selectors .....</b>	<b>117</b>
3.6.1 MUX4 .....	118
3.6.1.1 Inputs .....	118
3.6.1.2 Outputs .....	118
3.6.1.3 Truth table .....	118
3.6.1.4 Remarks .....	118
3.6.1.5 ST Language .....	118
3.6.1.6 FBD Language .....	118
3.6.1.7 FFLD Language .....	118
3.6.1.8 IL Language .....	119
3.6.2 MUX8 .....	120
3.6.2.1 Inputs .....	120
3.6.2.2 Outputs .....	120
3.6.2.3 Truth table .....	120
3.6.2.4 Remarks .....	120
3.6.2.5 ST Language .....	120
3.6.2.6 FBD Language .....	120
3.6.2.7 FFLD Language .....	120
3.6.2.8 IL Language .....	121
3.6.3 SEL .....	122
3.6.3.1 Inputs .....	122
3.6.3.2 Outputs .....	122
3.6.3.3 Truth table .....	122
3.6.3.4 Remarks .....	122
3.6.3.5 ST Language .....	122
3.6.3.6 FBD Language .....	122
3.6.3.7 FFLD Language .....	122
3.6.3.8 IL Language .....	122
<b>3.7 Registers .....</b>	<b>123</b>
3.7.1 AND_MASK .....	124
3.7.1.1 Inputs .....	124
3.7.1.2 Outputs .....	124
3.7.1.3 Remarks .....	124
3.7.1.4 ST Language .....	124
3.7.1.5 FBD Language .....	124
3.7.1.6 FFLD Language .....	124
3.7.1.7 IL Language: .....	124
3.7.2 HIBYTE .....	125
3.7.2.1 Inputs .....	125
3.7.2.2 Outputs .....	125
3.7.2.3 Remarks .....	125
3.7.2.4 ST Language .....	125
3.7.2.5 FBD Language .....	125
3.7.2.6 FFLD Language .....	125

---

3.7.2.7 IL Language: .....	125
3.7.3 LOBYTE .....	126
3.7.3.1 Inputs .....	126
3.7.3.2 Outputs .....	126
3.7.3.3 Remarks .....	126
3.7.3.4 ST Language .....	126
3.7.3.5 FBD Language .....	126
3.7.3.6 FFLD Language .....	126
3.7.3.7 IL Language: .....	126
3.7.4 HIWORD .....	127
3.7.4.1 Inputs .....	127
3.7.4.2 Outputs .....	127
3.7.4.3 Remarks .....	127
3.7.4.4 ST Language .....	127
3.7.4.5 FBD Language .....	127
3.7.4.6 FFLD Language .....	127
3.7.4.7 IL Language: .....	127
3.7.5 LOWORD .....	128
3.7.5.1 Inputs .....	128
3.7.5.2 Outputs .....	128
3.7.5.3 Remarks .....	128
3.7.5.4 ST Language .....	128
3.7.5.5 FBD Language .....	128
3.7.5.6 FFLD Language .....	128
3.7.5.7 IL Language: .....	128
3.7.6 MAKEDWORD .....	129
3.7.6.1 Inputs .....	129
3.7.6.2 Outputs .....	129
3.7.6.3 Remarks .....	129
3.7.6.4 ST Language .....	129
3.7.6.5 FBD Language .....	129
3.7.6.6 FFLD Language .....	129
3.7.6.7 IL Language: .....	129
3.7.7 MAKEWORD .....	130
3.7.7.1 Inputs .....	130
3.7.7.2 Outputs .....	130
3.7.7.3 Remarks .....	130
3.7.7.4 ST Language .....	130
3.7.7.5 FBD Language .....	130
3.7.7.6 FFLD Language .....	130
3.7.7.7 IL Language: .....	130
3.7.8 MBSHIFT .....	131
3.7.8.1 Inputs .....	131
3.7.8.2 Outputs .....	131
3.7.8.3 Remarks .....	131
3.7.8.4 ST Language .....	131
3.7.8.5 FBD Language .....	131

3.7.8.6	FFLD Language .....	131
3.7.8.7	IL Language: .....	131
3.7.9	NOT_MASK .....	132
3.7.9.1	Inputs .....	132
3.7.9.2	Outputs .....	132
3.7.9.3	Remarks .....	132
3.7.9.4	ST Language .....	132
3.7.9.5	FBD Language .....	132
3.7.9.6	FFLD Language .....	132
3.7.9.7	IL Language: .....	132
3.7.10	OR_MASK .....	133
3.7.10.1	Inputs .....	133
3.7.10.2	Outputs .....	133
3.7.10.3	Remarks .....	133
3.7.10.4	ST Language .....	133
3.7.10.5	FBD Language .....	133
3.7.10.6	FFLD Language .....	133
3.7.10.7	IL Language: .....	133
3.7.11	PACK8 .....	134
3.7.11.1	Inputs .....	134
3.7.11.2	Outputs .....	134
3.7.11.3	Remarks .....	134
3.7.11.4	ST Language .....	134
3.7.11.5	FBD Language .....	134
3.7.11.6	FFLD Language .....	135
3.7.11.7	IL Language .....	135
3.7.12	ROL .....	136
3.7.12.1	Inputs .....	136
3.7.12.2	Outputs .....	136
3.7.12.3	Diagram .....	136
3.7.12.4	Remarks .....	136
3.7.12.5	ST Language .....	136
3.7.12.6	FBD Language .....	136
3.7.12.7	FFLD Language .....	136
3.7.12.8	IL Language: .....	136
3.7.13	ROR .....	137
3.7.13.1	Inputs .....	137
3.7.13.2	Outputs .....	137
3.7.13.3	Diagram .....	137
3.7.13.4	Remarks .....	137
3.7.13.5	ST Language .....	137
3.7.13.6	FBD Language .....	137
3.7.13.7	FFLD Language .....	137
3.7.13.8	IL Language: .....	137
3.7.14	RORb/ ROR_SINT / ROR_USINT / ROR_BYTE .....	138
3.7.14.1	Inputs .....	138
3.7.14.2	Outputs .....	138

---

3.7.14.3	Diagram	138
3.7.14.4	Remarks	138
3.7.14.5	ST Language	138
3.7.14.6	FBD Language	138
3.7.14.7	FFLD Language	138
3.7.14.8	IL Language:	138
3.7.14.9	See also	138
3.7.15	RORw / ROR_INT / ROR_UINT / ROR_WORD	139
3.7.15.1	Inputs	139
3.7.15.2	Outputs	139
3.7.15.3	Diagram	139
3.7.15.4	Remarks	139
3.7.15.5	ST Language	139
3.7.15.6	FBD Language	139
3.7.15.7	FFLD Language	139
3.7.15.8	IL Language:	139
3.7.15.9	See also	139
3.7.16	SETBIT	140
3.7.16.1	Inputs	140
3.7.16.2	Outputs	140
3.7.16.3	Remarks	140
3.7.16.4	ST Language	140
3.7.16.5	FBD Language	140
3.7.16.6	FFLD Language	140
3.7.16.7	IL Language	140
3.7.17	SHL	141
3.7.17.1	Inputs	141
3.7.17.2	Outputs	141
3.7.17.3	Diagram	141
3.7.17.4	Remarks	141
3.7.17.5	ST Language	141
3.7.17.6	FBD Language	141
3.7.17.7	FFLD Language	141
3.7.17.8	IL Language:	141
3.7.18	SHR	142
3.7.18.1	Inputs	142
3.7.18.2	Outputs	142
3.7.18.3	Diagram	142
3.7.18.4	Remarks	142
3.7.18.5	ST Language	142
3.7.18.6	FBD Language	142
3.7.18.7	FFLD Language	142
3.7.18.8	IL Language:	142
3.7.19	TESTBIT	143
3.7.19.1	Inputs	143
3.7.19.2	Outputs	143
3.7.19.3	Remarks	143



3.7.19.4	ST Language .....	143
3.7.19.5	FBD Language .....	143
3.7.19.6	FFLD Language .....	143
3.7.19.7	IL Language .....	143
3.7.20	UNPACK8 .....	144
3.7.20.1	Inputs .....	144
3.7.20.2	Outputs .....	144
3.7.20.3	Remarks .....	144
3.7.20.4	ST Language .....	144
3.7.20.5	FBD Language .....	144
3.7.20.6	FFLD Language .....	144
3.7.20.7	IL Language: .....	145
3.7.21	XOR_MASK .....	146
3.7.21.1	Inputs .....	146
3.7.21.2	Outputs .....	146
3.7.21.3	Remarks .....	146
3.7.21.4	ST Language .....	146
3.7.21.5	FBD Language .....	146
3.7.21.6	FFLD Language .....	146
3.7.21.7	IL Language: .....	146
<b>3.8</b>	<b>Counters .....</b>	<b>147</b>
3.8.1	CTD / CTD <sub>r</sub> .....	148
3.8.1.1	Inputs .....	148
3.8.1.2	Outputs .....	148
3.8.1.3	Remarks .....	148
3.8.1.4	ST Language .....	148
3.8.1.5	FBD Language .....	148
3.8.1.6	FFLD Language .....	148
3.8.1.7	IL Language: .....	148
3.8.2	CTU / CTU <sub>r</sub> .....	149
3.8.2.1	Inputs .....	149
3.8.2.2	Outputs .....	149
3.8.2.3	Remarks .....	149
3.8.2.4	ST Language .....	149
3.8.2.5	FBD Language .....	149
3.8.2.6	FFLD Language .....	149
3.8.2.7	IL Language: .....	149
3.8.3	CTUD / CTUD <sub>r</sub> .....	150
3.8.3.1	Inputs .....	150
3.8.3.2	Outputs .....	150
3.8.3.3	Remarks .....	150
3.8.3.4	ST Language .....	150
3.8.3.5	FBD Language .....	150
3.8.3.6	FFLD Language .....	151
3.8.3.7	IL Language: .....	151
<b>3.9</b>	<b>Timers .....</b>	<b>152</b>
3.9.1	BLINK .....	153

---

3.9.1.1	Inputs	153
3.9.1.2	Outputs	153
3.9.1.3	Time diagram	153
3.9.1.4	Remarks	153
3.9.1.5	ST Language	153
3.9.1.6	FBD Language	153
3.9.1.7	FFLD Language	153
3.9.1.8	IL Language	153
3.9.2	BLINKA	154
3.9.2.1	Inputs	154
3.9.2.2	Outputs	154
3.9.2.3	Time diagram	154
3.9.2.4	Remarks	154
3.9.2.5	ST Language	154
3.9.2.6	FBD Language	154
3.9.2.7	FFLD Language	154
3.9.2.8	IL Language:	155
3.9.3	PLS	156
3.9.3.1	Inputs	156
3.9.3.2	Outputs	156
3.9.3.3	Time diagram	156
3.9.3.4	Remarks	156
3.9.3.5	ST Language	156
3.9.3.6	FBD Language	156
3.9.3.7	FFLD Language	156
3.9.3.8	IL Language	157
3.9.4	Sig_Gen	158
3.9.4.1	Inputs	158
3.9.4.2	Outputs	158
3.9.4.3	FFLD Language	158
3.9.5	TMD	159
3.9.5.1	Inputs	159
3.9.5.2	Outputs	159
3.9.5.3	Time diagram	159
3.9.5.4	Remarks	159
3.9.5.5	ST Language	159
3.9.5.6	FBD Language	159
3.9.5.7	FFLD Language	159
3.9.5.8	IL Language	160
3.9.6	TMU / TMUsec	161
3.9.6.1	Inputs	161
3.9.6.2	Outputs	161
3.9.6.3	Time diagram	161
3.9.6.4	Remarks	161
3.9.6.5	ST Language	161
3.9.6.6	FBD Language	161
3.9.6.7	FFLD Language	162

3.9.6.8 IL Language:	162
3.9.7 TOF / TOFR	163
3.9.7.1 Inputs	163
3.9.7.2 Outputs	163
3.9.7.3 Time diagram	163
3.9.7.4 Remarks	163
3.9.7.5 ST Language	163
3.9.7.6 FBD Language	163
3.9.7.7 FFLD Language	163
3.9.7.8 IL Language:	164
3.9.8 TON	165
3.9.8.1 Inputs	165
3.9.8.2 Outputs	165
3.9.8.3 Time diagram	165
3.9.8.4 Remarks	165
3.9.8.5 ST Language	165
3.9.8.6 FBD Language	165
3.9.8.7 FFLD Language	166
3.9.8.8 IL Language:	166
3.9.9 TP / TPR	167
3.9.9.1 Inputs	167
3.9.9.2 Outputs	167
3.9.9.3 Time diagram	167
3.9.9.4 Remarks	167
3.9.9.5 ST Language	167
3.9.9.6 FBD Language	167
3.9.9.7 FFLD Language	167
3.9.9.8 IL Language:	168
<b>3.10 Mathematic operations</b>	<b>169</b>
3.10.1 ABS / ABSL	170
3.10.1.1 Inputs	170
3.10.1.2 Outputs	170
3.10.1.3 Remarks	170
3.10.1.4 ST Language	170
3.10.1.5 FBD Language	170
3.10.1.6 FFLD Language	170
3.10.1.7 IL Language	170
3.10.2 EXPT	171
3.10.2.1 Inputs	171
3.10.2.2 Outputs	171
3.10.2.3 Remarks	171
3.10.2.4 ST Language	171
3.10.2.5 FBD Language	171
3.10.2.6 FFLD Language	171
3.10.2.7 IL Language:	171
3.10.3 EXP / EXPL	172
3.10.3.1 Inputs	172

---

3.10.3.2	Outputs	172
3.10.3.3	Remarks	172
3.10.3.4	ST Language	172
3.10.3.5	FBD Language	172
3.10.3.6	FFLD Language	172
3.10.3.7	IL Language:	172
3.10.4	LOG / LOGL	173
3.10.4.1	Inputs	173
3.10.4.2	Outputs	173
3.10.4.3	Remarks	173
3.10.4.4	ST Language	173
3.10.4.5	FBD Language	173
3.10.4.6	FFLD Language	173
3.10.4.7	IL Language:	173
3.10.5	LN / LNL	174
3.10.5.1	Inputs	174
3.10.5.2	Outputs	174
3.10.5.3	Remarks	174
3.10.5.4	ST Language	174
3.10.5.5	FBD Language	174
3.10.5.6	FFLD Language	174
3.10.5.7	IL Language:	174
3.10.6	POW ** POWL	175
3.10.6.1	Inputs	175
3.10.6.2	Outputs	175
3.10.6.3	Remarks	175
3.10.6.4	ST Language	175
3.10.6.5	FBD Language	175
3.10.6.6	FFLD Language	175
3.10.6.7	IL Language:	175
3.10.7	ROOT	176
3.10.7.1	Inputs	176
3.10.7.2	Outputs	176
3.10.7.3	Remarks	176
3.10.7.4	ST Language	176
3.10.7.5	FBD Language	176
3.10.7.6	FFLD Language	176
3.10.7.7	IL Language:	176
3.10.8	ScaleLin	177
3.10.8.1	Inputs	177
3.10.8.2	Outputs	177
3.10.8.3	Truth table	177
3.10.8.4	Remarks	177
3.10.8.5	ST Language	177
3.10.8.6	FBD Language	177
3.10.8.7	FFLD Language	178
3.10.8.8	IL Language	178

3.10.9 SQRT / SQRTL .....	179
3.10.9.1 Inputs .....	179
3.10.9.2 Outputs .....	179
3.10.9.3 Remarks .....	179
3.10.9.4 ST Language .....	179
3.10.9.5 FBD Language .....	179
3.10.9.6 FFLD Language .....	179
3.10.9.7 IL Language: .....	179
3.10.10 TRUNC / TRUNCL .....	180
3.10.10.1 Inputs .....	180
3.10.10.2 Outputs .....	180
3.10.10.3 Remarks .....	180
3.10.10.4 ST Language .....	180
3.10.10.5 FBD Language .....	180
3.10.10.6 FFLD Language .....	180
3.10.10.7 IL Language: .....	180
<b>3.11 Trigonometric functions .....</b>	<b>181</b>
3.11.1 ACOS / ACOSL .....	182
3.11.1.1 Inputs .....	182
3.11.1.2 Outputs .....	182
3.11.1.3 Remarks .....	182
3.11.1.4 ST Language .....	182
3.11.1.5 FBD Language .....	182
3.11.1.6 FFLD Language .....	182
3.11.1.7 IL Language: .....	182
3.11.2 ASIN / ASINL .....	183
3.11.2.1 Inputs .....	183
3.11.2.2 Outputs .....	183
3.11.2.3 Remarks .....	183
3.11.2.4 ST Language .....	183
3.11.2.5 FBD Language .....	183
3.11.2.6 FFLD Language .....	183
3.11.2.7 IL Language: .....	183
3.11.3 ATAN / ATANL .....	184
3.11.3.1 Inputs .....	184
3.11.3.2 Outputs .....	184
3.11.3.3 Remarks .....	184
3.11.3.4 ST Language .....	184
3.11.3.5 FBD Language .....	184
3.11.3.6 FFLD Language .....	184
3.11.3.7 IL Language: .....	184
3.11.4 ATAN2 / ATAN2L .....	185
3.11.4.1 Inputs .....	185
3.11.4.2 Outputs .....	185
3.11.4.3 Remarks .....	185
3.11.4.4 ST Language .....	185
3.11.4.5 FBD Language .....	185

---

3.11.4.6 FFLD Language .....	185
3.11.4.7 IL Language .....	185
3.11.5 COS / COSL .....	186
3.11.5.1 Inputs .....	186
3.11.5.2 Outputs .....	186
3.11.5.3 Remarks .....	186
3.11.5.4 ST Language .....	186
3.11.5.5 FBD Language .....	186
3.11.5.6 FFLD Language .....	186
3.11.5.7 IL Language: .....	186
3.11.6 SIN / SINL .....	187
3.11.6.1 Inputs .....	187
3.11.6.2 Outputs .....	187
3.11.6.3 Remarks .....	187
3.11.6.4 ST Language .....	187
3.11.6.5 FBD Language .....	187
3.11.6.6 FFLD Language .....	187
3.11.6.7 IL Language: .....	187
3.11.7 TAN / TANL .....	188
3.11.7.1 Inputs .....	188
3.11.7.2 Outputs .....	188
3.11.7.3 Remarks .....	188
3.11.7.4 ST Language .....	188
3.11.7.5 FBD Language .....	188
3.11.7.6 FFLD Language .....	188
3.11.7.7 IL Language: .....	188
3.11.8 UseDegrees .....	189
3.11.8.1 Inputs .....	189
3.11.8.2 Outputs .....	189
3.11.8.3 Remarks .....	189
3.11.8.4 ST Language .....	189
3.11.8.5 FBD Language .....	189
3.11.8.6 FFLD Language .....	189
3.11.8.7 IL Language .....	189
<b>3.12 String operations .....</b>	<b>190</b>
3.12.1 ArrayToString / ArrayToStringU .....	191
3.12.1.1 Inputs .....	191
3.12.1.2 Outputs .....	191
3.12.1.3 Remarks .....	191
3.12.1.4 ST Language .....	191
3.12.1.5 FBD Language .....	191
3.12.1.6 FFLD Language .....	191
3.12.1.7 IL Language .....	191
3.12.2 ASCII .....	192
3.12.2.1 Inputs .....	192
3.12.2.2 Outputs .....	192
3.12.2.3 Remarks .....	192

3.12.2.4	ST Language .....	192
3.12.2.5	FBD Language .....	192
3.12.2.6	FFLD Language .....	192
3.12.2.7	IL Language: .....	192
3.12.3	ATOH .....	193
3.12.3.1	Inputs .....	193
3.12.3.2	Outputs .....	193
3.12.3.3	Truth table (examples) .....	193
3.12.3.4	Remarks .....	193
3.12.3.5	ST Language .....	193
3.12.3.6	FBD Language .....	193
3.12.3.7	FFLD Language .....	193
3.12.3.8	IL Language: .....	193
3.12.4	CHAR .....	194
3.12.4.1	Inputs .....	194
3.12.4.2	Outputs .....	194
3.12.4.3	Remarks .....	194
3.12.4.4	ST Language .....	194
3.12.4.5	FBD Language .....	194
3.12.4.6	FFLD Language .....	194
3.12.4.7	IL Language: .....	194
3.12.5	CONCAT .....	195
3.12.5.1	Inputs .....	195
3.12.5.2	Outputs .....	195
3.12.5.3	Remarks .....	195
3.12.5.4	ST Language .....	195
3.12.5.5	FBD Language .....	195
3.12.5.6	FFLD Language .....	195
3.12.5.7	IL Language .....	195
3.12.6	CRC16 .....	196
3.12.6.1	Inputs .....	196
3.12.6.2	Outputs .....	196
3.12.6.3	Remarks .....	196
3.12.6.4	ST Language .....	196
3.12.6.5	FBD Language .....	196
3.12.6.6	FFLD Language .....	196
3.12.6.7	IL Language: .....	196
3.12.7	DELETE .....	197
3.12.7.1	Inputs .....	197
3.12.7.2	Outputs .....	197
3.12.7.3	Remarks .....	197
3.12.7.4	ST Language .....	197
3.12.7.5	FBD Language .....	197
3.12.7.6	FFLD Language .....	197
3.12.7.7	IL Language: .....	197
3.12.8	FIND .....	198
3.12.8.1	Inputs .....	198

---

3.12.8.2	Outputs	198
3.12.8.3	Remarks	198
3.12.8.4	ST Language	198
3.12.8.5	FBD Language	198
3.12.8.6	FPLD Language	198
3.12.8.7	IL Language:	198
3.12.9	HTOA	199
3.12.9.1	Inputs	199
3.12.9.2	Outputs	199
3.12.9.3	Truth table (examples)	199
3.12.9.4	Remarks	199
3.12.9.5	ST Language	199
3.12.9.6	FBD Language	199
3.12.9.7	FPLD Language	199
3.12.9.8	IL Language:	199
3.12.10	INSERT	200
3.12.10.1	Inputs	200
3.12.10.2	Outputs	200
3.12.10.3	Remarks	200
3.12.10.4	ST Language	200
3.12.10.5	FBD Language	200
3.12.10.6	FPLD Language	200
3.12.10.7	IL Language:	200
3.12.11	LEFT	201
3.12.11.1	Inputs	201
3.12.11.2	Outputs	201
3.12.11.3	Remarks	201
3.12.11.4	ST Language	201
3.12.11.5	FBD Language	201
3.12.11.6	FPLD Language	201
3.12.11.7	IL Language:	201
3.12.12	LoadString	202
3.12.12.1	Inputs	202
3.12.12.2	Outputs	202
3.12.12.3	Remarks	202
3.12.12.4	ST Language	202
3.12.12.5	FBD Language	202
3.12.12.6	FPLD Language	202
3.12.12.7	IL Language:	202
3.12.13	MID	203
3.12.13.1	Inputs	203
3.12.13.2	Outputs	203
3.12.13.3	Remarks	203
3.12.13.4	ST Language	203
3.12.13.5	FBD Language	203
3.12.13.6	FPLD Language	203
3.12.13.7	IL Language:	203



3.12.14 MLEN .....	204
3.12.14.1 Inputs .....	204
3.12.14.2 Outputs .....	204
3.12.14.3 Remarks .....	204
3.12.14.4 ST Language .....	204
3.12.14.5 FBD Language .....	204
3.12.14.6 FFLD Language .....	204
3.12.14.7 IL Language: .....	204
3.12.15 REPLACE .....	205
3.12.15.1 Inputs .....	205
3.12.15.2 Outputs .....	205
3.12.15.3 Remarks .....	205
3.12.15.4 ST Language .....	205
3.12.15.5 FBD Language .....	205
3.12.15.6 FFLD Language .....	205
3.12.15.7 IL Language: .....	205
3.12.16 RIGHT .....	206
3.12.16.1 Inputs .....	206
3.12.16.2 Outputs .....	206
3.12.16.3 Remarks .....	206
3.12.16.4 ST Language .....	206
3.12.16.5 FBD Language .....	206
3.12.16.6 FFLD Language .....	206
3.12.16.7 IL Language: .....	206
3.12.17 StringTable .....	207
3.12.17.1 Inputs .....	207
3.12.17.2 Outputs .....	207
3.12.17.3 Remarks .....	207
3.12.17.4 ST Language .....	207
3.12.17.5 FBD Language .....	207
3.12.17.6 FFLD Language .....	207
3.12.17.7 IL Language: .....	207
3.12.18 StringToArray / StringToArrayU .....	208
3.12.18.1 Inputs .....	208
3.12.18.2 Outputs .....	208
3.12.18.3 Remarks .....	208
3.12.18.4 ST Language .....	208
3.12.18.5 FBD Language .....	208
3.12.18.6 FFLD Language .....	208
3.12.18.7 IL Language: .....	208
<b>3.13 UDP Functions for PDMM/PCMM &amp; Simulator .....</b>	<b>209</b>
3.13.1 udpAddrMake .....	210
3.13.1.1 Description .....	210
3.13.1.2 Arguments .....	210
3.13.1.5 Examples .....	211
3.13.2 udpClose .....	212
3.13.2.1 Description .....	212

3.13.2.2 Arguments .....	212
3.13.2.5 Examples .....	213
3.13.3 udpCreate .....	214
3.13.3.1 Description .....	214
3.13.3.2 Arguments .....	214
3.13.3.5 Examples .....	215
3.13.4 udplsValid .....	216
3.13.4.1 Description .....	216
3.13.4.2 Arguments .....	216
3.13.4.5 Examples .....	217
3.13.5 udpRcvFrom .....	218
3.13.5.1 Description .....	218
3.13.5.2 Arguments .....	218
3.13.5.5 Examples .....	219
3.13.6 udpRcvFromArray .....	220
3.13.6.1 Description .....	220
3.13.6.2 Arguments .....	220
3.13.6.5 Examples .....	221
3.13.7 udpSendTo .....	222
3.13.7.1 Description .....	222
3.13.7.2 Arguments .....	222
3.13.7.5 Examples .....	223
3.13.8 udpSendToArray .....	224
3.13.8.1 Description .....	224
3.13.8.2 Arguments .....	224
3.13.8.5 Examples .....	225
<b>4 PLC Advanced Libraries .....</b>	<b>226</b>
<b>4.1 ALARM_A .....</b>	<b>228</b>
4.1.1 Inputs .....	228
4.1.2 Outputs .....	228
4.1.3 Sequence .....	228
4.1.4 Remarks .....	228
4.1.5 ST Language .....	228
4.1.6 FBD Language .....	228
4.1.7 FFLD Language .....	229
4.1.8 IL Language .....	229
<b>4.2 ALARM_M .....</b>	<b>230</b>
4.2.1 Inputs .....	230
4.2.2 Outputs .....	230
4.2.3 Sequence .....	230
4.2.4 Remarks .....	230
4.2.5 ST Language .....	230
4.2.6 FBD Language .....	230
4.2.7 FFLD Language .....	231
4.2.8 IL Language .....	231
<b>4.3 ApplyRecipeColumn .....</b>	<b>232</b>
4.3.1 Inputs .....	232

4.3.2 Outputs .....	233
4.3.3 Remarks .....	233
4.3.4 ST Language .....	233
4.3.5 FBD Language .....	234
4.3.6 FFLD Language .....	234
4.3.7 IL Language .....	234
<b>4.4 AS-interface Functions .....</b>	<b>235</b>
<b>4.5 AVERAGE / AVERAGEL .....</b>	<b>236</b>
4.5.1 Inputs .....	236
4.5.2 Outputs .....	236
4.5.3 Remarks .....	236
4.5.4 ST Language .....	236
4.5.5 FBD Language .....	236
4.5.6 FFLD Language .....	236
4.5.7 IL Language: .....	237
<b>4.6 CurveLin .....</b>	<b>238</b>
4.6.1 Inputs .....	238
4.6.2 Outputs .....	238
4.6.3 Remarks .....	238
<b>4.7 DERIVATE .....</b>	<b>239</b>
4.7.1 Inputs .....	239
4.7.2 Outputs .....	239
4.7.3 Remarks .....	239
4.7.4 ST Language .....	239
4.7.5 FBD Language .....	239
4.7.6 FFLD Language .....	239
4.7.7 IL Language: .....	239
<b>4.8 EnableEvents .....</b>	<b>240</b>
4.8.1 Inputs .....	240
4.8.2 Outputs .....	240
4.8.3 Remarks .....	240
4.8.4 ST Language .....	240
4.8.5 FBD Language .....	240
4.8.6 FFLD Language .....	240
4.8.7 IL Language: .....	240
<b>4.9 FIFO .....</b>	<b>241</b>
4.9.1 Inputs .....	241
4.9.2 Outputs .....	241
4.9.3 Remarks .....	241
4.9.4 ST Language .....	241
4.9.5 FBD Language .....	242
4.9.6 FFLD Language .....	242
4.9.7 IL Language .....	242
<b>4.10 FilterOrder1 .....</b>	<b>243</b>
4.10.1 Inputs .....	243
4.10.2 Outputs .....	243
4.10.3 Remarks .....	243

4.10.4 ST Language .....	243
4.10.5 Example .....	243
<b>4.11 File Management .....</b>	<b>244</b>
4.11.1 SD Card Access .....	246
4.11.2 File Path Conventions .....	247
4.11.2.1 Shared Directory Path Conventions .....	248
4.11.2.2 SD Card Path Conventions .....	249
4.11.2.3 File Name Warning & Limitations .....	250
4.11.3 File Management Functions .....	251
4.11.3.1 F_AOPEN .....	251
4.11.3.2 F_CLOSE .....	251
4.11.3.3 F_COPY .....	251
4.11.3.4 F_DELETE .....	252
4.11.3.5 F_EOF .....	252
4.11.3.6 F_EXIST .....	252
4.11.3.7 F_GETSIZE .....	252
4.11.3.8 F_RENAME .....	252
4.11.3.9 F_ROPEN .....	253
4.11.3.13 F_SEEK .....	253
4.11.3.14 F_WOPEN .....	253
4.11.3.15 FA_READ .....	254
4.11.3.16 FA_WRITE .....	254
4.11.3.17 FB_READ .....	254
4.11.3.18 FB_WRITE .....	254
4.11.3.19 FM_READ .....	255
4.11.3.20 FM_WRITE .....	255
4.11.3.22 SD_MOUNT .....	256
4.11.3.23 SD_UNMOUNT .....	256
4.11.3.24 SD_ISREADY .....	256
4.11.4 File Management Function Examples .....	257
<b>4.12 GETSYSINFO .....</b>	<b>258</b>
4.12.1 Inputs .....	258
4.12.2 Outputs .....	258
4.12.3 Remarks .....	258
4.12.4 ST Language .....	258
4.12.5 FBD Language .....	258
4.12.6 FFLD Language .....	259
4.12.7 IL Language: .....	259
<b>4.13 HYSTER .....</b>	<b>260</b>
4.13.1 Inputs .....	260
4.13.2 Outputs .....	260
4.13.3 Remarks .....	260
4.13.4 ST Language .....	260
4.13.5 FBD Language .....	260
4.13.6 FFLD Language .....	260
4.13.7 IL Language: .....	260
<b>4.14 INTEGRAL .....</b>	<b>261</b>

4.14.1 Inputs .....	261
4.14.2 Outputs .....	261
4.14.3 Remarks .....	261
4.14.4 ST Language .....	261
4.14.5 FBD Language .....	261
4.14.6 FFLD Language .....	261
4.14.7 IL Language: .....	261
<b>4.15 LIFO .....</b>	<b>263</b>
4.15.1 Inputs .....	263
4.15.2 Outputs .....	263
4.15.3 Remarks .....	263
4.15.4 ST Language .....	263
4.15.5 FBD Language .....	264
4.15.6 FFLD Language .....	264
4.15.7 IL Language .....	264
<b>4.16 LIM_ALRM .....</b>	<b>265</b>
4.16.1 Inputs .....	265
4.16.2 Outputs .....	265
4.16.3 Remarks .....	265
4.16.4 ST Language .....	265
4.16.5 FBD Language .....	265
4.16.6 FFLD Language .....	265
4.16.7 IL Language: .....	265
<b>4.17 LogFileCSV .....</b>	<b>267</b>
4.17.1 Inputs .....	267
4.17.2 Outputs .....	267
4.17.3 Remarks .....	267
4.17.4 ST Language .....	268
4.17.5 FBD Language .....	268
4.17.6 FFLD Language .....	268
4.17.7 IL Language .....	269
<b>4.18 PID .....</b>	<b>270</b>
4.18.1 Inputs .....	270
4.18.2 Outputs .....	270
4.18.3 Diagram .....	271
4.18.4 Remarks .....	271
4.18.5 ST Language .....	271
4.18.6 FBD Language .....	271
4.18.7 FFLD Language .....	272
4.18.8 IL Language .....	272
<b>4.19 PWM .....</b>	<b>273</b>
4.19.1 Inputs .....	273
4.19.2 Outputs .....	273
4.19.3 Remarks .....	273
4.19.4 ST Language .....	273
4.19.5 Example .....	273
<b>4.20 RAMP .....</b>	<b>275</b>

---

4.20.1 Inputs .....	275
4.20.2 Outputs .....	275
4.20.3 Time diagram .....	275
4.20.4 Remarks .....	275
4.20.5 ST Language .....	275
4.20.6 FBD Language .....	275
4.20.7 FFLD Language .....	276
4.20.8 IL Language .....	276
<b>4.21 Real Time Clock Management Functions .....</b>	<b>277</b>
4.21.1 DAY_TIME .....	279
4.21.1.1 Inputs .....	279
4.21.1.2 Outputs .....	279
4.21.1.3 Remarks .....	279
4.21.1.4 ST Language .....	279
4.21.1.5 FBD Language .....	279
4.21.1.6 FFLD Language .....	279
4.21.1.7 IL Language .....	279
4.21.2 DTFORMAT .....	280
4.21.2.1 Inputs .....	280
4.21.2.2 Outputs .....	280
4.21.2.3 Remarks .....	280
4.21.2.4 ST Language .....	280
4.21.2.5 FBD Language .....	280
4.21.2.6 FFLD Language .....	280
4.21.2.7 IL Language .....	280
4.21.3 DTAT .....	282
4.21.3.1 Inputs .....	282
4.21.3.2 Outputs .....	282
4.21.3.3 Remarks .....	282
4.21.3.4 ST Language .....	282
4.21.3.5 FBD Language .....	282
4.21.3.6 FFLD Language .....	283
4.21.3.7 IL Language: .....	283
4.21.4 DTEVERY .....	284
4.21.4.1 Inputs .....	284
4.21.4.2 Outputs .....	284
4.21.4.3 Remarks .....	284
4.21.4.4 ST Language .....	284
4.21.4.5 FBD Language .....	284
4.21.4.6 FFLD Language .....	284
4.21.4.7 IL Language: .....	284
<b>4.22 Serializeln .....</b>	<b>285</b>
4.22.1 Description .....	285
4.22.2 Arguments .....	285
4.22.2.1 Input .....	285
4.22.2.2 Output .....	286
4.22.3 Examples .....	286

4.22.3.1 Structured Text .....	286
<b>4.23 SerializeOut .....</b>	<b>287</b>
4.23.1 Description .....	287
4.23.2 Arguments .....	287
4.23.2.1 Input .....	287
4.23.2.2 Output .....	288
4.23.3 Examples .....	288
4.23.3.1 Structured Text .....	288
<b>4.24 SigID .....</b>	<b>289</b>
4.24.1 Inputs .....	289
4.24.2 Outputs .....	289
4.24.3 Remarks .....	289
4.24.4 ST Language .....	289
4.24.5 FBD Language .....	289
4.24.6 FFLD Language .....	289
4.24.7 IL Language .....	289
<b>4.25 SigPlay .....</b>	<b>290</b>
4.25.1 Inputs .....	290
4.25.2 Outputs .....	290
4.25.3 Remarks .....	290
4.25.4 ST Language .....	290
4.25.5 FBD Language .....	290
4.25.6 FFLD Language .....	290
4.25.7 IL Language .....	290
<b>4.26 SigScale .....</b>	<b>292</b>
4.26.1 Inputs .....	292
4.26.2 Outputs .....	292
4.26.3 Remarks .....	292
4.26.4 ST Language .....	292
4.26.5 FBD Language .....	292
4.26.6 FFLD Language .....	292
4.26.7 IL Language .....	292
<b>4.27 STACKINT .....</b>	<b>293</b>
4.27.1 Inputs .....	293
4.27.2 Outputs .....	293
4.27.3 Remarks .....	293
4.27.4 ST Language .....	293
4.27.5 FBD Language .....	293
4.27.6 FFLD Language .....	293
4.27.7 IL Language .....	293
<b>4.28 SurfLin .....</b>	<b>295</b>
4.28.1 Inputs .....	295
4.28.2 Outputs .....	295
4.28.3 Remarks .....	295
<b>4.29 VLID .....</b>	<b>296</b>
4.29.1 Inputs .....	296
4.29.2 Outputs .....	296

---

4.29.3 Remarks .....	296
4.29.4 ST Language .....	296
4.29.5 FBD Language .....	296
4.29.6 FFLD Language .....	296
4.29.7 IL Language .....	296



## 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:

[SFC: Sequential Function Chart](#)

FBD: Function Block Diagram

[FFLD: Free Form Ladder Diagram](#)

ST: Structured Text

IL: Instruction List

Use of ST instructions in graphic languages

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

### 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:*

Steps and initial steps  
Transitions and divergences  
Parallel branches  
Macro-steps  
Jump to a step

*Programming:*

Actions within a step  
Timeout on a step  
Programming a transition condition  
How SFC is executed

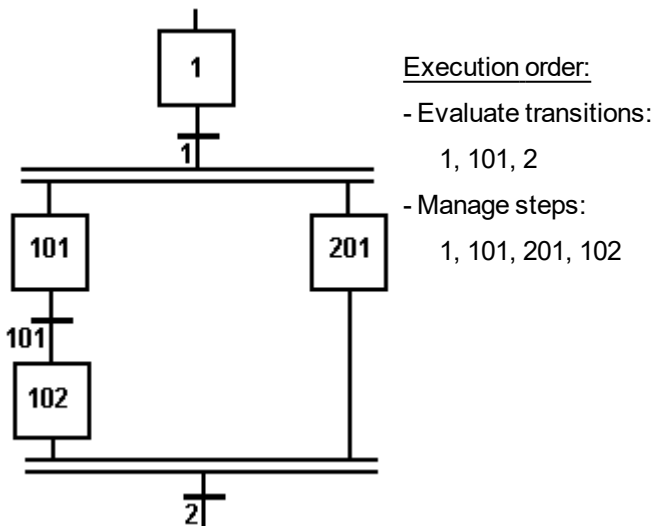
The KAS IDE fully supports SFC programming with several hierarchical levels of charts: i.e. a chart that controls another chart. Working with a hierarchy of SFC charts is an easy and powerful way for managing complex sequences and saves performances at runtime. Refer to the following sections for further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

- ["Hierarchy of SFC programs" \(p. 34\)](#)
- 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:



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 is 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 divergence 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.

## 2.2 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

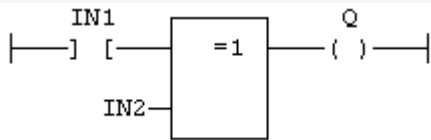
### 2.2.1 Use of the "EN" input and the "ENO" output for blocks

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 same status as the "EN" input: the rung state is not modified by a block having an ENO output.

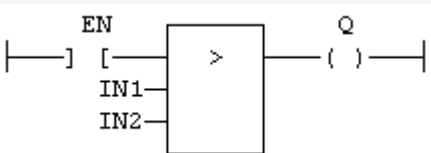
- Below is the example of the "XOR" block, having Boolean inputs and outputs, and requiring no EN or ENO pin:

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



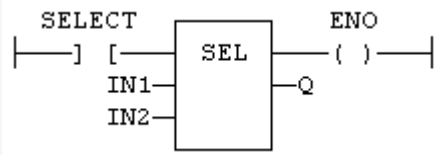
- Below is the example of the ">" (greater than) block, having non Boolean inputs and a Boolean output. This block has an "EN" input in FFLD language:

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



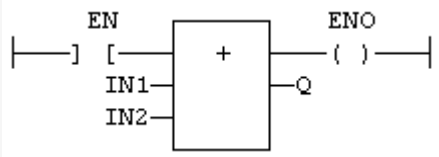
- Below is the example of the "SEL" function, having a first Boolean input, but an integer output. This block has an "ENO" output in FFLD language:

```
(* 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 *)
```



### 2.2.2 Contacts and coils

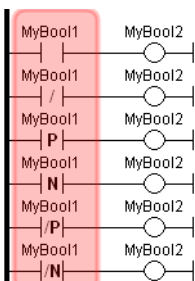
The table below contains a list of the contact and coil types available:

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)-


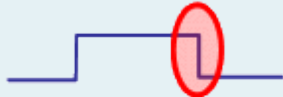
#### 2.2.2.1 FFLD Contacts

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:

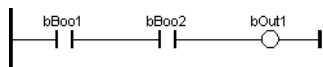


Contacts	Description
boolVariable -   -	<b>Normal:</b> The flow on the right is the Boolean AND operation between: (1) the flow on the left and (2) the associated variable.

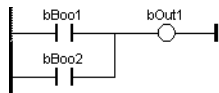
Contacts	Description
boolVariable -]/[-	<b>Negated:</b> The flow on the right is the Boolean AND operation between: (1) the flow on the left and (2) the negation of the associated variable.
boolVariable -]P[-	<b>Positive Transition:</b> The flow on the right is TRUE when the flow on the left is TRUE and the associated variable is TRUE and was FALSE the last time this contact was scanned (rising edge) 
boolVariable -]N[-	<b>Negative Transition:</b> 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). 
boolVariable -]/P[-	<b>Normally Closed Positive Transition:</b> 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 -]/N[-	<b>Normally Closed Negative Transition:</b> 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.

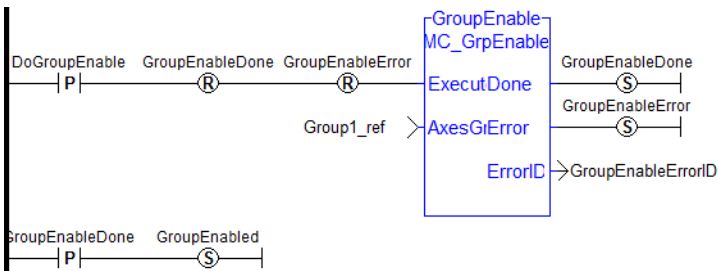


### Transition Contacts

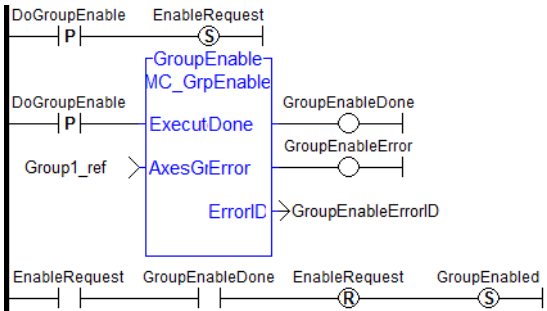
The transition contacts -]P[-, -]N -]/P[-, and -]/N[- compare the current state of the Boolean variable to the Boolean's state the last time the contact was scanned. This means that the Boolean variable could change states several times during a scan, but if it's back to the same state when the transition 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:



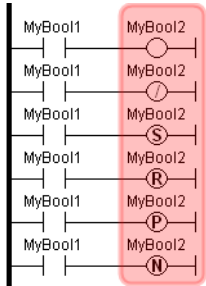
**TIP**

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

**2.2.2.2 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 - ( ) -	<b>Normal:</b> the associated variable is forced to the value of the flow on the left of the coil.
boolVariable - (/) -	<b>Negated:</b> the associated variable is forced to the negation of the flow on the left of the coil.

Coils	Description
boolVariable - (S) -	<p><b>Set:</b> the associated variable is forced to TRUE if the flow on the left is TRUE. (no action if the flow is FALSE)</p> <p><b>Rules for Set coil animation:</b></p> <ul style="list-style-type: none"> <li>• Power Flow on left is TRUE: <ul style="list-style-type: none"> <li>• The horizontal wires on either side of the (S) are red</li> <li>• The variable and the (S) are red</li> </ul> </li> <li>• Power Flow on left is FALSE and the (S) variable is Energized (ON) <ul style="list-style-type: none"> <li>• The horizontal lines on either side of (S) are black</li> <li>• The variable and the (S) are red</li> </ul> </li> <li>• In all other cases: <ul style="list-style-type: none"> <li>• The horizontal wires are black</li> <li>• The variable and the (S) are black</li> </ul> </li> </ul>
boolVariable - (R) -	<p><b>Reset:</b> the associated variable is forced to FALSE if the flow on the left is TRUE. (no action if the rung state is FALSE)</p> <p><b>Rules for Reset coil animation:</b></p> <ul style="list-style-type: none"> <li>• Power Flow on left is TRUE: <ul style="list-style-type: none"> <li>• The horizontal lines are red</li> <li>• The variable above (R) is black</li> <li>• The R and the circle around the R are black</li> </ul> </li> <li>• Power Flow on left is FALSE and variable above reset coil is NOT Energized (OFF) <ul style="list-style-type: none"> <li>• The horizontal lines are black</li> <li>• The variable above (R) is black</li> <li>• The R and the circle around the R are black</li> </ul> </li> <li>• Power Flow on left is FALSE and variable above reset coil is Energized (ON) <ul style="list-style-type: none"> <li>• The horizontal lines are black</li> <li>• The variable above (R) is red</li> <li>• The R and the circle around the R are red</li> </ul> </li> </ul>
boolVariable - (P) -	<p><b>Positive transition:</b> the associated variable is forced to TRUE if the flow on the left changes from <b>FALSE to TRUE</b>(and forced to FALSE in all other cases)</p>
boolVariable - (N) -	<p><b>Negative transition:</b> the associated variable is forced to TRUE if the flow on the left changes from <b>TRUE to FALSE</b>(and forced to FALSE in all other cases)</p>

### 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:

- "Basic Operations" (p. 41)
- "Boolean operations" (p. 64)
- "Arithmetic operations" (p. 82)
- "Comparison Operations" (p. 96)
- "Type conversion functions" (p. 104)
- "Selectors" (p. 117)
- "Registers" (p. 123)
- "Counters" (p. 147)
- "Timers" (p. 152)
- "Mathematic operations" (p. 169)
- "Trigonometric functions" (p. 181)
- "String operations" (p. 190)
- "PLC Advanced Libraries" (p. 226)

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:

<a href="#">IF</a>	Conditional execution of statements.
<a href="#">WHILE</a>	Repeat statements while a condition is TRUE.
<a href="#">REPEAT</a>	Repeat statements until a condition is TRUE.
<a href="#">FOR</a>	Execute iterations of statements.
<a href="#">CASE</a>	Switch to one of various possible statements.
<a href="#">EXIT</a>	Exit from a loop instruction.
<a href="#">WAIT</a>	Delay program execution.
<a href="#">ON</a>	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**

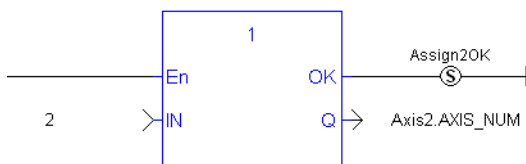
Q := IN; (\* copy IN into variable Q \*)  
 Q := (IN1 + (IN2 / IN 3)) \* IN4; (\* assign the result of a complex expression \*)  
 result := SIN (angle); (\* assign a variable with the result of a function \*)  
 time := MyTon.ET; (\* assign a variable with an output parameter of a function block \*)

**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<sup>1</sup> (  ) is called once and it performs an action. This is synchronous.
- A Function Block<sup>2</sup> (  ) 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.

---

<sup>1</sup>A 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.

<sup>2</sup>A function block groups an algorithm and a set of private data. It has inputs and outputs.

### 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
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);

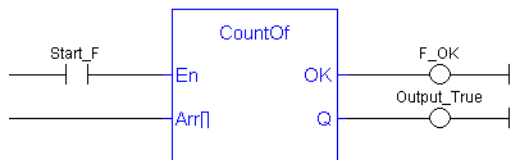
#### 3.1.6.5 FBD Language



#### 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

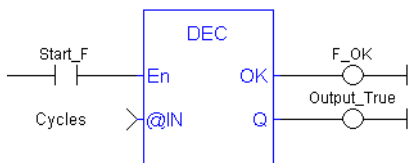
```
IN := 2;
Q := DEC (IN);
(* now: IN = 1 ; Q = 1 *)
```

```
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.

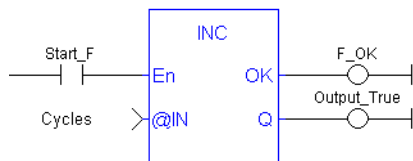
The IN input must be directly connected to a variable, and cannot be a constant or complex expression.

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.

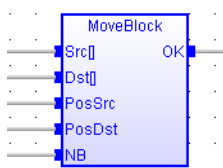
The function copies a number (NB) of consecutive items starting at the PosSRC index in SRC array to PosDST position in DST array. SRC and DST can be the same array. In that case, the function avoids lost items when source and destination areas overlap.

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

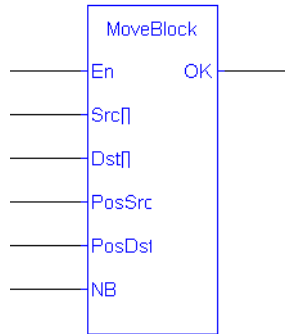
```
OK := MOVEBLOCK (SRC, DST, PosSRC, PosDST, NB);
```

#### 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

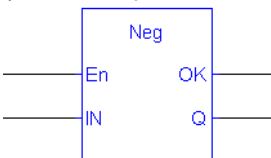
```
Q := -IN;
Q := - (IN1 + IN2);
```

#### 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;
END_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.

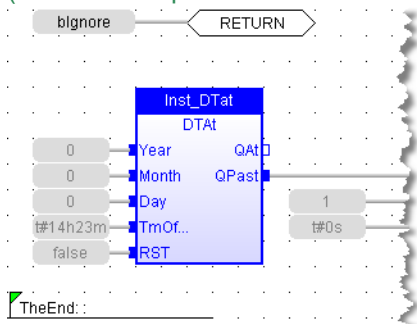
When used within an action block of an SFC step, the RETURN statement jumps to the end of the action block.

#### 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 *)
```

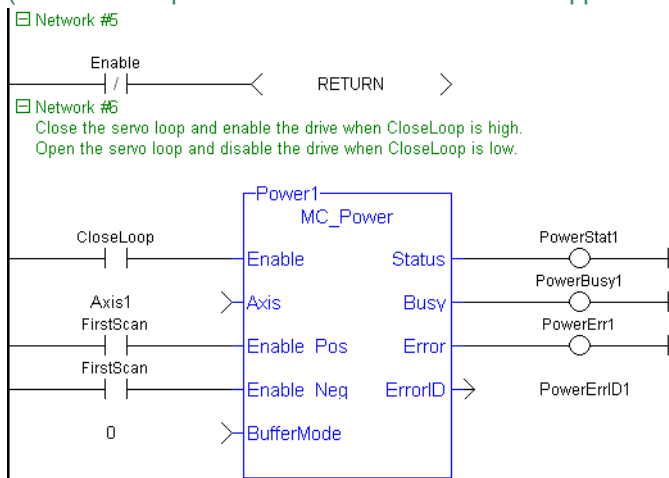
#### 3.1.17.3 FBD Language

(\* 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
<a href="#">XOR</a>	performs an exclusive OR
<a href="#">NOT</a>	performs a Boolean negation of its input
"QOR" (p. 70)	qualified OR
<a href="#">S</a>	force a Boolean output to TRUE
<a href="#">R</a>	force a Boolean output to FALSE

Below are the available blocks for managing Boolean signals:

<a href="#">RS</a>	reset dominant bistable
<a href="#">SR</a>	set dominant bistable
<a href="#">R_TRIG</a>	rising pulse detection
<a href="#">F_TRIG</a>	falling pulse detection
<a href="#">SEMA</a>	semaphore
<a href="#">FLIPFLOP</a>	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

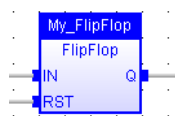
The output is systematically reset to FALSE if RST is TRUE.  
The output changes on each rising edge of the IN input, if RST is FALSE.

#### 3.2.1.4 ST Language

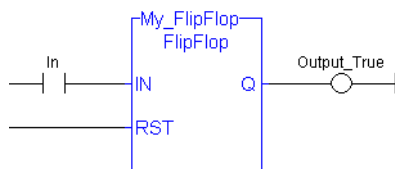
(\* MyFlipFlop is declared as an instance of FLIPFLOP function block \*)

```
MyFlipFlop (IN, RST);
Q := MyFlipFlop.Q;
```

#### 3.2.1.5 FBD Language



#### 3.2.1.6 FLD 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

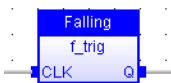
#### 3.2.2.4 Remarks

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.

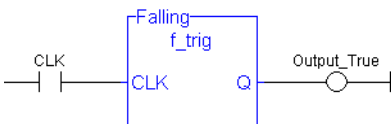
#### 3.2.2.5 ST Language

(\* 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 \*)

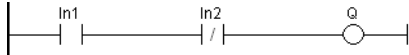


(\* use of a negated link: Q is IN1 AND NOT IN2 \*)

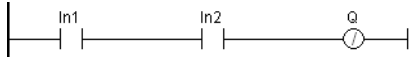


#### 3.2.3.7 FFLD Language

(\* 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);
```

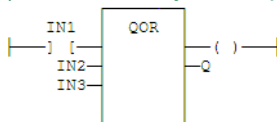
#### 3.2.4.5 FBD Language

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



#### 3.2.4.6 FLD 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 <i>prev</i>	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

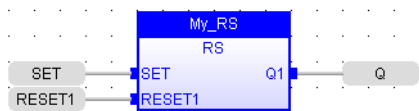
#### 3.2.6.4 Remarks

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

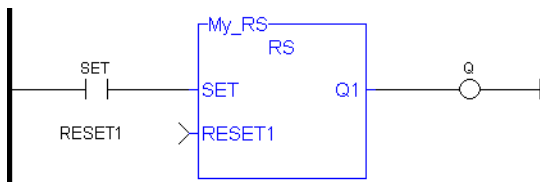
```
(* MyRS is declared as an instance of RS function block *)
MyRS (SET, RESET1);
Q1 := MyRS.Q1;
```

#### 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 <i>prev</i>	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

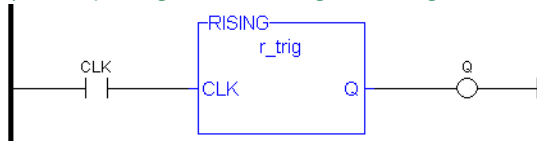
```
(* MyTrigger is declared as an instance of R_TRIG function block
*)
MyTrigger (CLK);
Q := MyTrigger.Q;
```

#### 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 PLCopen ✓

*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;
```

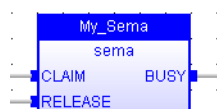
In FFLD language, the input rung is the CLAIM command. The output rung is the BUSY output signal.

#### 3.2.9.4 ST Language

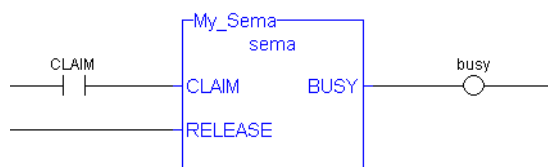
(\* MySema is a declared instance of SEMA function block \*)

```
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**

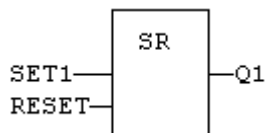
SET1	RESET	Q1 <i>prev</i>	Q1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

**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).

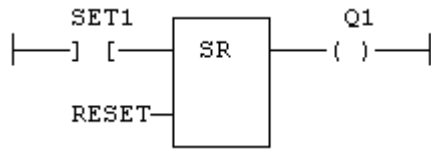
**3.2.10.5 ST Language**

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

**3.2.10.6 FBD Language**

### 3.2.10.7 FFLD Language

(\* the SET1 command is the rung - 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

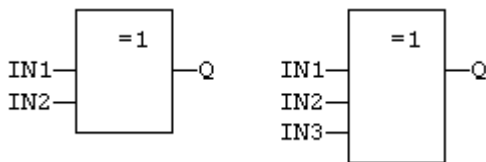
#### 3.2.11.4 Remarks

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

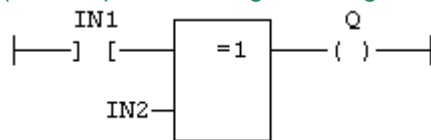
Q := IN1 XOR IN2;  
 Q := IN1 XOR IN2 XOR IN3;

#### 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:

<b>"+" Addition" (p. 83)</b>	addition
<b>"-" Subtraction" (p. 95)</b>	subtraction
<b>"*" Multiply" (p. 92)</b>	multiplication
<b>"/ Divide" (p. 85)</b>	division
<b>- (NEG)</b>	integer negation (unary operator)

Below are the standard functions that perform arithmetic operations:

<b><u>MIN</u></b>	get the minimum of two integers or an ANY
<b><u>MAX</u></b>	get the maximum of two integers or an ANY
<b><u>LIMIT</u></b>	bound an integer to low and high limits or an ANY
<b><u>MOD</u></b>	modulo
<b><u>ODD</u></b>	test if an integer is odd
<b>"SetWithin" (p. 94)</b>	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.

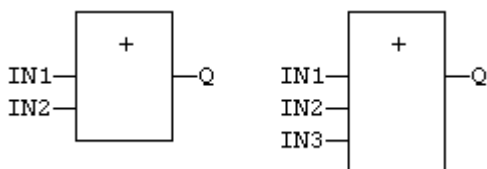
The addition can be used with strings. The result is the concatenation of the input strings.

#### 3.3.1.4 ST Language

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

#### 3.3.1.5 FBD Language

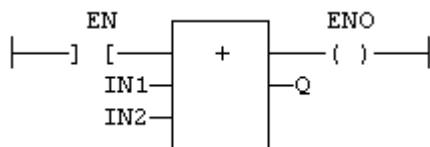
(\* the block can have up to 16 inputs \*)



#### 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

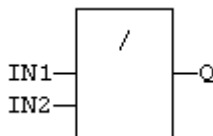
#### 3.3.2.3 Remarks

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

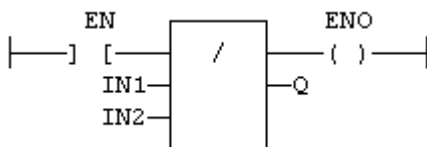
Q := IN1 / IN2;

#### 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

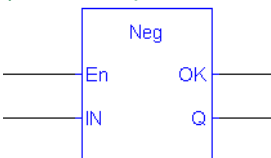
```
Q := -IN;
Q := - (IN1 + IN2);
```

#### 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.

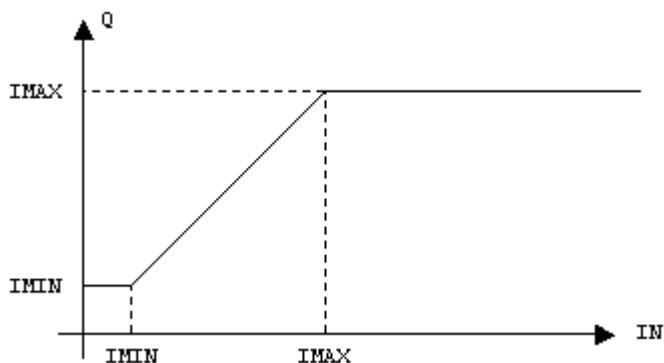
#### 3.3.4.1 Inputs

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



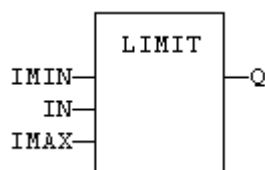
#### 3.3.4.4 Remarks

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);

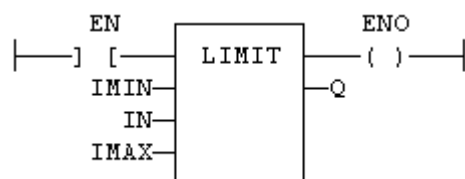
#### 3.3.4.6 FBD Language



#### 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

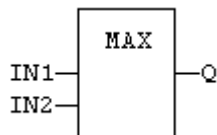
#### 3.3.5.3 Remarks

In FFD 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);

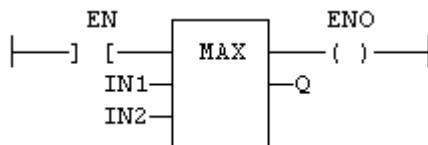
#### 3.3.5.5 FBD Language



#### 3.3.5.6 FFD 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

*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

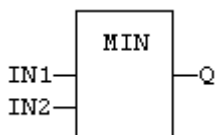
#### 3.3.6.3 Remarks

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);

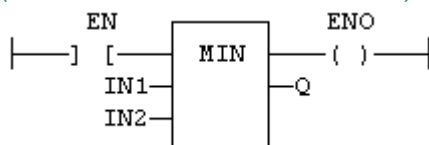
#### 3.3.6.5 FBD Language



#### 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.

Inputs		Function			Description
		MOD	MODR	MODLR	
IN		DINT	REAL	LREAL	Input value
BASE		DINT	REAL	LREAL	Base of the modulo
Output		Function			Description
		MOD	MODR	MODLR	
Q		DINT	REAL	LREAL	Modulo: rest of the integer division (IN / BASE)

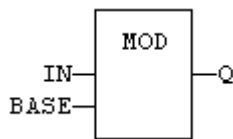
#### 3.3.7.1 Remarks

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.

#### 3.3.7.2 ST Language

Q := MOD (IN, BASE);

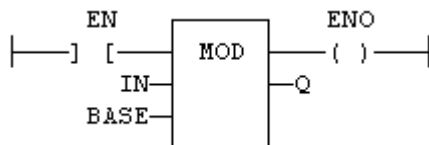
#### 3.3.7.3 FBD Language



#### 3.3.7.4 FFLD Language

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

(\* ENO has the same value as EN \*)



#### 3.3.7.5 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

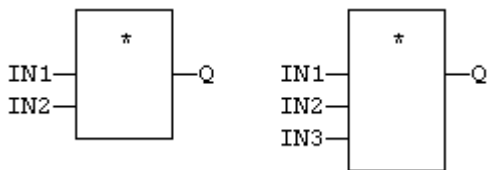
All inputs and the output must have the same type. In FBD language, the block can have up to 16 inputs. In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the same value as the input rung. In IL language, the MUL instruction performs a multiplication between the current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

#### 3.3.8.4 ST Language

Q := IN1 \* IN2;

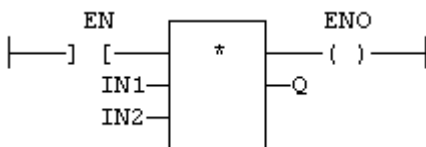
#### 3.3.8.5 FBD Language

(\* 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**

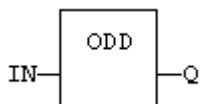
Q : BOOL TRUE if IN is odd. FALSE if IN is even.

**3.3.9.3 Remarks**

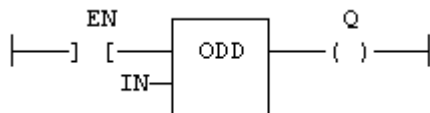
In FFLD language, the input rung (EN) enables the operation, and the output rung is the result of the function.  
In IL language, the input must be loaded before the function call.

**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** 

*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

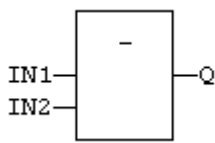
#### 3.3.11.3 Remarks

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

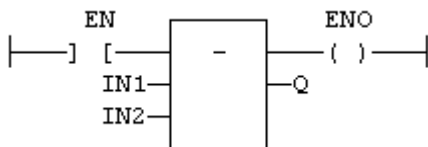
Q := IN1 - IN2;

#### 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
<b>CMP</b>	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

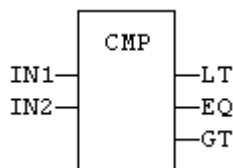
#### 3.4.1.3 Remarks

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

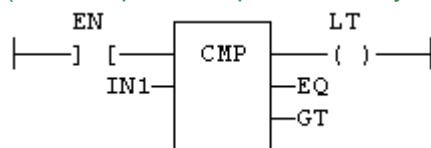
(\* MyCmp is declared as an instance of CMP function block \*)  
MyCMP (IN1, IN2);  
bLT := MyCmp.LT;  
bEQ := MyCmp.EQ;  
bGT := MyCmp.GT;

#### 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**

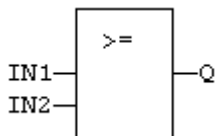
Both inputs must have the same type. In FFLD language, the input rung (EN) enables the operation, and the output rung is the result of the comparison. In IL language, the GE instruction performs the comparison between the current result and the operand. The current result and the operand must have the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

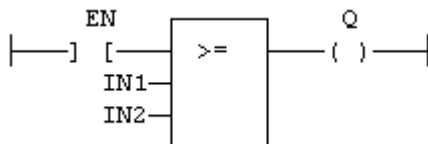
Q := IN1 >= IN2;

**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** 

*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

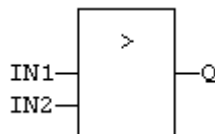
**3.4.3.3 Remarks**

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.

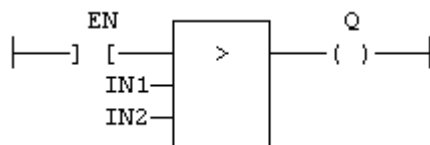
Comparisons can be used with strings. In that case, the lexical order is used for comparing the input strings. For instance, "ABC" is less than "ZX" ; "ABCD" is greater than "ABC".

**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**

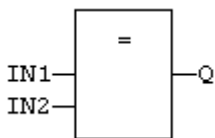
Both inputs must have the same type. In FFLD language, the input rung (EN) enables the operation, and the output rung is the result of the comparison. In IL language, the EQ instruction performs the comparison between the current result and the operand. The current result and the operand must have the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

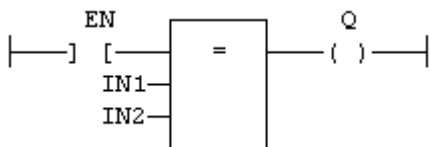
Equality comparisons cannot be used with TIME variables. The reason is that the timer actually has the resolution of the target cycle and test can be unsafe as some values can never be reached.

**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**

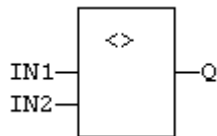
Both inputs must have the same type. In FFLD language, the input rung (EN) enables the operation, and the output rung is the result of the comparison. In IL language, the NE instruction performs the comparison between the current result and the operand. The current result and the operand must have the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

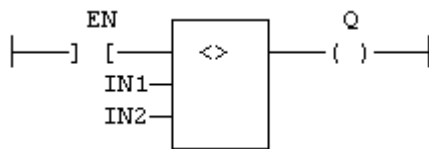
Equality comparisons cannot be used with TIME variables. The reason is that the timer actually has the resolution of the target cycle and test can be unsafe as some values can never be reached

**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

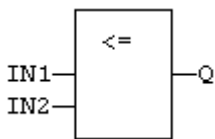
Both inputs must have the same type. In FFLD language, the input rung (EN) enables the operation, and the output rung is the result of the comparison. In IL language, the LE instruction performs the comparison between the current result and the operand. The current result and the operand must have the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

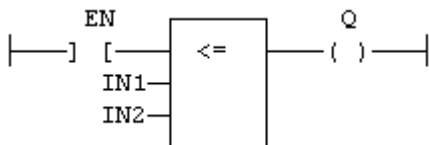
Q := IN1 <= IN2;

#### 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

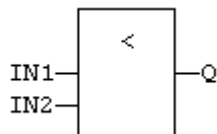
**3.4.7.3 Remarks**

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.

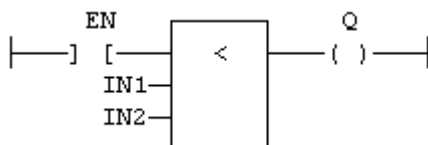
Comparisons can be used with strings. In that case, the lexical order is used for comparing the input strings. For instance, "ABC" is less than "ZX" ; "ABCD" is greater than "ABC".

**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:

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

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

<a href="#">BIN_TO_BCD</a>	converts a binary value to a BCD value
<a href="#">BCD_TO_BIN</a>	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

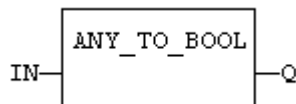
#### 3.5.1.3 Remarks

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

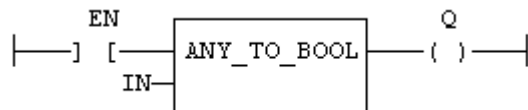


#### 3.5.1.6 FFLD Language

(\* 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

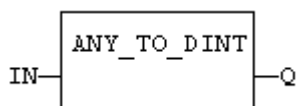
#### 3.5.2.3 Remarks

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);

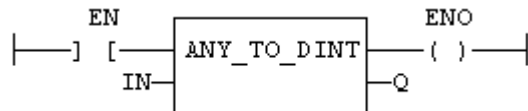
#### 3.5.2.5 FBD Language



#### 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

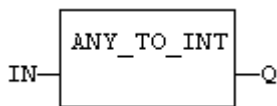
#### 3.5.3.3 Remarks

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);

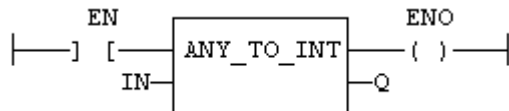
#### 3.5.3.5 FBD Language



#### 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

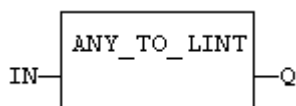
#### 3.5.4.3 Remarks

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);

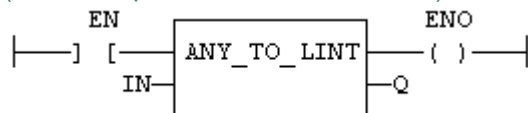
#### 3.5.4.5 FBD Language



#### 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**

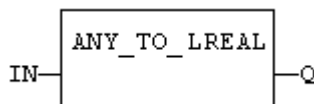
Q : LREAL Value converted to double precision real

**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.

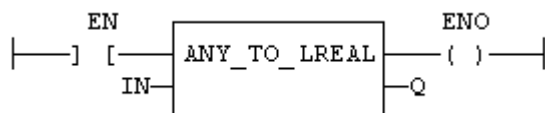
**3.5.5.4 ST Language**

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

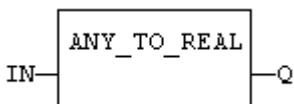
#### 3.5.6.3 Remarks

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);

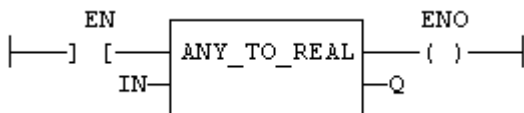
#### 3.5.6.5 FBD Language



#### 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**

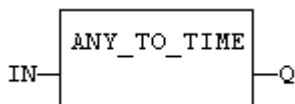
Q : TIME Value converted to time

**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.

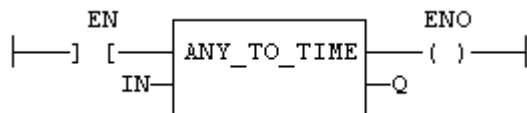
**3.5.7.4 ST Language**

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

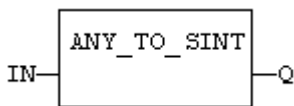
#### 3.5.8.3 Remarks

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);

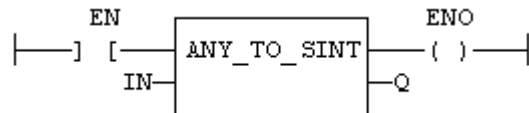
#### 3.5.8.5 FBD Language



#### 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

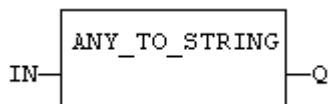
#### 3.5.9.3 Remarks

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);

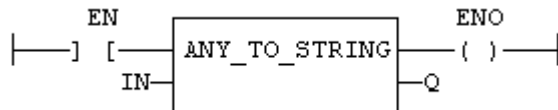
#### 3.5.9.5 FBD Language



#### 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

*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

**3.5.11.3.1 Truth table (examples)**

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

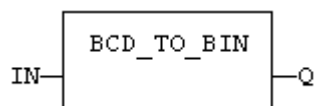
**3.5.11.4 Remarks**

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.

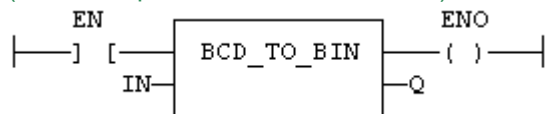
**3.5.11.5 ST Language**

Q := BCD\_TO\_BIN (IN);

**3.5.11.6 FBD Language****3.5.11.7 FFLD Language**

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

(\* ENO keeps the same value as EN \*)

**3.5.11.8 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)

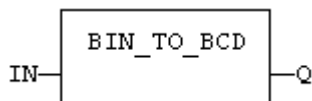
**3.5.12.4 Remarks**

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.

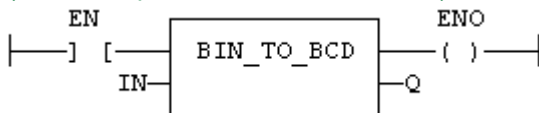
**3.5.12.5 ST Language**

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:

<a href="#">SEL</a>	2 integer inputs
<a href="#">MUX4</a>	4 integer inputs
<a href="#">MUX8</a>	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

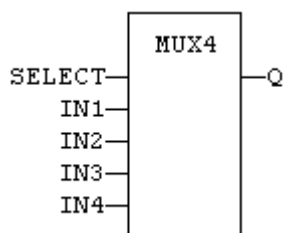
#### 3.6.1.4 Remarks

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

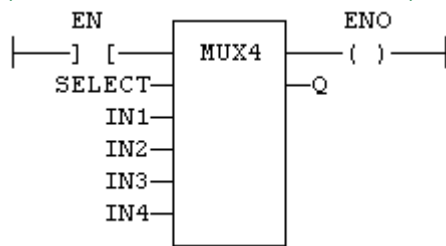
Q := MUX4 (SELECT, IN1, IN2, IN3, IN4);

#### 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

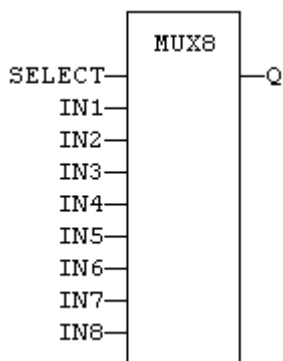
#### 3.6.2.4 Remarks

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);

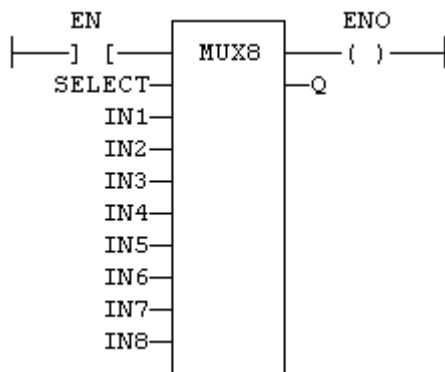
#### 3.6.2.6 FBD Language



#### 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

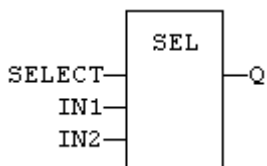
#### 3.6.3.4 Remarks

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

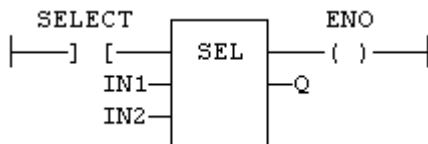
Q := SEL (SELECT, IN1, IN2);

#### 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:

<a href="#">SHL</a>	shift left
<a href="#">SHR</a>	shift right
<a href="#">ROL</a>	rotation left
<a href="#">ROR</a>	rotation right

Below are advanced functions for register manipulation:

<a href="#">MBSHift</a>	multibyte shift / rotate
-------------------------	--------------------------

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

<a href="#">AND_MASK</a>	Boolean AND
<a href="#">OR_MASK</a>	Boolean OR
<a href="#">XOR_MASK</a>	exclusive OR
<a href="#">NOT_MASK</a>	Boolean negation

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

<a href="#">LOBYTE</a>	Get the lowest byte of a word
<a href="#">HIBYTE</a>	Get the highest byte of a word
<a href="#">LOWORD</a>	Get the lowest word of a double word
<a href="#">HIWORD</a>	Get the highest word of a double word
<a href="#">MAKEWORD</a>	Pack bytes to a word
<a href="#">MAKEDWORD</a>	Pack words to a double word
<a href="#">PACK8</a>	Pack bits in a byte
<a href="#">UNPACK8</a>	Extract bits from a byte

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

<a href="#">SETBIT</a>	Set a bit in a register
<a href="#">TESTBIT</a>	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

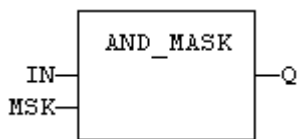
Arguments can be signed or unsigned integers from 8 to 32 bits.

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);

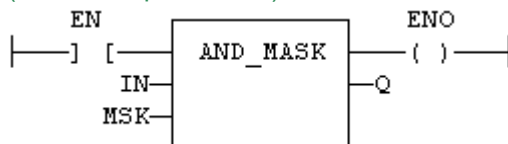
#### 3.7.1.5 FBD Language



#### 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

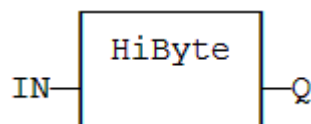
**3.7.2.3 Remarks**

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.

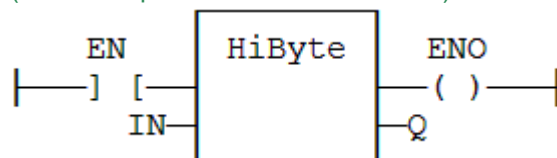
**3.7.2.4 ST Language**

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

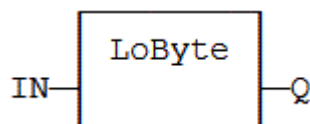
**3.7.3.3 Remarks**

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.

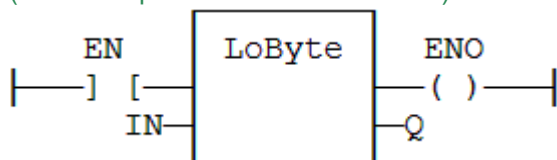
**3.7.3.4 ST Language**

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

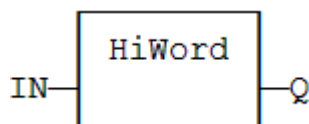
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

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

#### 3.7.4.4 ST Language

Q := HIWORD (IN);

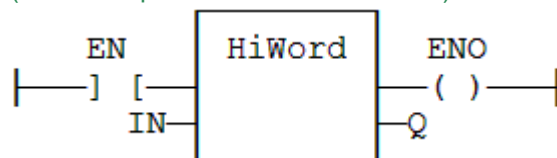
#### 3.7.4.5 FBD Language



#### 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

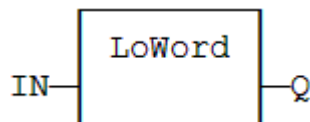
**3.7.5.3 Remarks**

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.

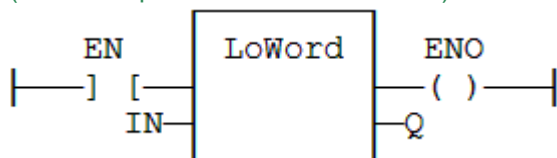
**3.7.5.4 ST Language**

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

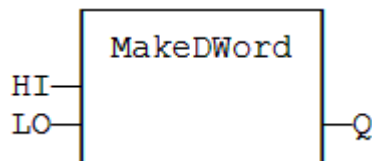
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

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

#### 3.7.6.4 ST Language

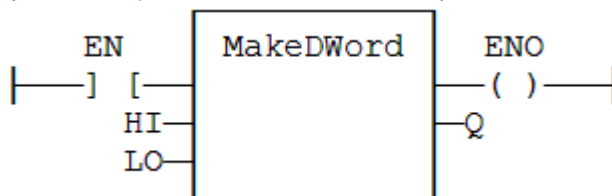
Q := MAKEDWORD (HI, LO);

#### 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

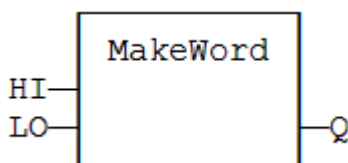
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

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);

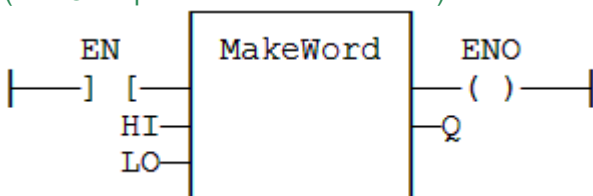
#### 3.7.7.5 FBD Language



#### 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

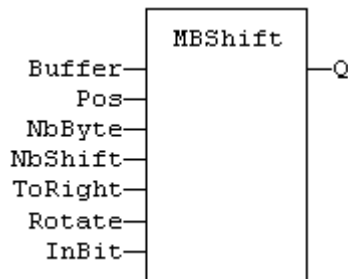
Use the "ToRight" argument to specify a shift to the left (FALSE) or to the right (TRUE). Use the "Rotate" argument to specify either a shift (FALSE) or a rotation (TRUE). In case of a shift, the "InBit" argument specifies the value of the bit that replaces the last shifted bit.

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

#### 3.7.8.4 ST Language

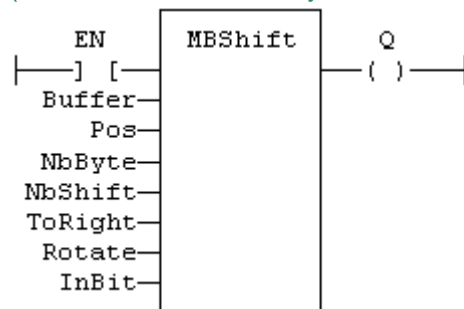
Q := MBSHift (Buffer, Pos, NbByte, NbShift, ToRight, Rotate, InBit);

#### 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

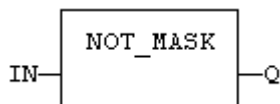
**3.7.9.3 Remarks**

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.

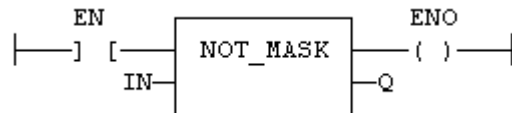
**3.7.9.4 ST Language**

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

*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

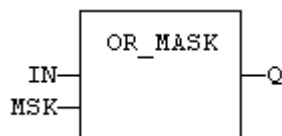
Arguments can be signed or unsigned integers from 8 to 32 bits.

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);

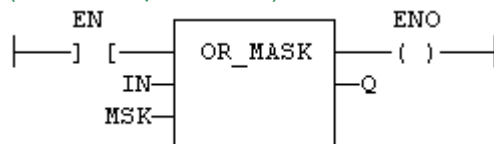
#### 3.7.10.5 FBD Language



#### 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

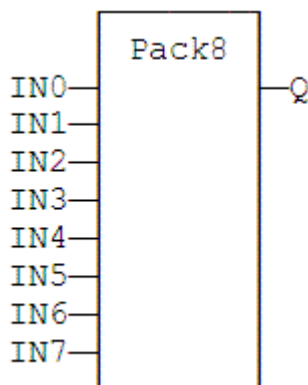
#### 3.7.11.3 Remarks

- 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

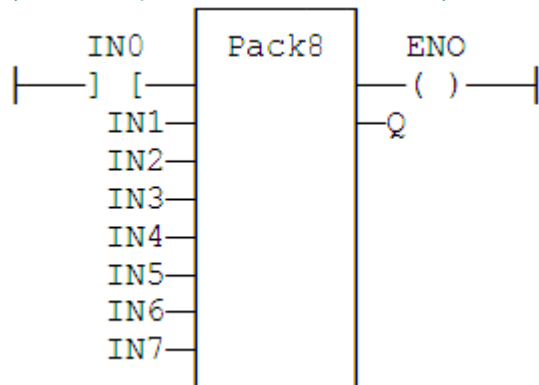
```
Q := PACK8 (IN0, IN1, IN2, IN3, IN4, IN5, IN6, IN7);
```

#### 3.7.11.5 FBD Language



### 3.7.11.6 FLD 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

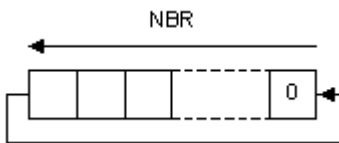
IN : ANY register

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

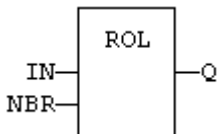
Arguments can be signed or unsigned integers from 8 to 32 bits.

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);

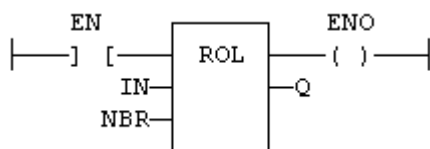
#### 3.7.12.6 FBD Language



#### 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** PLCopen ✓

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

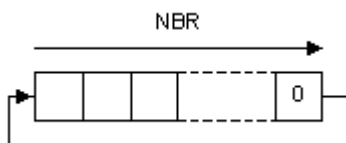
**3.7.13.1 Inputs**

IN : ANY register

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

**3.7.13.2 Outputs**

Q : ANY Rotated register

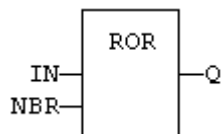
**3.7.13.3 Diagram****3.7.13.4 Remarks**

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.

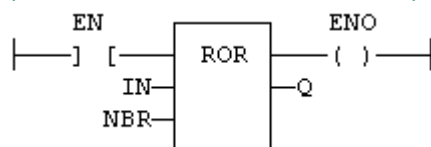
**3.7.13.5 ST Language**

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



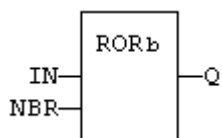
#### 3.7.14.4 Remarks

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);

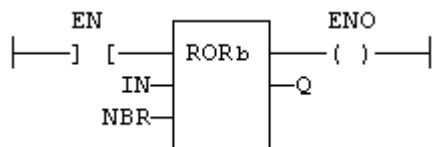
#### 3.7.14.6 FBD Language



#### 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



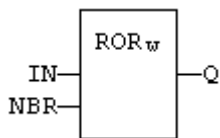
#### 3.7.15.4 Remarks

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);

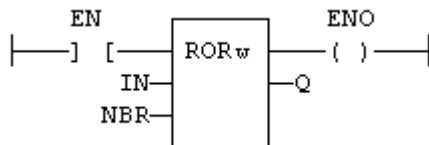
#### 3.7.15.6 FBD Language



#### 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 32 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

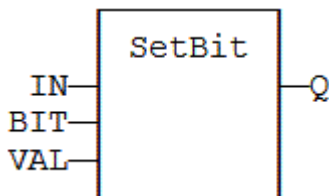
**3.7.16.3 Remarks**

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.

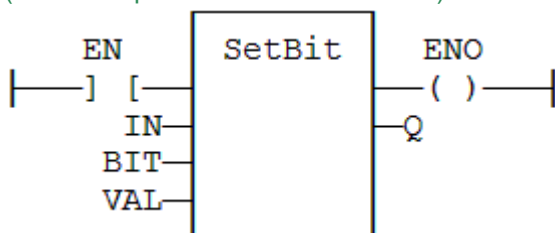
**3.7.16.4 ST Language**

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.

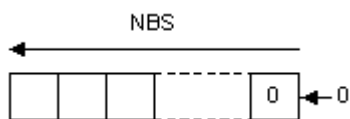
**3.7.17.1 Inputs**

IN : ANY register

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

**3.7.17.2 Outputs**

Q : ANY Shifted register

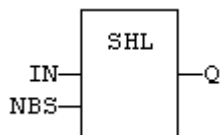
**3.7.17.3 Diagram****3.7.17.4 Remarks**

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.

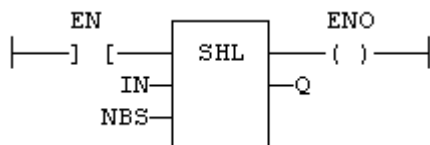
**3.7.17.5 ST Language**

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

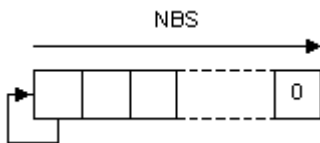
IN : ANY register

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

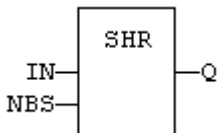
Arguments can be signed or unsigned integers from 8 to 32 bits.

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);

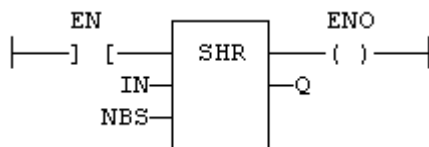
#### 3.7.18.6 FBD Language



#### 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 32 bit integer register  
 BIT : DINT    Bit number (0 = less significant bit)

**3.7.19.2 Outputs**

Q : BOOL     Bit value

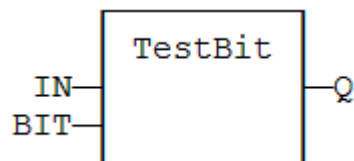
**3.7.19.3 Remarks**

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.

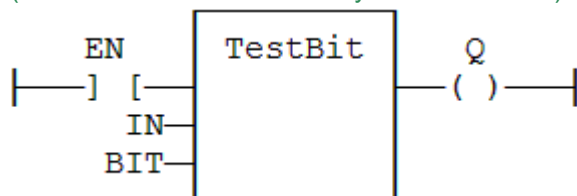
In FFD language, the operation is executed only if the input rung (EN) is TRUE. The output rung is the output of the function.

**3.7.19.4 ST Language**

Q := TESTBIT (IN, BIT);

**3.7.19.5 FBD Language****3.7.19.6 FFD 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;

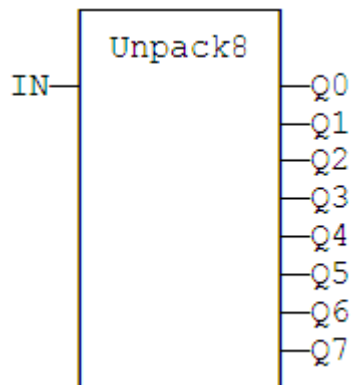
Q3 := MyUnpack.Q3;

Q4 := MyUnpack.Q4;

Q5 := MyUnpack.Q5;

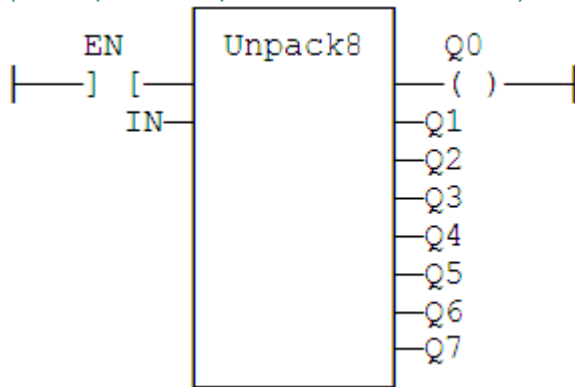
Q6 := MyUnpack.Q6;

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

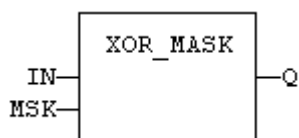
Arguments can be signed or unsigned integers from 8 to 32 bits.

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);

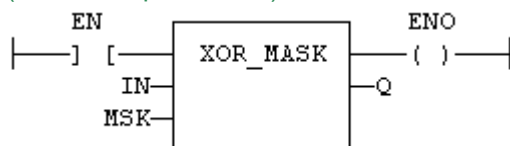
#### 3.7.21.5 FBD Language



#### 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:

<a href="#">CTU</a>	Up counter
<a href="#">CTD</a>	Down Counter
<a href="#">CTUD</a>	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

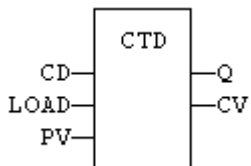
The counter is empty (CV = 0) when the application starts. The counter does not include a pulse detection for CD input. Use R\_TRIG or F\_TRIG function block for counting pulses of CD input signal. In FFLD language, CD is the input rung. The output rung is the Q output.

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.

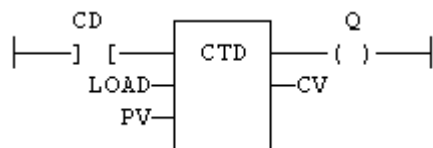
#### 3.8.1.4 ST Language

(\* 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

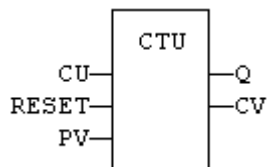
(\* MyCounter is a declared instance of CTU function block \*)

MyCounter (CU, RESET, PV);

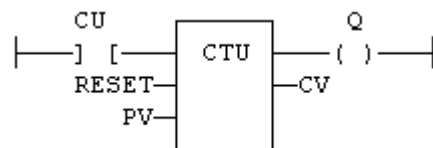
Q := MyCounter.Q;

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

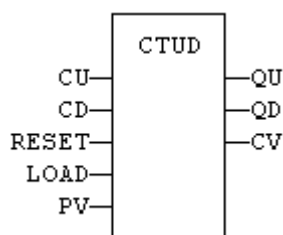
#### 3.8.3.3 Remarks

- 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.

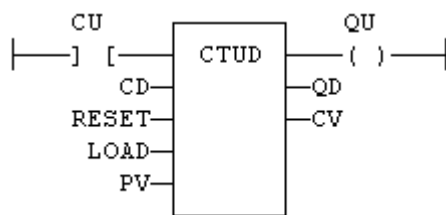
#### 3.8.3.4 ST Language

```
(* 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:

<a href="#">TON</a>	On timer
<a href="#">TOF</a>	Off timer
<a href="#">TP</a>	Pulse timer
<a href="#">BLINK</a>	Blinker
<a href="#">BLINKA</a>	Asymmetric blinker
<a href="#">PLS</a>	Pulse signal generator
<a href="#">TMU</a>	Up-counting stop watch
<a href="#">TMUsec</a>	Up-counting stop watch (seconds)
<a href="#">TMD</a>	Down-counting stop watch



**3.9.1 BLINK** PLCopen 

*Function Block - Blinker.*

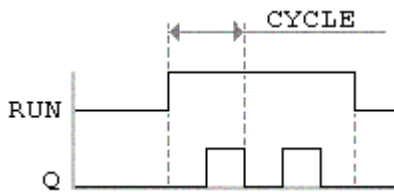
**3.9.1.1 Inputs**

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**

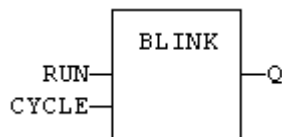
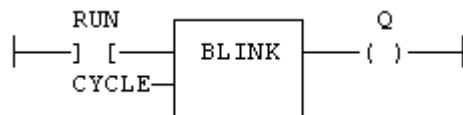
The output signal is FALSE when the RUN input is FALSE. The CYCLE input is the complete period of the blinking signal. In FFLD language, the input rung is the IN command. The output rung is the Q output signal.

**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.

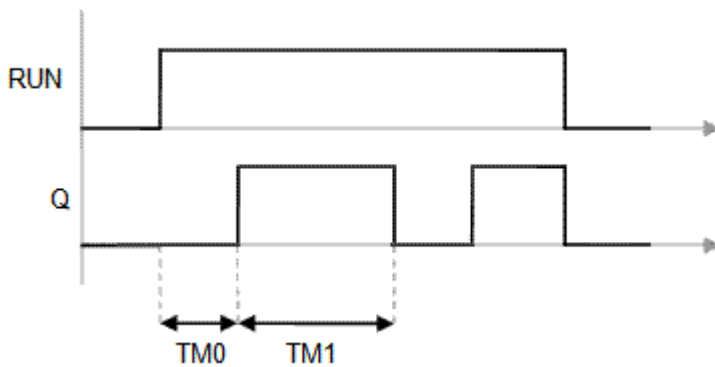
#### 3.9.2.1 Inputs

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



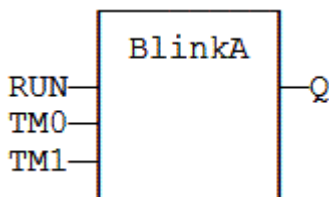
#### 3.9.2.4 Remarks

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

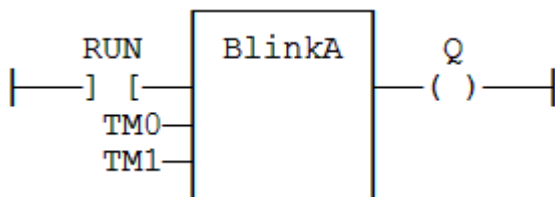
#### 3.9.2.5 ST Language

(\* 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 FLD 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**

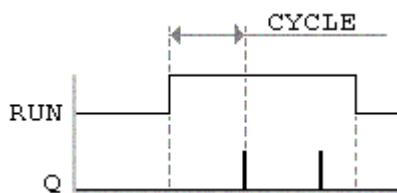
[TON](#) [TOF](#) [TP](#)

**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**

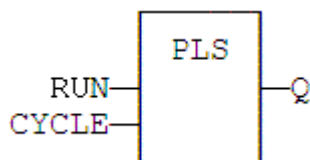
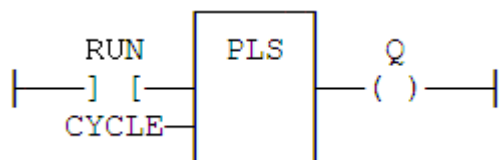
Q : BOOL    Output pulse signal

**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

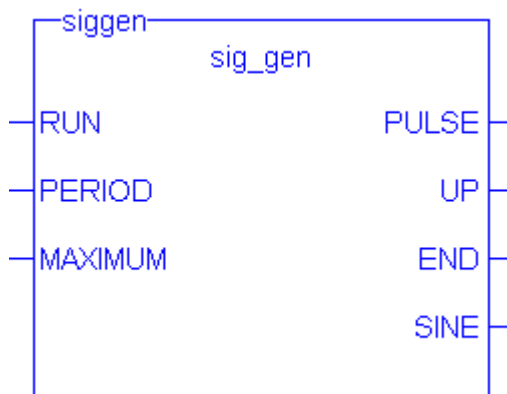
MAXIMUM : DINT Maximum growth during the signal period

#### 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.

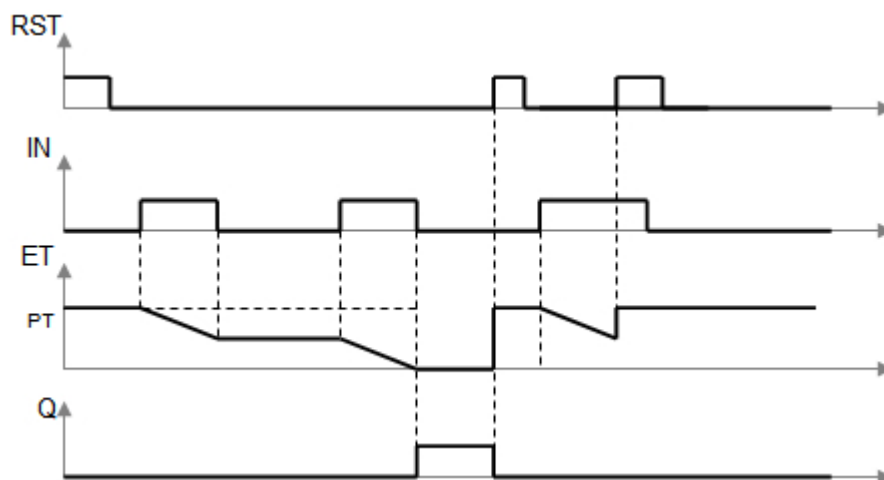
#### 3.9.5.1 Inputs

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

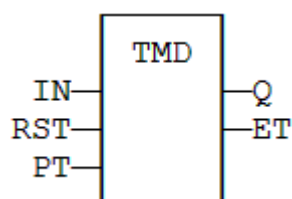
(\* MyTimer is a declared instance of TMD function block \*)

MyTimer (IN, RST, PT);

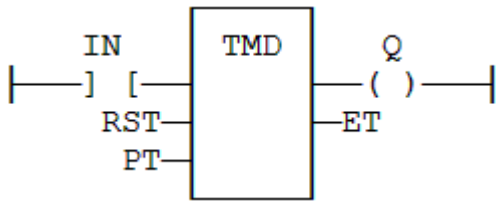
Q := MyTimer.Q;

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** PLCopen ✔

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

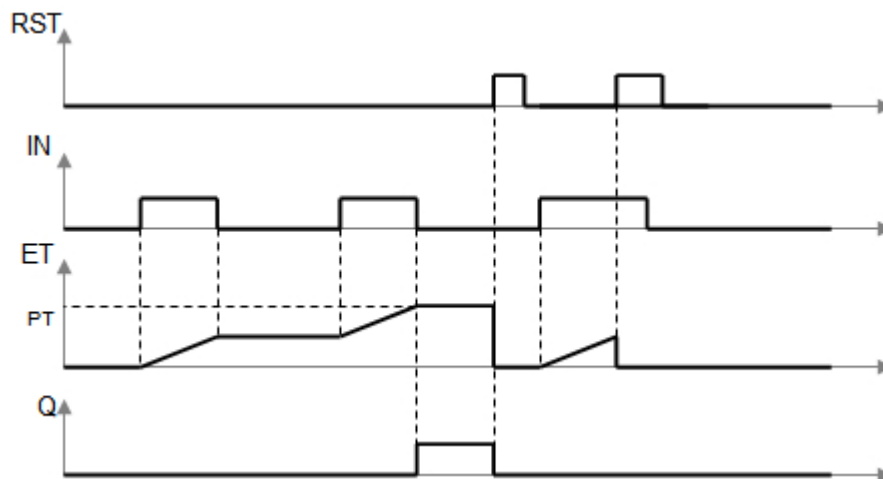
**3.9.6.1 Inputs**

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

**3.9.6.2 Outputs**

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

<b>Q</b>	BOOL	Timer elapsed output signal
<b>ET</b>	TIME	Elapsed time
<b>ETsec</b>	UDINT	Elapsed time. (TMU - seconds)

**3.9.6.3 Time diagram****3.9.6.4 Remarks**

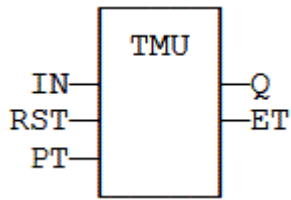
The timer counts up when the IN input is TRUE. It stops when the programmed time is elapsed. The timer is reset when the RST input is TRUE. It is not reset when IN is false.

**3.9.6.5 ST Language**

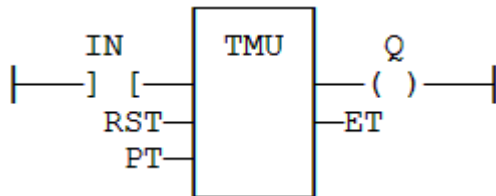
(\* MyTimer is a declared instance of TMU function block \*)

```
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.*

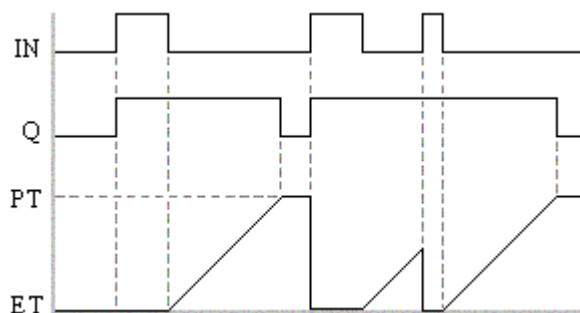
#### 3.9.7.1 Inputs

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

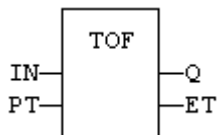
In FFLD language, the input rung is the IN command. The output rung is Q the output signal.

#### 3.9.7.5 ST Language

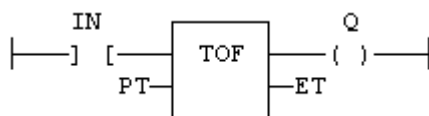
(\* MyTimer is a declared instance of TOF function block \*)

```
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.*

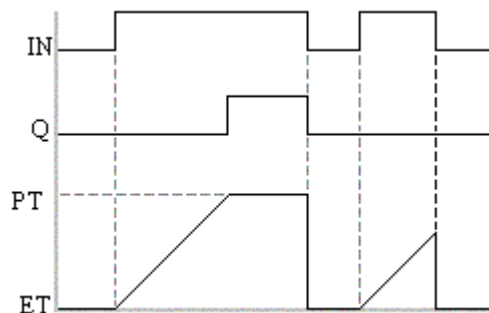
#### 3.9.8.1 Inputs

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



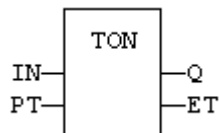
#### 3.9.8.4 Remarks

- 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.

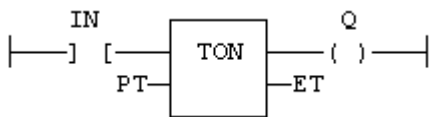
#### 3.9.8.5 ST Language

```
(* 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.

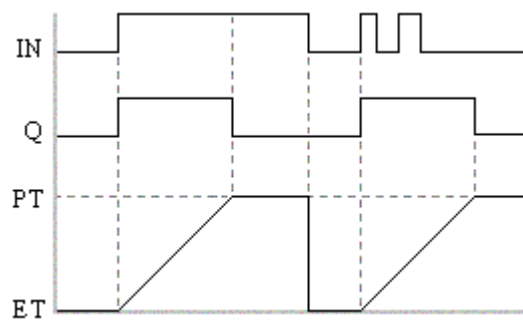
#### 3.9.9.1 Inputs

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

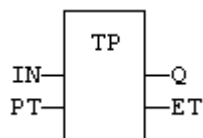
(\* MyTimer is a declared instance of TP function block \*)

MyTimer (IN, PT);

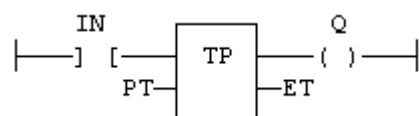
Q := MyTimer.Q;

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:

<a href="#">ABS</a>	absolute value
<a href="#">TRUNC</a>	integer part
<a href="#">LOG</a> , <a href="#">"LN / LNL"</a> (p. 174)	logarithm, natural logarithm
<a href="#">POW</a> , <a href="#">EXPT</a> , <a href="#">"EXP / EXPL"</a> (p. 172)	power
<a href="#">SQRT</a> , <a href="#">"ROOT"</a> (p. 176)	square root, root extraction
<a href="#">SCALELIN</a>	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

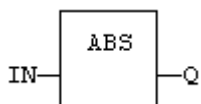
#### 3.10.1.3 Remarks

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

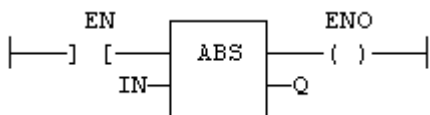
Q := ABS (IN);

#### 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

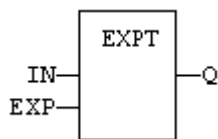
**3.10.2.3 Remarks**

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.

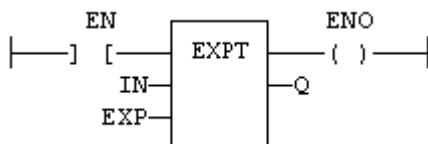
**3.10.2.4 ST Language**

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.

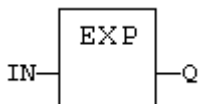
**3.10.3.3 Remarks**

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.

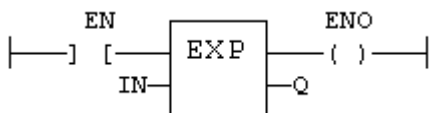
**3.10.3.4 ST Language**

```
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

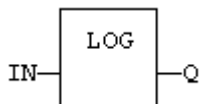
**3.10.4.3 Remarks**

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.

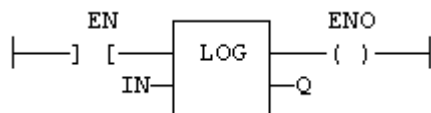
**3.10.4.4 ST Language**

```
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

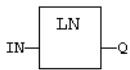
**3.10.5.3 Remarks**

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.

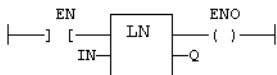
**3.10.5.4 ST Language**

```
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

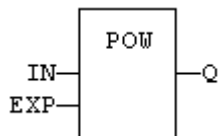
Alternatively, in ST language, the "\*\*\*\*" operator can be used. In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

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

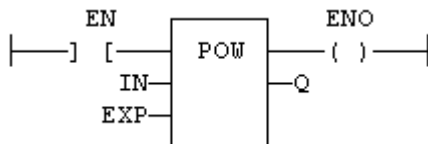
Q := POW (IN, EXP);  
Q := IN \*\* EXP;

#### 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

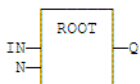
**3.10.7.3 Remarks**

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.

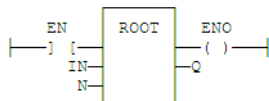
**3.10.7.4 ST Language**

```
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 ✔

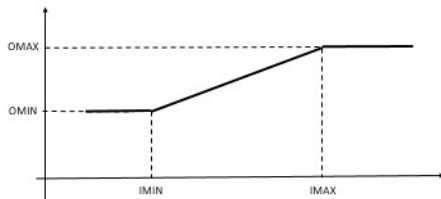
*Function* - Scaling - linear conversion.

#### 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
$IMIN \geq IMAX$	$= IN$
$IN < IMIN$	$= IMIN$
$IN > IMAX$	$= IMAX$
other	$= OMIN + IN * (OMAX - OMIN) / (IMAX - IMIN)$

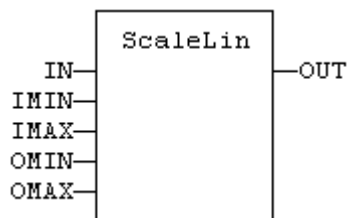
#### 3.10.8.4 Remarks

- 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

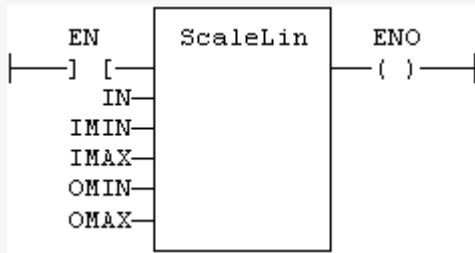
```
OUT := ScaleLin (IN, IMIN, IMAX, OMIN, OMAX);
```

#### 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

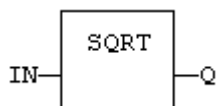
**3.10.9.3 Remarks**

In FFD 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.

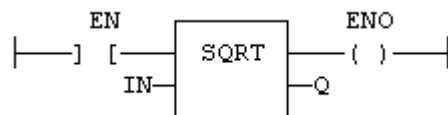
**3.10.9.4 ST Language**

Q := SQRT (IN);

**3.10.9.5 FBD Language****3.10.9.6 FFD 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

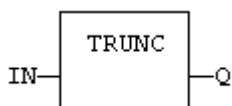
**3.10.10.3 Remarks**

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.

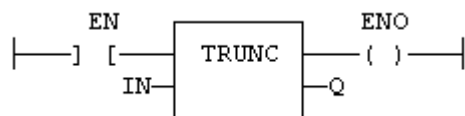
**3.10.10.4 ST Language**

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:

<a href="#">SIN</a>	sine
<a href="#">COS</a>	cosine
<a href="#">TAN</a>	tangent
<a href="#">ASIN</a>	arc-sine
<a href="#">ACOS</a>	arc-cosine
<a href="#">ATAN</a>	arc-tangent
<a href="#">ATAN2</a>	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

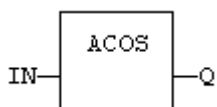
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

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

#### 3.11.1.4 ST Language

Q := ACOS (IN);

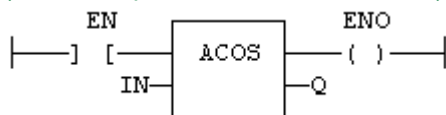
#### 3.11.1.5 FBD Language



#### 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

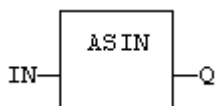
**3.11.2.3 Remarks**

In FFD 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.

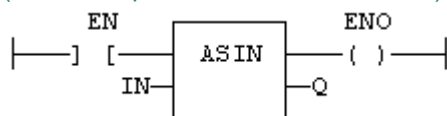
**3.11.2.4 ST Language**

Q := ASIN (IN);

**3.11.2.5 FBD Language****3.11.2.6 FFD 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

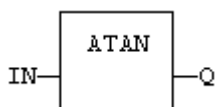
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

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

#### 3.11.3.4 ST Language

Q := ATAN (IN);

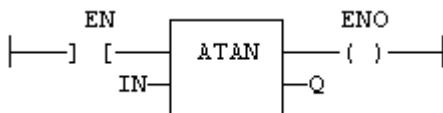
#### 3.11.3.5 FBD Language



#### 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

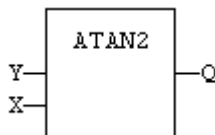
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

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

#### 3.11.4.4 ST Language

Q := ATAN2 (IN);

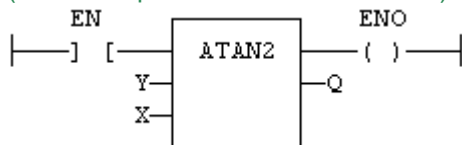
#### 3.11.4.5 FBD Language



#### 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

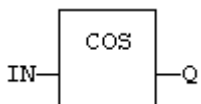
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

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

#### 3.11.5.4 ST Language

Q := COS (IN);

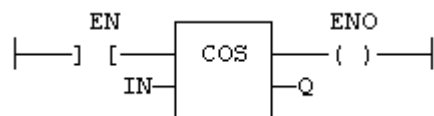
#### 3.11.5.5 FBD Language



#### 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

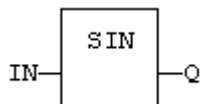
**3.11.6.3 Remarks**

In FFD 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.

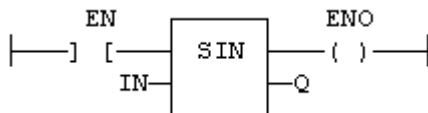
**3.11.6.4 ST Language**

Q := SIN (IN);

**3.11.6.5 FBD Language****3.11.6.6 FFD 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

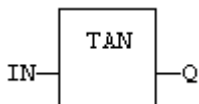
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

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

#### 3.11.7.4 ST Language

Q := TAN (IN);

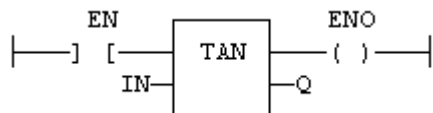
#### 3.11.7.5 FBD Language



#### 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  
                   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

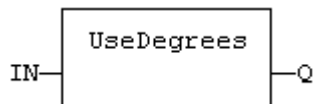
This function sets the working unit for the following functions:

<a href="#">SIN</a>	sine
<a href="#">COS</a>	cosine
<a href="#">TAN</a>	tangent
<a href="#">ASIN</a>	arc-sine
<a href="#">ACOS</a>	arc-cosine
<a href="#">ATAN</a>	arc-tangent
<a href="#">ATAN2</a>	arc-tangent of Y / X

#### 3.11.8.4 ST Language

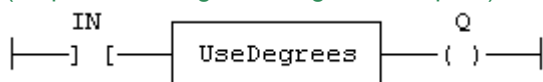
Q := UseDegrees (IN);

#### 3.11.8.5 FBD Language



#### 3.11.8.6 FLD 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:

<a href="#">+</a>	concatenation of strings
<a href="#">CONCAT</a>	concatenation of strings
<a href="#">MLEN</a>	get string length
<a href="#">DELETE</a>	delete characters in a string
<a href="#">INSERT</a>	insert characters in a string
<a href="#">FIND</a>	find characters in a string
<a href="#">REPLACE</a>	replace characters in a string
<a href="#">LEFT</a>	extract a part of a string on the left
<a href="#">RIGHT</a>	extract a part of a string on the right
<a href="#">MID</a>	extract a part of a string
<a href="#">CHAR</a>	build a single character string
<a href="#">ASCII</a>	get the ASCII code of a character within a string
<a href="#">ATOH</a>	converts string to integer using hexadecimal basis
<a href="#">HTOA</a>	converts integer to string using hexadecimal basis
<a href="#">CRC16</a>	CRC16 calculation
<a href="#">ArrayToString</a>	copies elements of an SINT array to a STRING
<a href="#">StringToArray</a>	copies characters of a STRING to an SINT array

Other functions are available for managing string tables as resources:

<a href="#">StringTable</a>	Select the active string table resource
<a href="#">LoadString</a>	Load a string from the active string table

### 3.12.1 ArrayToString / ArrayToStringU PLCopen ✓

*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

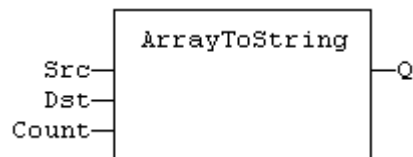
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

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);

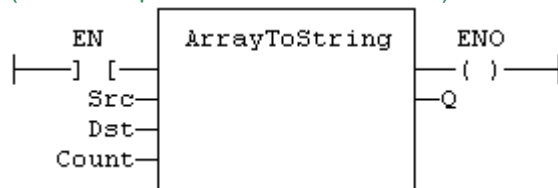
#### 3.12.1.5 FBD Language



#### 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

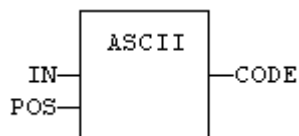
#### 3.12.2.3 Remarks

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);

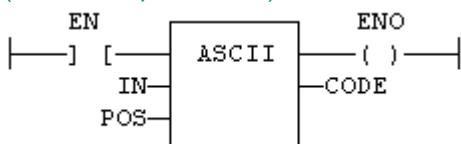
#### 3.12.2.5 FBD Language



#### 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

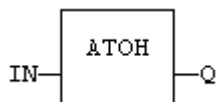
The function is case insensitive. The result is 0 for an empty string. The conversion stops before the first invalid character. In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

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

#### 3.12.3.5 ST Language

Q := ATOH (IN);

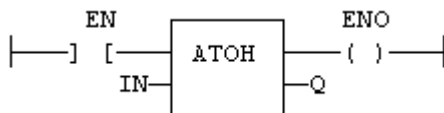
#### 3.12.3.6 FBD Language



#### 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

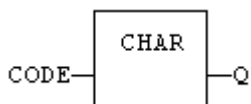
#### 3.12.4.3 Remarks

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);

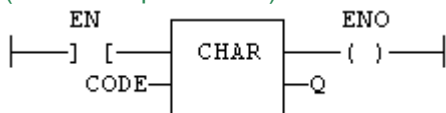
#### 3.12.4.5 FBD Language



#### 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

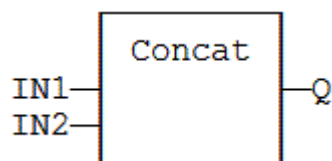
In FBD or FFLD language, the block can have up to 16 inputs. In IL or ST, the function accepts a variable number of inputs (at least 2).

Note that you also can use the "+" operator to concatenate strings.

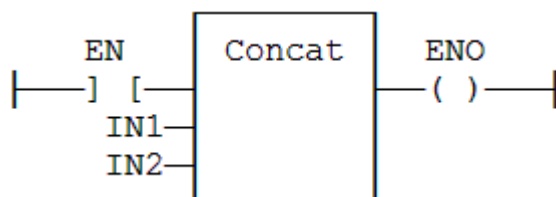
#### 3.12.5.4 ST Language

```
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.

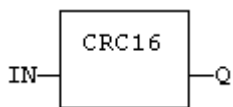
**3.12.6.3 Remarks**

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.

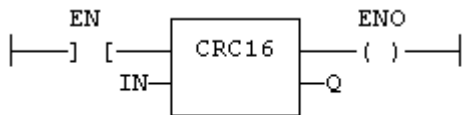
**3.12.6.4 ST Language**

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.

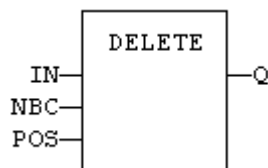
**3.12.7.3 Remarks**

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 commas.

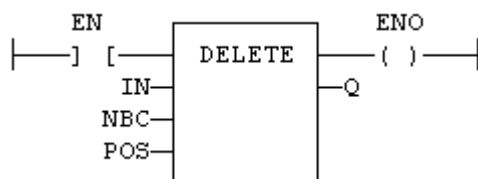
**3.12.7.4 ST Language**

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

*Function* - Find position of characters in a string.

#### 3.12.8.1 Inputs

<b>IN</b>	<b>STRING</b>	Character string
<b>STR</b>	<b>STRING</b>	Specific characters to search for within the STRING

#### 3.12.8.2 Outputs

<b>POS</b>	<b>DINT</b>	Position of the first character of STR in IN, or 0 if not found.
------------	-------------	--

#### 3.12.8.3 Remarks

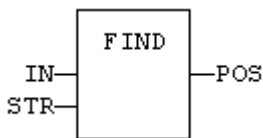
The first valid character position is 1. A return value of 0 means that the STR string has not been found. Search is case sensitive. In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

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);

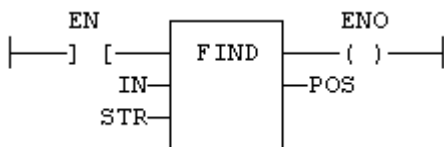
#### 3.12.8.5 FBD Language



#### 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'

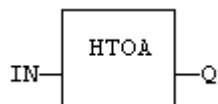
**3.12.9.4 Remarks**

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.

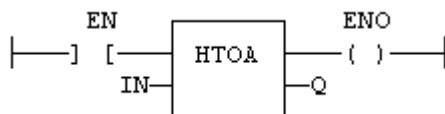
**3.12.9.5 ST Language**

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.

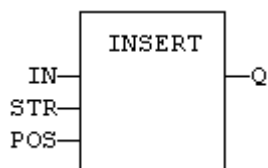
**3.12.10.3 Remarks**

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.

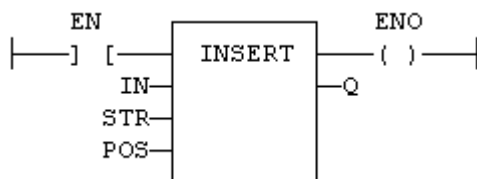
**3.12.10.4 ST Language**

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.

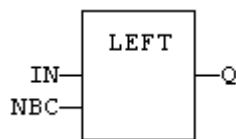
**3.12.11.3 Remarks**

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.

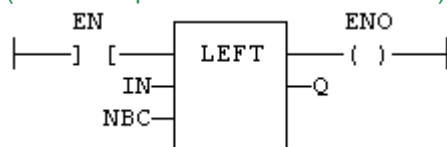
**3.12.11.4 ST Language**

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

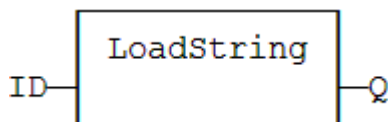
This function loads a string from the active string table and stores it into a STRING variable. The [StringTable\(\)](#) function is used for selecting the active string table.

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.

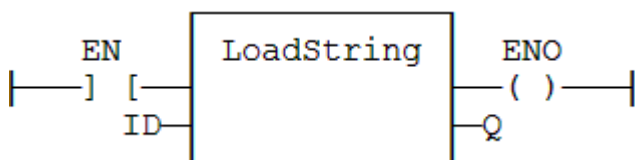
#### 3.12.12.4 ST Language

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.

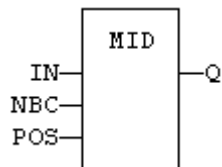
**3.12.13.3 Remarks**

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 commas.

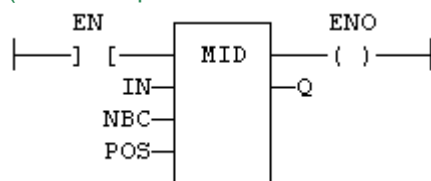
**3.12.13.4 ST Language**

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.

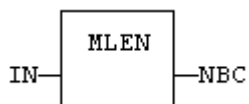
**3.12.14.3 Remarks**

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.

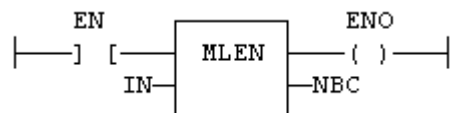
**3.12.14.4 ST Language**

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

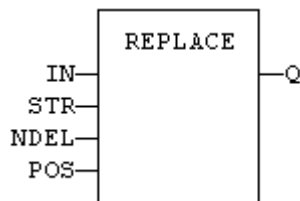
The first valid character position is 1. In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

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 commas.

#### 3.12.15.4 ST Language

Q := REPLACE (IN, STR, NDEL, POS);

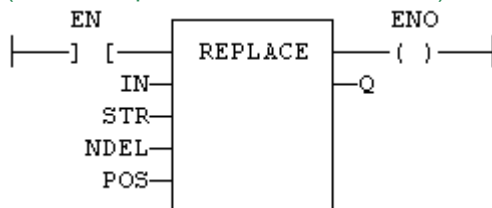
#### 3.12.15.5 FBD Language



#### 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.

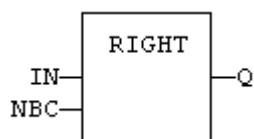
**3.12.16.3 Remarks**

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.

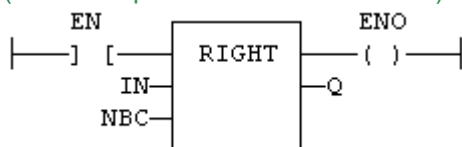
**3.12.16.4 ST Language**

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

*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.

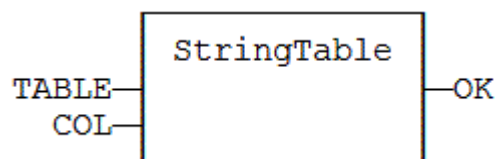
Arguments must be constant string expressions and must fit to a declared string table and a valid column name within this table.

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.

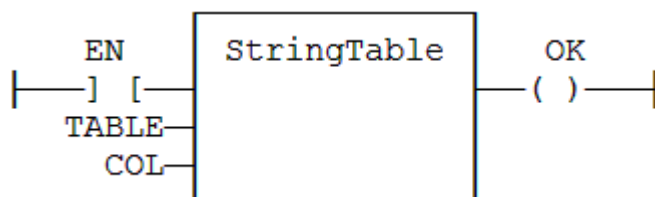
#### 3.12.17.4 ST Language

```
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

### 3.12.18 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.

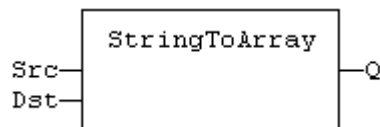
In IL, the input must be loaded in the current result before calling the function.

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);

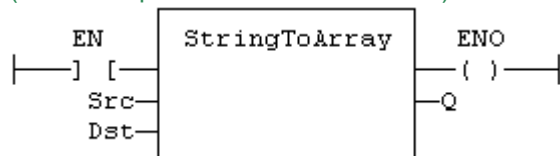
#### 3.12.18.5 FBD Language



#### 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](#)



### 3.13 UDP Functions for PDMM/PCMM & Simulator

UDP ([User Datagram Protocol](#)) is a communications protocol which allows computers to exchange messages across an IP network. The UDP functions listed below provide a KAS controller to communicate with a remote PC or another KAS controller over an Ethernet network.

When a UDP packet is sent to a broadcast address such as '255.255.255.255', the AKD PDMM or PCMM automatically converts the given broadcast address to the broadcast address of its Ethernet interface. For example if the controller's IP address is 192.168.1.10 and the subnet mask is 255.255.255.0, then the controller's Ethernet interface broadcast address is 192.168.1.255.

Function	Description
"udpAddrMake" (p. 210)	Build an address buffer for UDP functions
"udpClose" (p. 212)	Close a socket
"udpCreate" (p. 214)	Create a UDP socket
"udpIsValid" (p. 216)	Test if a socket is valid
"udpRcvFrom" (p. 218)	Receive a telegram
"udpRcvFromArray" (p. 220)	Receive a byte array through UDP
udpRcvFromVar	Receives the contents of a variable through UDP
"udpSendTo" (p. 222)	Send a telegram
"udpSendToArray" (p. 224)	Send a byte array through UDP
udpSendToVar	Sends the contents of a local variable through UDP

#### TIP

See Wikipedia for more information on the [UDP protocol](#).

### 3.13.1 udpAddrMake

#### 3.13.1.1 Description

This function builds an address buffer for UDP functions. This function is required for building an internal "UDP" address to be passed to the "udpSendTo" (p. 222) function in case of UDP client processing.

#### 3.13.1.2 Arguments

##### 3.13.1.3.1 Input

<b>En</b>	<b>Description</b>	Execute the function
	<b>Data type</b>	BOOL
	<b>Range</b>	[0,1]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>IPaddr</b>	<b>Description</b>	IP address in the form XXX.XXX.XXX.XXX
	<b>Data type</b>	STRING
	<b>Range</b>	[0.0.0.0,255.255.255.255]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>port</b>	<b>Description</b>	IP port number
	<b>Data type</b>	DINT
	<b>Range</b>	[0,+65535]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>add[]</b>	<b>Description</b>	Buffer where to store the UDP address (filled on output)
	<b>Data type</b>	USINT
	<b>Range</b>	[0,32]
	<b>Unit</b>	n/a
	<b>Default</b>	—

### 3.13.1.4.2 Output

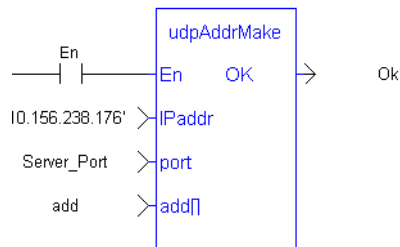
<b>OK</b>	<b>Description</b>	Returns true when the function successfully executes. See <a href="#">Function - General rules</a> .
	<b>Data type</b>	BOOL
	<b>Unit</b>	n/a

### 3.13.1.5 Examples

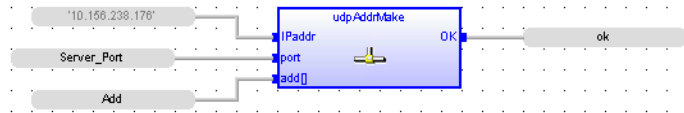
#### 3.13.1.6.1 Structured Text

```
bAddrMake := udpAddrMake('10.156.238.176', Server_Port, add);
//server details
```

#### 3.13.1.7.2 Ladder Diagram



#### 3.13.1.8.3 Function Block Diagram



### 3.13.2 udpClose PLCopen

#### 3.13.2.1 Description

This function closes a socket.

#### 3.13.2.2 Arguments

##### 3.13.2.3.1 Input

<b>En</b>	<b>Description</b>	Execute the function
	<b>Data type</b>	BOOL
	<b>Range</b>	[0,1]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>sock</b>	<b>Description</b>	ID of the socket
	<b>Data type</b>	DINT
	<b>Range</b>	[0,+65535]
	<b>Unit</b>	n/a
	<b>Default</b>	—

##### 3.13.2.4.2 Output

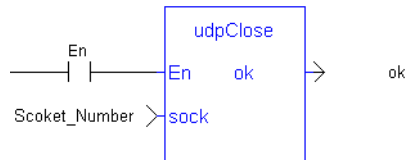
<b>OK</b>	<b>Description</b>	Returns true when the function successfully executes. See <a href="#">Function - General rules</a> .
	<b>Data type</b>	BOOL
	<b>Unit</b>	n/a

### 3.13.2.5 Examples

#### 3.13.2.6.1 Structured Text

```
udpClose(Socket_Number); //Close socket
```

#### 3.13.2.7.2 Ladder Diagram



#### 3.13.2.8.3 Function Block Diagram



### 3.13.3 udpCreate

#### 3.13.3.1 Description

This function creates a UDP socket.

#### 3.13.3.2 Arguments

##### 3.13.3.3.1 Input

<b>En</b>	<b>Description</b>	Execute the function
	<b>Data type</b>	BOOL
	<b>Range</b>	[0,1]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>port</b>	<b>Description</b>	UDP port number to be attached to the server socket or 0 for a client socket.
	<b>Data type</b>	DINT
	<b>Range</b>	[0,+65535]
	<b>Unit</b>	n/a
	<b>Default</b>	—

##### 3.13.3.4.2 Output

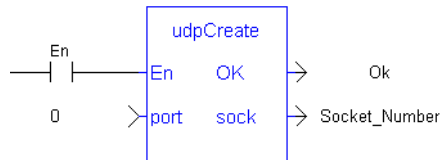
<b>OK</b>	<b>Description</b>	Returns true when the function successfully executes. See <a href="#">Function - General rules</a> .
	<b>Data type</b>	BOOL
	<b>Unit</b>	n/a
<b>sock</b>	<b>Description</b>	ID of the new socket
	<b>Data type</b>	DINT
	<b>Unit</b>	n/a

### 3.13.3.5 Examples

#### 3.13.3.6.1 Structured Text

```
Socket_Number := udpCreate(Client_Port); //create a socket
```

#### 3.13.3.7.2 Ladder Diagram



#### 3.13.3.8.3 Function Block Diagram



### 3.13.4 udplsValid

#### 3.13.4.1 Description

This function states whether a socket is valid or not.

#### 3.13.4.2 Arguments

##### 3.13.4.3.1 Input

<b>En</b>	<b>Description</b>	Execute the function
	<b>Data type</b>	BOOL
	<b>Range</b>	[0,1]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>sock</b>	<b>Description</b>	ID of the socket
	<b>Data type</b>	DINT
	<b>Range</b>	[0,+65535]
	<b>Unit</b>	n/a
	<b>Default</b>	—

##### 3.13.4.4.2 Output

<b>OK</b>	<b>Description</b>	Returns true when the function successfully executes. See <a href="#">Function - General rules</a> .
	<b>Data type</b>	BOOL
	<b>Unit</b>	n/a

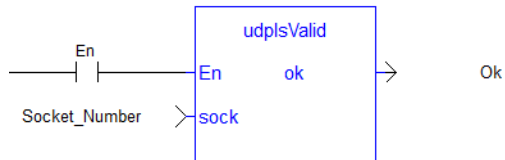


### 3.13.4.5 Examples

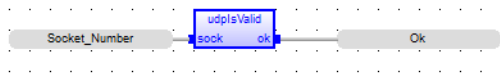
#### 3.13.4.6.1 Structured Text

```
bIsValid := udpIsValid(Socket_Number); //Valid socket?
```

#### 3.13.4.7.2 Ladder Diagram



#### 3.13.4.8.3 Function Block Diagram



### 3.13.5 udpRcvFrom

#### 3.13.5.1 Description

This function receives a UDP telegram. If the characters are received, the function fills the ADD argument with the internal "UDP" of the sender. This buffer can then be passed to the "[udpSendTo](#)" (p. 222) function to send the answer.

#### 3.13.5.2 Arguments

##### 3.13.5.3.1 Input

<b>En</b>	<b>Description</b>	Execute the function
	<b>Data type</b>	BOOL
	<b>Range</b>	[0,1]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>sock</b>	<b>Description</b>	ID of the socket
	<b>Data type</b>	DINT
	<b>Range</b>	[0,+65535]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>nb</b>	<b>Description</b>	Maximum number of characters received
	<b>Data type</b>	DINT
	<b>Range</b>	[0,+65535]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>add[]</b>	<b>Description</b>	Buffer containing the UDP address of the transmitter (filled on output)
	<b>Data type</b>	USINT
	<b>Range</b>	[0,32]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>data</b>	<b>Description</b>	Buffer where to store received characters
	<b>Data type</b>	STRING

<b>Range</b>	[0,255]
<b>Unit</b>	n/a
<b>Default</b>	—

### 3.13.5.4.2 Output

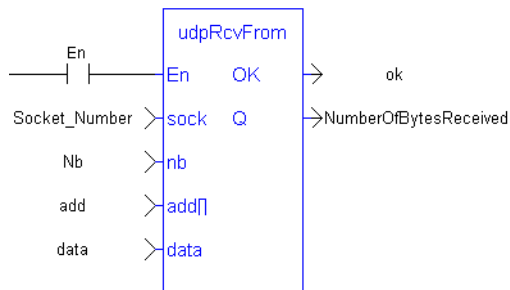
<b>OK</b>	<b>Description</b>	Returns true when the function successfully executes. See <a href="#">Function - General rules</a> .
	<b>Data type</b>	BOOL
	<b>Unit</b>	n/a
<b>Q</b>	<b>Description</b>	Actual number of received characters
	<b>Data type</b>	DINT
	<b>Unit</b>	n/a

### 3.13.5.5 Examples

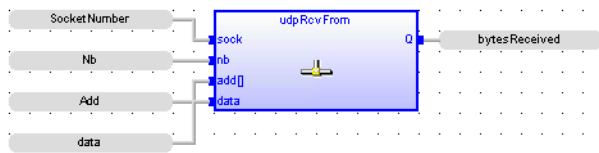
#### 3.13.5.6.1 Structured Text

```
ReceivedBytes := udpRcvFrom(Socket_Number, 5, add, data); //Read
the position
```

#### 3.13.5.7.2 Ladder Diagram



#### 3.13.5.8.3 Function Block Diagram



### 3.13.6 udpRcvFromArray

#### 3.13.6.1 Description

This function receives an array of bytes.

#### 3.13.6.2 Arguments

##### 3.13.6.3.1 Input

<b>En</b>	<b>Description</b>	Execute the function
	<b>Data type</b>	BOOL
	<b>Range</b>	[0,1]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>sock</b>	<b>Description</b>	Socket number, return value from " <a href="#">udpCreate</a> " (p. 214)
	<b>Data type</b>	DINT
	<b>Range</b>	[0,+65535]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>nb</b>	<b>Description</b>	Number of bytes to be transferred
	<b>Data type</b>	DINT
	<b>Range</b>	[0,+65535]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>add[]</b>	<b>Description</b>	Array which contains information about the server
	<b>Data type</b>	USINT
	<b>Range</b>	[0,32]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>data[]</b>	<b>Description</b>	Array of bytes to be transferred
	<b>Data type</b>	USINT
	<b>Range</b>	[0,+65535]
	<b>Unit</b>	n/a

<b>Default</b>	—
----------------	---

### 3.13.6.4.2 Output

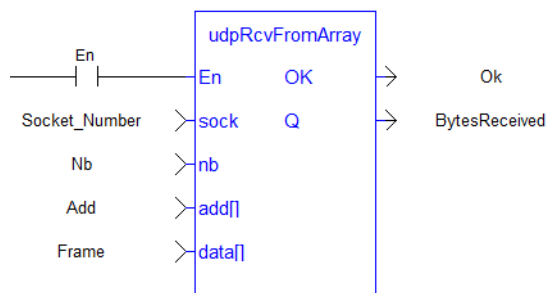
<b>OK</b>	<b>Description</b>	Returns true when the function successfully executes. See <a href="#">Function - General rules</a> .
	<b>Data type</b>	BOOL
	<b>Unit</b>	n/a
<b>Q</b>	<b>Description</b>	Number of bytes received
	<b>Data type</b>	DINT
	<b>Unit</b>	n/a

### 3.13.6.5 Examples

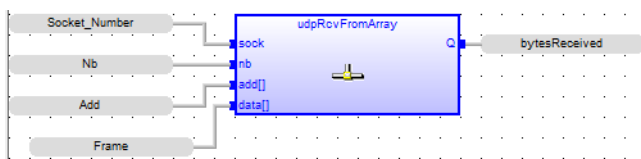
#### 3.13.6.6.1 Structured Text

```
BytesReceived := udpRcvFromArray(Socket_Number, nb, add, Frame);
```

#### 3.13.6.7.2 Ladder Diagram



#### 3.13.6.8.3 Function Block Diagram



### 3.13.7 udpSendTo

#### 3.13.7.1 Description

This function sends UDP data to a server.

#### 3.13.7.2 Arguments

##### 3.13.7.3.1 Input

<b>En</b>	<b>Description</b>	Execute the function
	<b>Data type</b>	BOOL
	<b>Range</b>	[0,1]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>sock</b>	<b>Description</b>	ID of the client socket
	<b>Data type</b>	DINT
	<b>Range</b>	[0,+65535]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>nb</b>	<b>Description</b>	Number of bytes of data to send
	<b>Data type</b>	DINT
	<b>Range</b>	[0,65535]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>add[]</b>	<b>Description</b>	Buffer containing the UDP address
	<b>Data type</b>	USINT
	<b>Range</b>	[0,32]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>data</b>	<b>Description</b>	The characters to send
	<b>Data type</b>	STRING
	<b>Range</b>	[0,255]
	<b>Unit</b>	n/a

<b>Default</b>	—
----------------	---

### 3.13.7.4.2 Output

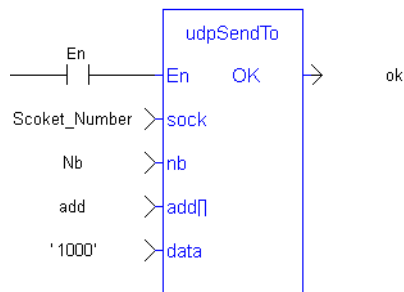
<b>OK</b>	<b>Description</b>	Returns true when the function successfully executes. See <a href="#">Function - General rules</a> .
	<b>Data type</b>	BOOL
	<b>Unit</b>	n/a

### 3.13.7.5 Examples

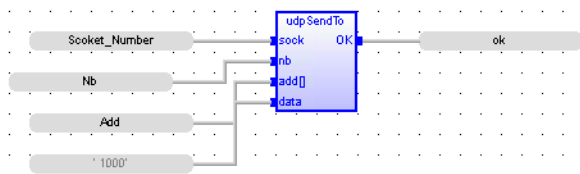
#### 3.13.7.6.1 Structured Text

```
bUdpSendTo := udpSendTo(Socket_Number, 5, add, '1000');
```

#### 3.13.7.7.2 Ladder Diagram



#### 3.13.7.8.3 Function Block Diagram



### 3.13.8 udpSendToArray

#### 3.13.8.1 Description

This function sends an array of bytes.

#### 3.13.8.2 Arguments

##### 3.13.8.3.1 Input

<b>En</b>	<b>Description</b>	Execute the function
	<b>Data type</b>	BOOL
	<b>Range</b>	[0,1]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>sock</b>	<b>Description</b>	Socket number, return value from <a href="#">"udpCreate" (p. 214)</a>
	<b>Data type</b>	DINT
	<b>Range</b>	[0,+65535]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>nb</b>	<b>Description</b>	Number of bytes to be transferred
	<b>Data type</b>	DINT
	<b>Range</b>	[0,+65535]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>add[]</b>	<b>Description</b>	Array which contains information about the server
	<b>Data type</b>	USINT
	<b>Range</b>	[0,32]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>data[]</b>	<b>Description</b>	Array of bytes to be transferred
	<b>Data type</b>	USINT
	<b>Range</b>	[0,+65535]
	<b>Unit</b>	n/a



<b>Default</b>	—
----------------	---

### 3.13.8.4.2 Output

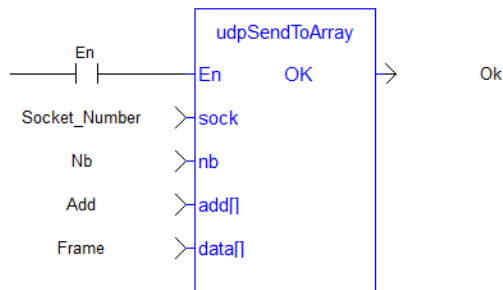
<b>OK</b>	<b>Description</b>	Returns true when the function successfully executes. See <a href="#">Function - General rules</a> .
	<b>Data type</b>	BOOL
	<b>Unit</b>	n/a

### 3.13.8.5 Examples

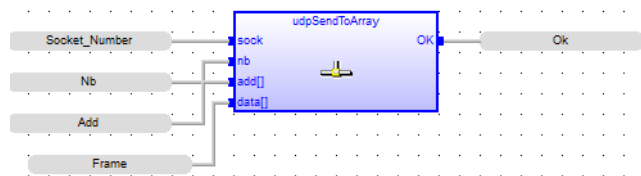
#### 3.13.8.6.1 Structured Text

```
Success := udpSendToArray(Socket_Number, nb, add, Frame);
```

#### 3.13.8.7.2 Ladder Diagram



#### 3.13.8.8.3 Function Block Diagram



## 4 PLC Advanced Libraries

Below are the standard blocks that perform advanced operations.

### Analog signal processing:

Block	Description
<a href="#">Average / AverageL</a>	Calculate the average of signal samples
<a href="#">Integral</a>	Calculate the integral of a signal
<a href="#">Derivate</a>	Derive a signal
<a href="#">PID</a>	PID loop
<a href="#">Ramp</a>	Ramp signal
Rand	Give a Random value modulo the input value
<a href="#">Lim_Alm</a>	Low / High level detection
<a href="#">Hyster</a> , HysterAcc	Hysteresis calculation
<a href="#">SigPlay</a>	Play an analog signal from a resource
<a href="#">SigScale</a>	Get a point from a signal resource
<a href="#">CurveLin</a>	Linear interpolation on a curve
<a href="#">SurfLin</a>	Linear interpolation on a surface

### Alarm management:

Block	Description
<a href="#">Lim_Alm</a>	Low / High level detection
<a href="#">Alarm_M</a>	Alarm with manual reset
"ALARM_A" (p. 228)	Alarm with automatic reset

### Data collections and serialization:

Block	Description
<a href="#">StackInt</a>	Stack of integers
<a href="#">FIFO</a>	"First in / first out" list
<a href="#">LIFO</a>	"Last in / first out" stack
"SerializeIn" (p. 285)	Extract data from a binary frame.
"SerializeOut" (p. 287)	Write data to a binary frame.

### Data Logging:

Block	Description
<a href="#">LogFileCSV</a>	Log values of variables to a CSV file

### Special operations:

Block	Description
<a href="#">GetSysInfo</a>	Get system information
Printf	Trace messages
CycleStop	Sets the application in cycle stepping mode
FatalStop	Breaks the cycle and stop with fatal error
<a href="#">EnableEvents</a>	Enable / disable produced events for binding
<a href="#">ApplyRecipeColumn</a>	Apply the values of a column from a recipe file

Block	Description
<a href="#">VLID</a>	Get the ID of an embedded list of variables
<a href="#">SigID</a>	Get the ID of a signal resource

**Communication:**

[AS-interface](#)

**Others:**

[Real Time Clock](#)

"File Management Functions" (p. 251)

## 4.1 ALARM\_A PLCopen ✓

Function Block - Alarm with automatic reset

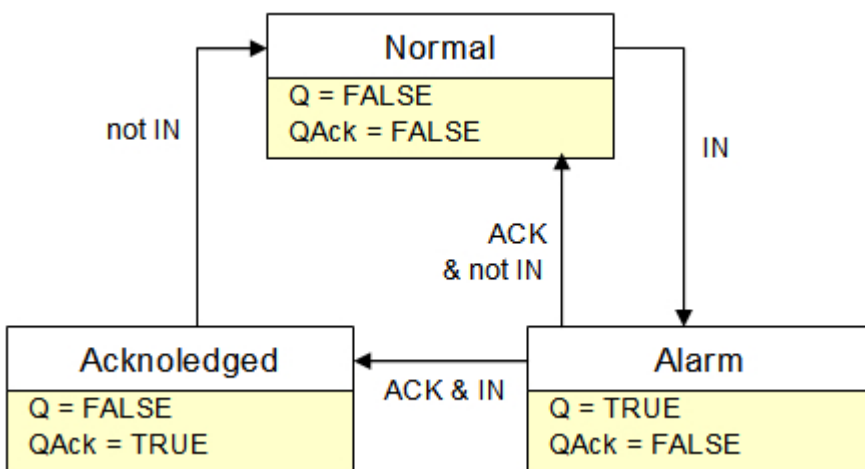
### 4.1.1 Inputs

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

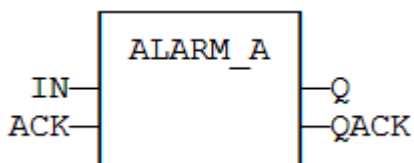
Combine this block with the [LIM\\_ALARM](#) block for managing analog alarms.

### 4.1.5 ST Language

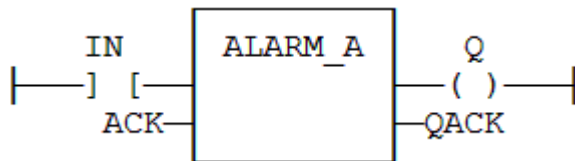
```

    (* 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



### 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*

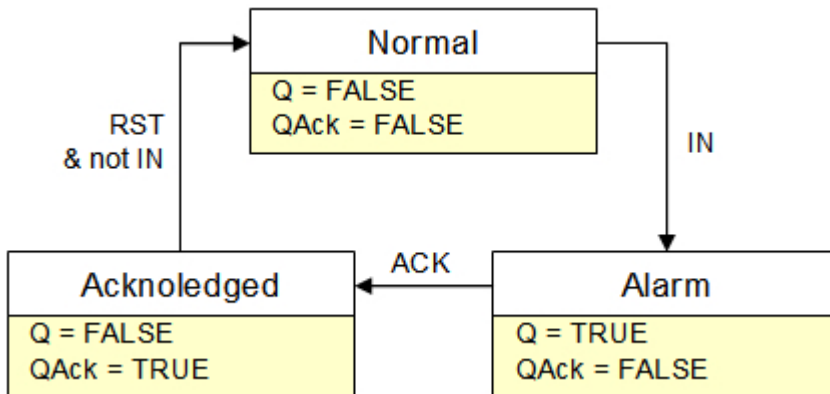
### 4.2.1 Inputs

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

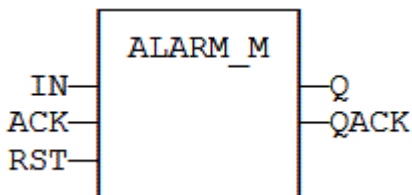
Combine this block with the [LIM\\_ALARM](#) block for managing analog alarms.

### 4.2.5 ST Language

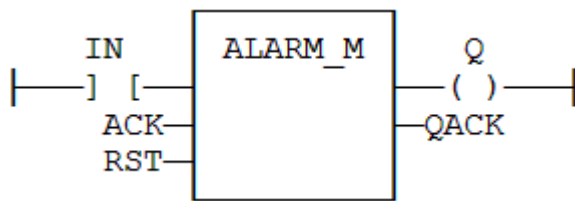
```

(* 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

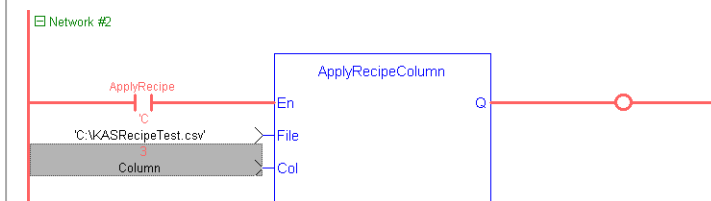
Example of CSV file with five variables and five set of values

```
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

Dictionary

Controller: PLC  Track Selection

Name	Value	Type
Global variables		
TravelSpeed	400.0000...	LREAL
MasterAbsPos	135.0000...	LREAL
MasterDeltaPos	270.0000...	LREAL
MachineSpeed	200.0000...	LREAL
Axis1Status	447	DINT
Axis2Status	447	DINT
MachineState	1	DINT

### 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.

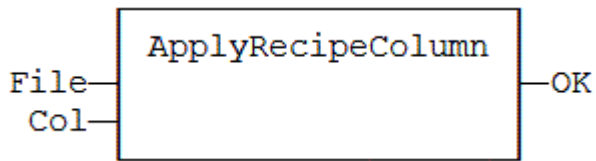
#### ⚠ IMPORTANT

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

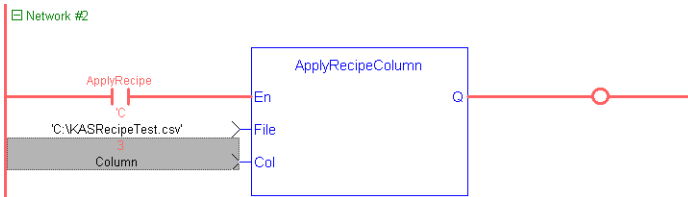
```
OK := ApplyRecipeColumn ('MyFile.rcp', COL);
```

### 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.

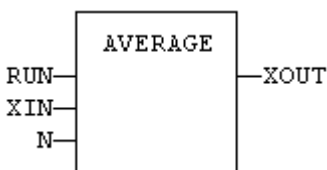
#### **NOTE**

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

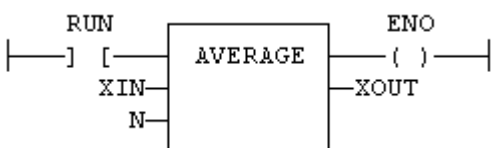
```
(* MyAve is a declared instance of AVERAGE function block *)
MyAve (RUN, XIN, N);
XOUT := MyAve.XOUT;
```

### 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

*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.

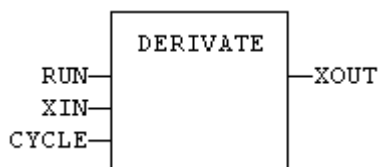
### 4.7.4 ST Language

(\* MyDerv is a declared instance of DERIVATE function block \*)

MyDerv (RUN, XIN, CYCLE);

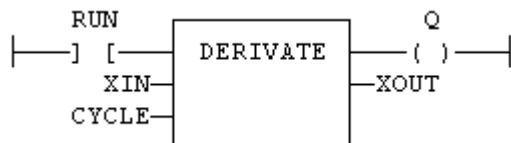
XOUT := MyDerv.XOUT;

### 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\\_ALARM](#) [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

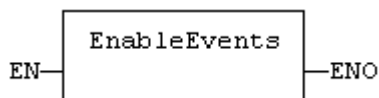
Production is enabled when the application starts. The first production will be operated after the first cycle. So to disable events since the beginning, you must call `EnableEvents (FALSE)` in the very first cycle.

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);

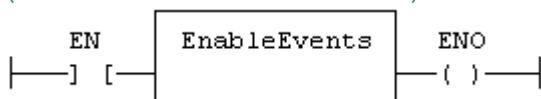
### 4.8.5 FBD Language



### 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

<b>PUSH</b>	<b>BOOL</b>	Push a new value (on rising edge)
<b>POP</b>	<b>BOOL</b>	Pop a new value (on rising edge)
<b>RST</b>	<b>BOOL</b>	Reset the list
<b>IN</b>	<b>ANY</b>	Value to be pushed
<b>@Tail</b>	<b>ANY</b>	Value of the oldest pushed value - <i>updated after call!</i>
<b>Buf[]</b>	<b>ANY</b>	Array for storing values

### 4.9.2 Outputs

<b>EMPTY</b>	<b>BOOL</b>	TRUE if the list is empty
<b>OFLO</b>	<b>BOOL</b>	TRUE if overflow on a PUSH command
<b>Count</b>	<b>DINT</b>	Number of values in the list
<b>pRead</b>	<b>DINT</b>	Index in the buffer of the oldest pushed value
<b>pWrite</b>	<b>DINT</b>	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

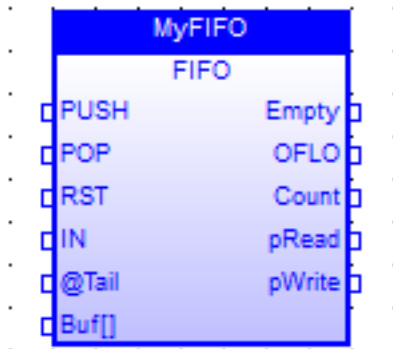
```
(* MyFIFO is a declared instance of FIFO function block *)
MyFIFO (PUSH, POP, RST, IN, @Tail , BUFFER);
EMPTY := MyFIFO.EMPTY;
```

```

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 FilterOrder1 PLCopen ✓

Function block - first order filter

### 4.10.1 Inputs

XIN : REAL	Input analog value
GAIN : REAL	Transformation gain

### 4.10.2 Outputs

XOUT : REAL	Output signal
-------------	---------------

### 4.10.3 Remarks

The operation performed is:

$$\text{Output} = (\text{Input} \times \text{Gain}) + (\text{OutputPrev} * (1-\text{Gain}))$$

The allowed range for the gain is [0.05 .. 1.0]

### 4.10.4 ST Language

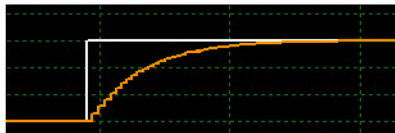
Filt1 is a declared instance of FilterOrder1 function block.

```
Filt1 (rIn, rGain);
Signal := Filt1.Xout;
```

### 4.10.5 Example



Symbol	C...	# Dia...
Main/IN	<input type="text"/>	1
Main/OUT	<input type="text"/>	1



## 4.11 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 remote drive
- Read cam tables into the program from the SD card or a remote drive.
- Store machine operational data in internal PxMM flash memory (retrievable through the web server), the SD card, or a remote drive

*Please note that a remote drive 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](#)" (p. 190).
- If the file is in .CSV format, the following functions can be used: "[LogFileCSV](#)" (p. 267), "[ApplyRecipeColumn](#)" (p. 232)
- 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 functions enable sequential read / write operations in disk files:

Name	Use
"F_AOPEN" (p. 251)	Create or open a file in append mode
"F_CLOSE" (p. 251)	Close an open file
"F_COPY" (p. 251)	Copy a file
"F_DELETE" (p. 252)	Remove a file
"F_EOF" (p. 252)	Test if the end of the file is reached in a file that is open for reading
"F_EXIST" (p. 252)	Test if a file exists
F_GETSIZE (→ p. 252)	Get the size of a file
"F_RENAME" (p. 252)	Rename a file
"F_ROPEN" (p. 253)	Open a file for reading
"F_SEEK" (p. 253)	Set the current position of a file
"F_WOPEN" (p. 253)	Create or reset a file and open it for writing
"FA_READ" (p. 254)	Read a DINT integer from a binary file
"FA_WRITE" (p. 254)	Write a DINT integer to a binary file
"FB_READ" (p. 254)	Read binary data from a file
"FB_WRITE" (p. 254)	Write binary data to a file
"FM_READ" (p. 255)	Read a string value from a text file
"FM_WRITE" (p. 255)	Write a string value to a text file
"SD_ISREADY" (p. 256)	Check that the SD card is ready for read/write
"SD_MOUNT" (p. 256)	Mount an SD card
"SD_UNMOUNT" (p. 256)	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 functions and used in all other calls for identifying the file.

### Related function blocks:

[LogFileCSV](#) log values of variables to a CSV file

**! IMPORTANT**

- These functions can have a serious impact on CPU load and the life expectancy of a flash drive. **It is highly recommended that these be used on an event basis, and not at every PLC cycle.**
- 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 "F\_AOPEN" (p. 251), "F\_ROPEN" (p. 253), and "F\_WOPEN" (p. 253)) is limited by the resources available on the target system.**
- Ensure that each file successfully opened using "F\_AOPEN" (p. 251), "F\_ROPEN" (p. 253), and "F\_WOPEN" (p. 253) has a corresponding "F\_CLOSE" (p. 251) to close the file. Closing the file will release the file ID, making it available for operations on other files.

**NOTE**

- Opening a file with "F\_AOPEN" (p. 251), "F\_ROPEN" (p. 253), and "F\_WOPEN" (p. 253) 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](#)" (p. 247) so you understand hardware-based functional differences.

### 4.11.1 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" \(p. 256\)](#)
3. Ensure the card is accessible using ["SD\\_ISREADY" \(p. 256\)](#) before performing a read or write action
4. Unmount the card, if desired, using ["SD\\_UNMOUNT" \(p. 256\)](#) after performing read/write actions

### 4.11.2 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
<b>AKD PDMM or PCMM</b>			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
<b>Simulator</b>			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
<b>PAC</b>			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 Runtime/Application/userdata

#### See Also:

- ["Shared Directory Path Conventions" \(p. 248\)](#)
- ["SD Card Path Conventions" \(p. 249\)](#)
- ["File Name Warning & Limitations" \(p. 250\)](#)

#### 4.11.2.1 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 or PCMM 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" \(p. 250\)](#)



#### 4.11.2.2 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" \(p. 250\)](#)

#### 4.11.2.3 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.**

### 4.11.3 File Management Functions

The following functions enable sequential read / write operations in disk files:

Name	Use
"F_AOPEN" (p. 251)	Create or open a file in append mode
"F_CLOSE" (p. 251)	Close an open file
"F_COPY" (p. 251)	Copy a file
"F_DELETE" (p. 252)	Remove a file
"F_EOF" (p. 252)	Test if the end of the file is reached in a file that is open for reading
"F_EXIST" (p. 252)	Test if a file exists
F_GETSIZE (→ p. 252)	Get the size of a file
"F_RENAME" (p. 252)	Rename a file
"F_ROPEN" (p. 253)	Open a file for reading
"F_SEEK" (p. 253)	Set the current position of a file
"F_WOPEN" (p. 253)	Create or reset a file and open it for writing
"FA_READ" (p. 254)	Read a DINT integer from a binary file
"FA_WRITE" (p. 254)	Write a DINT integer to a binary file
"FB_READ" (p. 254)	Read binary data from a file
"FB_WRITE" (p. 254)	Write binary data to a file
"FM_READ" (p. 255)	Read a string value from a text file
"FM_WRITE" (p. 255)	Write a string value to a text file
"SD_ISREADY" (p. 256)	Check that the SD card is ready for read/write
"SD_MOUNT" (p. 256)	Mount an SD card
"SD_UNMOUNT" (p. 256)	Unmount an SD card

#### 4.11.3.1 F\_AOPEN PLCopen

Open a file in "append" mode

```
ID := F_AOPEN (PATH);
```

**PATH : STRING** Name of the file. Can include a path name according to target system conventions.

**ID : DINT** ID of the open file or NULL if the file can't be read

If the file does not exist, it is created. If the file already exists, it is opened at the end for appending.

#### 4.11.3.2 F\_CLOSE PLCopen

Close an open file

```
OK := F_CLOSE (ID);
```

**ID : DINT** ID of the open file

**OK : BOOL** return check; TRUE if successful

#### 4.11.3.3 F\_COPY PLCopen

Copy source file contents to a destination file. Please note that large files will take a noticeable amount of time to complete. For example, a 1000KB file takes approximately 0.6 seconds. The output status is set after the file copy operation is complete.

```
OK := F_COPY (SRC, DST);
```

<b>SRC : STRING</b>	Name of the source file (must exist). Can include a pathname according to target system conventions.
<b>DST : STRING</b>	Name of the destination file. Can include a pathname according to target system conventions.
<b>OK : BOOL</b>	TRUE is successful

#### 4.11.3.4 F\_DELETE

Remove a file

```
OK := F_DELETE (PATH);
```

<b>PATH : STRING</b>	Name of the file (must exist). Can include a pathname according to target system conventions.
<b>OK : BOOL</b>	TRUE if successful

#### 4.11.3.5 F\_EOF

Test if the end of a file is encountered

```
OK := F_EOF (ID);
```

<b>ID : DINT</b>	ID of the open file
<b>OK : BOOL</b>	TRUE if the end of the file has been encountered

F\_EOF must be used only for files open in read mode by the F\_ROPEN function.

#### 4.11.3.6 F\_EXIST

Test if file exists

```
OK := F_EXIST (PATH);
```

<b>PATH : STRING</b>	Name of the file, can include a path name according to target system conventions.
<b>OK : BOOL</b>	TRUE if the file exists

#### 4.11.3.7 F\_GETSIZE

Get the size of a file. Note that this function block returns 0 when the file size is zero or if the file is not present.

```
SIZE := F_GETSIZE (PATH);
```

<b>PATH : STRING</b>	Name of the file, can include a path name according to target system conventions
<b>SIZE : DINT</b>	Size of the file in bytes

#### 4.11.3.8 F\_RENAME

Rename a file

```
OK := F_RENAME (PATH, NEWNAME);
```

<b>PATH : STRING</b>	Name of the file (must exist). Can include a pathname according to target system conventions.
----------------------	---

<b>NEWNAME :</b>	New name for the file
<b>STRING</b>	
<b>OK : BOOL</b>	TRUE if successful

**4.11.3.9 F\_ROPEN** PLCopen ✔

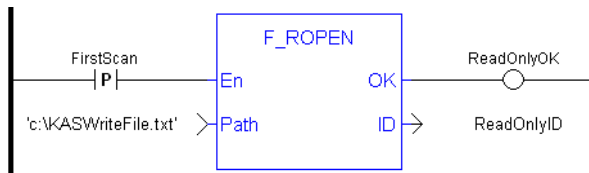
Open a file for reading

**4.11.3.10.1 Example**

**4.11.3.11.2.1 Structured Text**

```
ID := F_ROPEN ( PATH ) ;
```

**4.11.3.12.3.2 Ladder Diagram**



**NOTE**

The positive transition on each file operation FB prevents to open the file every time the program runs (each cycle).

<b>PATH</b>	Name of the file; the file must exist. Can include a path name according to target system
<b>: STRING</b>	conventions.
<b>ID : DINT</b>	ID of the open filer NULL if the file can't be read

**4.11.3.13 F\_SEEK** PLCopen ✔

Set the current position in an open file

```
OK := F_SEEK ( ID, POS, ORG ) ;
```

<b>ID</b>	DINT	ID of an open file
<b>POS</b>	DINT	Number of bytes to offset from ORG. If ORG = SEEK_SET, then POS should be ≥ 0 If ORG = SEEK_END, then POS should be ≤ 0 If ORG = SEEK_CUR, then POS can be positive or negative
<b>ORG</b>	DINT	Origin of the move: SEEK_SET = 0 - beginning of the file SEEK_CUR = 1 - current position SEEK_END = 2 - end of the file
<b>OK</b>	BOOL	return check: TRUE if successful

**4.11.3.14 F\_WOPEN** PLCopen ✔

Open a file for writing

```
ID := F_WOPEN ( PATH ) ;
```

<b>PATH : STRING</b>	Name of the file. Can include a path name according to target system conventions.
<b>ID : DINT</b>	ID of the open file or NULL if the file can't be read

If the file does not exist, it is created. If the file already exists, its contents are cleared.

#### 4.11.3.15 FA\_READ

Read a DINT value from a file

```
Q := FA_READ (ID);
```

<b>ID: DINT</b>	ID of a file open for reading
<b>Q : DINT</b>	read value or 0 in case of error

This function reads an integer values from a file. Integer values read by `FA_READ` must have been written by the `FA_WRITE` function. Integers are stored in binary format in the file, using memory conventions of the target system. After the file is opened, the first call to `FA_READ` will read the first integer data, the second call will read the next, etc. To re-start reading at the beginning of the file, close the file and re-open it.

#### 4.11.3.16 FA\_WRITE

Write a DINT value to a file

```
OK := FA_WRITE (ID, IN);
```

<b>ID : DINT</b>	ID of a file open for writing
<b>IN : DINT</b>	integer value to be written
<b>OK : BOOL</b>	return check; TRUE if successful

Integers are stored in binary format in the file, using memory conventions of the target system.

#### 4.11.3.17 FB\_READ

Read binary data from a file

```
OK := FB_READ (ID, V);
```

<b>ID : DINT</b>	ID of a file open for writing
<b>V : ANY</b>	variable to be read; cannot be a string.
<b>OK : BOOL</b>	return check; TRUE if successful

This function reads binary data from a file. Variables are stored in binary format in the file, using memory conventions of the target system. After the file is opened, the first call to `FB_READ` will read the first binary data, the second call will read the next, etc. To re-start reading at the beginning of the file, close the file and re-open it.

#### 4.11.3.18 FB\_WRITE

Write binary data to a file

```
OK := FB_WRITE (ID, V);
```

<b>ID : DINT</b>	ID of a file open for writing
<b>V : ANY</b>	variable to be written; cannot be a string.
<b>OK : BOOL</b>	return check; TRUE if successful

Variables are stored in binary format in the file, using memory conventions of the target system.

**4.11.3.19 FM\_READ** PLCopen ✓

Read a string value from a file

```
Q := FM_READ (ID);
```

<b>ID : DINT</b>	ID of a file open for reading
<b>Q : STRING</b>	read value or empty string in case of error

This function reads a string from a file. After the file is opened, the first call to FM\_READ will read the first line of text, the second call will read the next line of text, etc. The maximum string length is defined by either the maximum length for the return variable or when the EOL (end of line) character is encountered, whichever occurs first. Close the file and reopen it to restart reading it at the beginning.

**4.11.3.20 FM\_WRITE** PLCopen ✓

Write a string value to a file

```
OK := FM_WRITE (ID, IN);
```

<b>ID : DINT</b>	ID of a file open for writing
<b>IN : STRING</b>	string value to be written
<b>OK : BOOL</b>	return check; TRUE if successful

This function writes a text line in the file. End of line character is systematically written after the input string.

**4.11.3.21.1 String Escape Sequences**

For greater formatting control over your STRING output, you may escape the STRING by prepending a \$ and use a pre-defined sequence. This is called a string escape sequence.

Escape Sequence	Result
\$\$	\$
\$'	'
\$L	linefeed
\$N	newline
\$P	page (form feed)
\$R	return
\$T	tab
\$xx	hex value

The following is an example of how STRING escape sequences can be used.

```
ID:=F_WOPEN('c:\ myfile.txt');
WOK:=FM_WRITE(ID,'123456$N');
//WOK:=FM_WRITE(ID,'$N');
WOK:=FM_WRITE(ID,'abcd$N');
WOK:=FM_WRITE(ID,'the end');
WOK:=F_Close(ID);
```

The example outputs a file which reads:

```
123456
abcd
the end
```

**4.11.3.22 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.

**4.11.3.23 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.

**4.11.3.24 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)



#### 4.11.4 File Management Function Examples

Following are several examples of how File Management functions may be used. The functions used include "F\_AOPEN" (p. 251), "F\_CLOSE" (p. 251), "F\_WOPEN" (p. 253), and "FM\_WRITE" (p. 255).

```
// Determine if this is a UNIX-based or Windows operating system
and set the directory.
ID:= F_AOPEN('C:\Program Files\Kollmorgen\Kollmorgen Automation
Suite\Sinope Runtime\Resources\http.conf');
IF ID > 0 THEN
    OutputFile := '\' + FileName;
    F_CLOSE(ID);
ELSE
    OutputFile := FileName;
END_IF;

IF (AddFileExt = true) THEN
    OutputFile := OutputFile + '.csv';
END_IF;
```

```
// Create a file for writing
FileID := F_WOPEN (OutputFile);
IF (FileID = 0) THEN
    RETURN;
END_IF;
```

```
// Write header information to a file
HeaderStr := 'Time[ms], ' + Header1 + '$R';
bStatus := FM_WRITE (FileID, HeaderStr);
IF (bStatus = false) THEN
    F_CLOSE (FileID);
    FileID := 0;
    RETURN;
END_IF;
```

## 4.12 GETSYSINFO

*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
_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

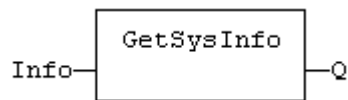
In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung (ENO) keeps the same value as the input rung.

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

### 4.12.4 ST Language

Q := GETSYSINFO (INFO);

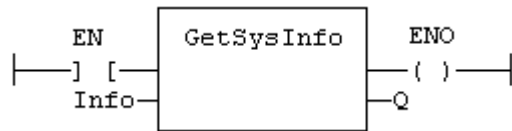
### 4.12.5 FBD Language



#### 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 Hysteresis

#### 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.

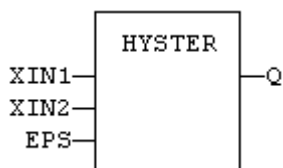
#### 4.13.4 ST Language

(\* MyHyst is a declared instance of HYSTER function block \*)

MyHyst (XIN1, XIN2, EPS);

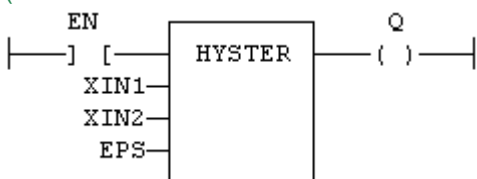
Q := MyHyst.Q;

#### 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\\_ALARM](#) [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

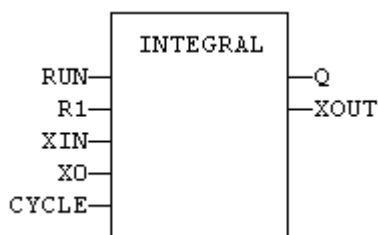
### 4.14.3 Remarks

In FFLD language, the input rung is the RUN command. The output rung is the Q report status.

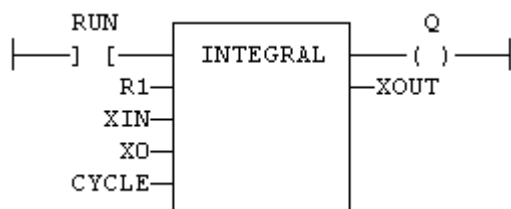
### 4.14.4 ST Language

```
(* 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\\_ALARM](#) [HYSTER](#) [STACKINT](#)

## 4.15 LIFO PLCopen

*Function block* - Manages a "last in / first out" stack

### 4.15.1 Inputs

<b>PUSH</b>	<b>BOOL</b>	Push a new value (on rising edge)
<b>POP</b>	<b>BOOL</b>	Pop a new value (on rising edge)
<b>RST</b>	<b>BOOL</b>	Reset the list
<b>NEXTIN</b>	<b>ANY</b>	Value to be pushed
<b>NEXTOUT</b>	<b>ANY</b>	Value at the top of the stack - <i>updated after call!</i>
<b>BUFFER</b>	<b>ANY</b>	Array for storing values

### 4.15.2 Outputs

<b>EMPTY</b>	<b>BOOL</b>	TRUE if the stack is empty
<b>OFLO</b>	<b>BOOL</b>	TRUE if overflow on a PUSH command
<b>COUNT</b>	<b>DINT</b>	Number of values in the stack
<b>PREAD</b>	<b>DINT</b>	Index in the buffer of the top of the stack
<b>PWRITE</b>	<b>DINT</b>	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.

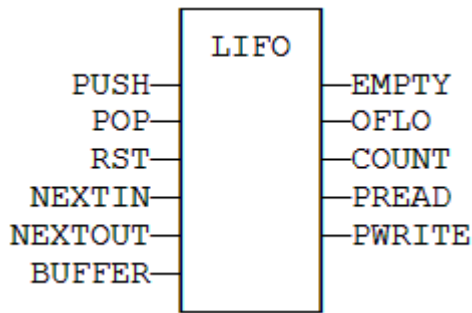
The first time the block is called, it remembers on which array it should work. If you call later the same instance with another BUFFER input, the call is considered as invalid and makes nothing. Outputs reports an empty stack in this case.

In FFLD language, input rung is the PUSH input. The output rung is the EMPTY output.

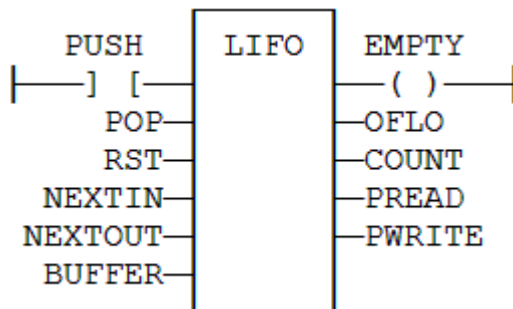
### 4.15.4 ST Language

```
(* 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\_ALARM 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

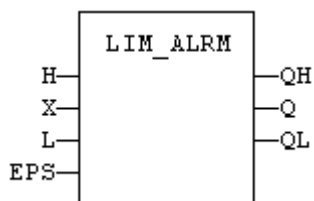
### 4.16.3 Remarks

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

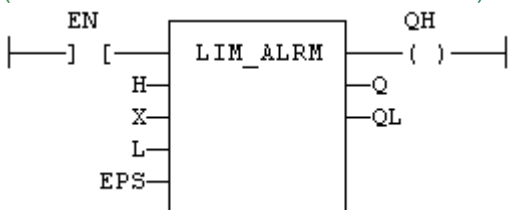
```
(* MyAlarm is a declared instance of LIM_ALARM function block *)
MyAlarm (H, X, L, EPS);
QH := MyAlarm.QH;
Q := MyAlarm.Q;
QL := MyAlarm.QL;
```

### 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_ALARM 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

*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 <a href="#">VLID</a> function)
PATH : STRING	Path name of the CSV file

### 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.

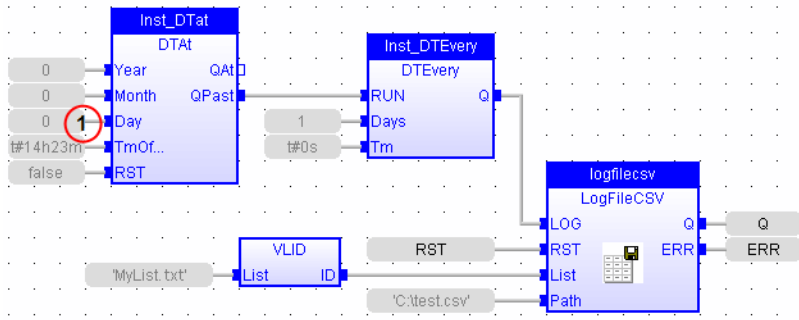
When a LOG or RST command is requested, the Q output is set to TRUE if successful.

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

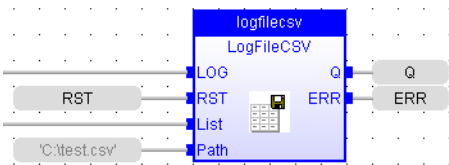
(2:23 pm) (see call out )



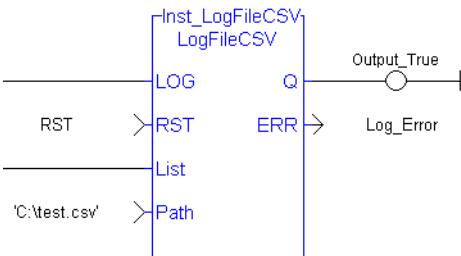
#### 4.17.4 ST Language

```
(* 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 PLCopen

Function Block - PID loop

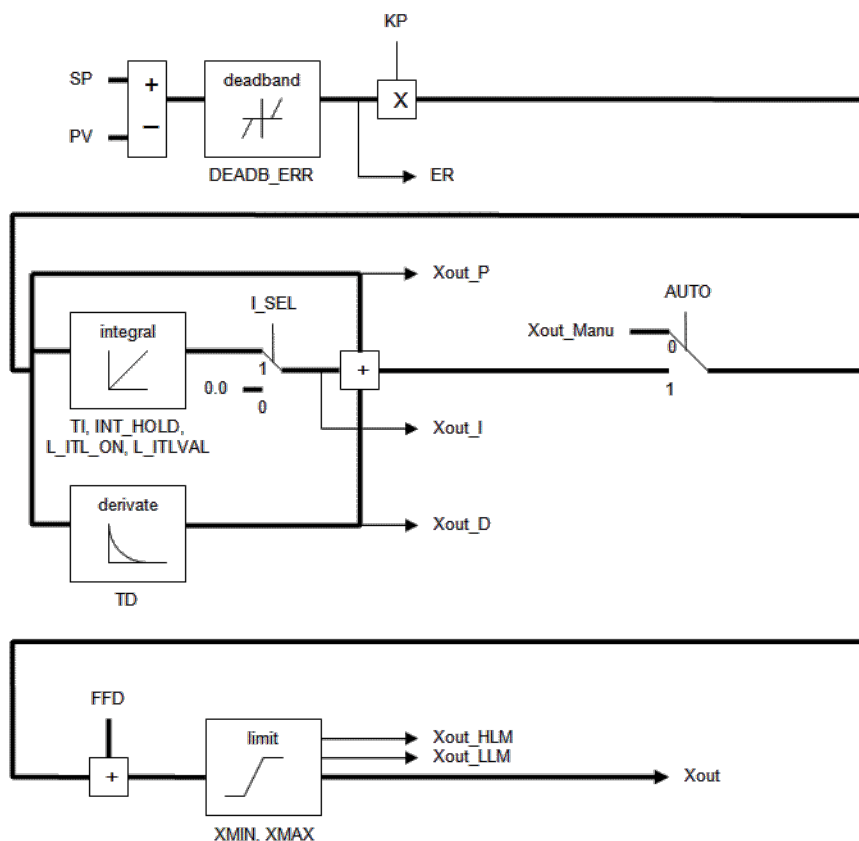
### 4.18.1 Inputs

Input	Type	Description
<b>AUTO</b>	BOOL	TRUE = normal mode , FALSE = manual mode.
<b>PV</b>	REAL	Process value.
<b>SP</b>	REAL	Set point.
<b>Xout_Manu</b>	REAL	Output value in manual mode.
<b>KP</b>	REAL	Gain.
<b>TI</b>	REAL	Integration time factor. A value of zero will stop the integrator and freeze Xout I at the previous sample's calculated value.
<b>TD</b>	REAL	Derivation factor.
<b>TS</b>	TIME	Sampling period.
<b>XMIN</b>	REAL	Minimum allowed output value.
<b>XMAX</b>	REAL	Maximum output value.
<b>I_SEL</b>	BOOL	If FALSE, the integrated value is ignored.
<b>INT_HOLD</b>	BOOL	If TRUE, the integrated value is frozen.
<b>I_ITL_ON</b>	BOOL	If TRUE, the integrated value is reset to I_ITLVAL.
<b>I_ITLVAL</b>	REAL	Reset value for integration when I_ITL_ON is TRUE.
<b>DEADB_ERR</b>	REAL	Hysteresis on PV. PV will be considered as unchanged if greater than (PVprev - DEADBAND_W) and less that (PRprev + DEADBAND_W).
<b>FFD</b>	REAL	Disturbance value on output.

### 4.18.2 Outputs

Output	Type	Description
<b>Xout</b>	REAL	Output command value.
<b>ER</b>	REAL	Last calculated error.
<b>Xout_P</b>	REAL	Last calculated proportional value.
<b>Xout_I</b>	REAL	Last calculated integrated value.
<b>Xout_D</b>	REAL	Last calculated derivated value.
<b>Xout_HLM</b>	BOOL	TRUE if the output value is saturated to XMIN.
<b>Xout_LLM</b>	BOOL	TRUE if the output value is saturated to XMAX.

### 4.18.3 Diagram



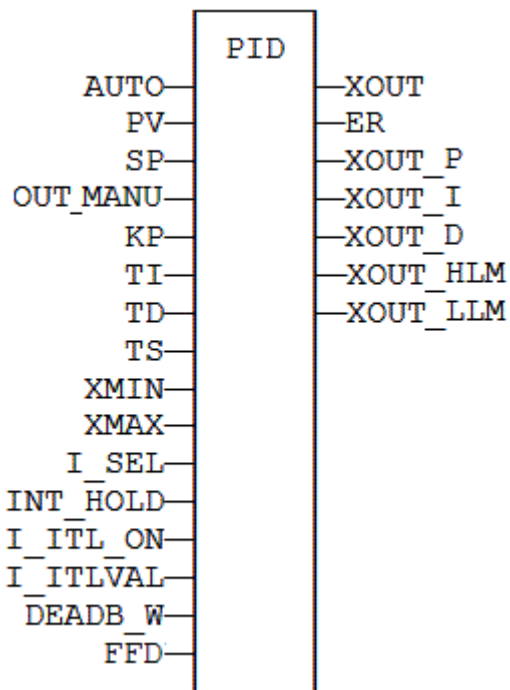
### 4.18.4 Remarks

- 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

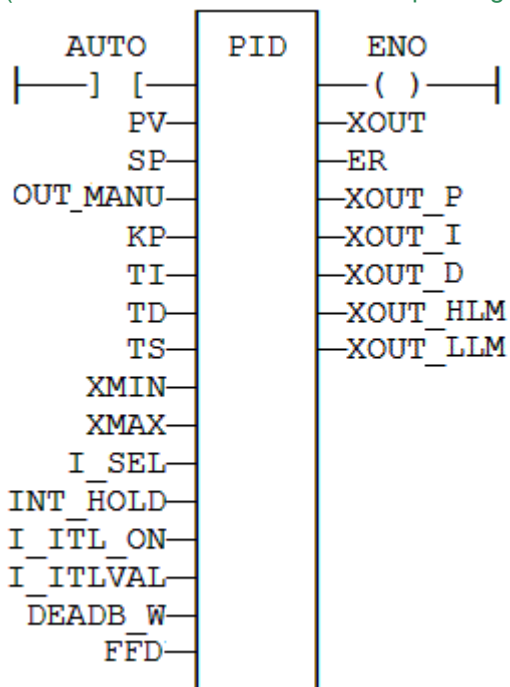
```
(* MyPID is a declared instance of PID function block *)
MyPID (AUTO, PV, SP, XOUT_MANU, KP, TI, TD, TS, XMIN, XMAX, I_
SEL, I_ITL_ON, I_ITLVAL, DEADB_ERR, FFD);
XOUT := MyPID.XOUT;
ER := MyPID.ER;
XOUT_P := MyPID.XOUT_P;
XOUT_I := MyPID.XOUT_I;
XOUT_D := MyPID.XOUT_D;
XOUT_HLM := MyPID.XOUT_HLM;
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

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

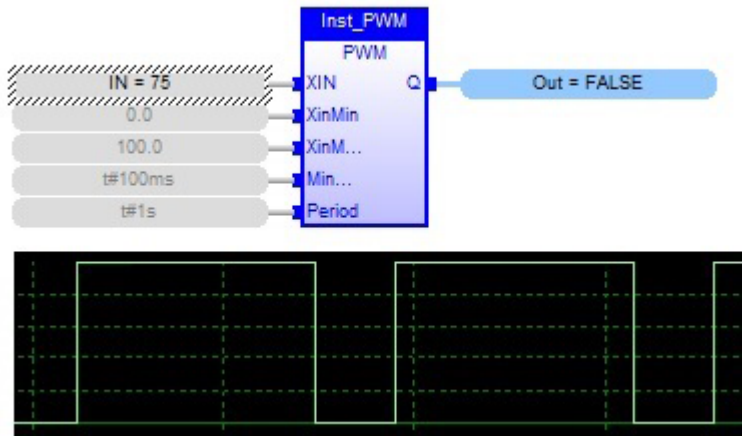
PWM1 is a declared instance of PWM function block.

```

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

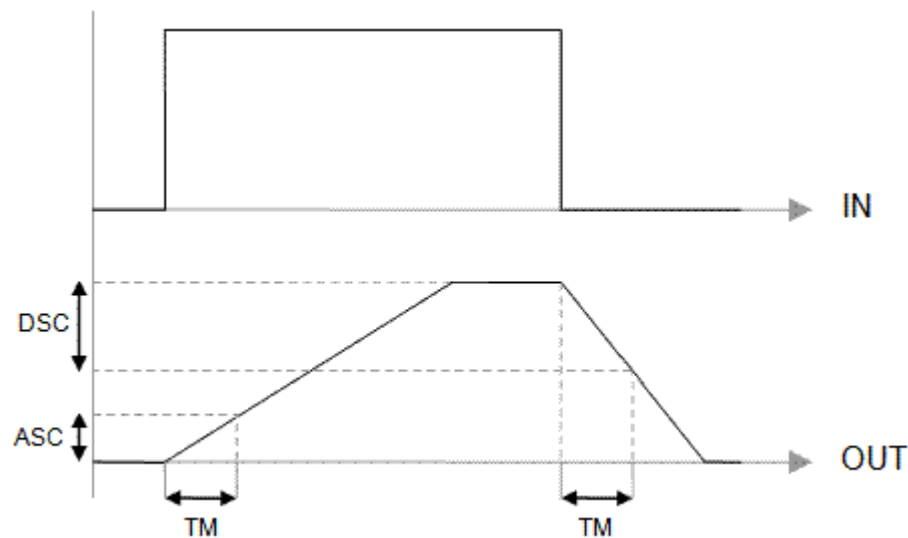
### 4.20.1 Inputs

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.

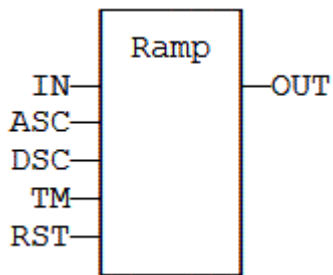
### 4.20.5 ST Language

(\* MyRamp is a declared instance of RAMP function block \*)

```
MyRamp (IN, ASC, DSC, TM, RST);
```

```
OUT := MyBlinker.OUT;
```

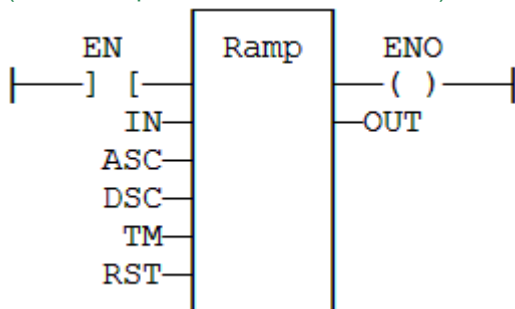
### 4.20.6 FBD Language



#### 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:

<b>DTCurDate</b>	Get current date stamp
<b>DTCurTime</b>	Get current time stamp
<b>DTDay</b>	Get day from date stamp
<b>DTMonth</b>	Get month from date stamp
<b>DTYear</b>	Get year from date stamp
<b>DTSec</b>	Get seconds from time stamp
<b>DTMin</b>	Get minutes from time stamp
<b>DTHour</b>	Get hours from time stamp
<b>DTMs</b>	Get milliseconds from time stamp

The following functions format the current date/time to a string:

<a href="#">DAY_TIME</a>	With predefined format
<a href="#">DTFORMAT</a>	With custom format

The following functions are used for triggering operations:

<a href="#">DTAt</a>	Pulse signal at the given date/time
<a href="#">DTEvery</a>	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.

#### 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**

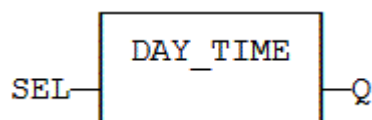
Possible values of the SEL input are:

- 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.

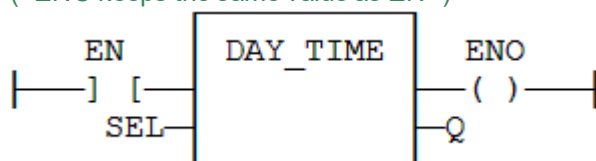
**4.21.1.4 ST Language**

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)

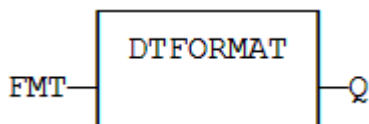
#### Example

```
(* 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

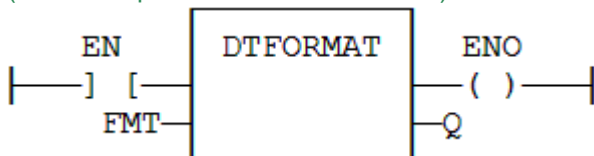
```
Q := DTFORMAT (FMT);
```

#### 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, tmoftday=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

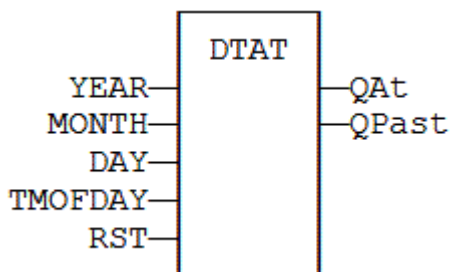
(\* MyDTAT is a declared instance of DTAT function block \*)

```
MyDTAT (YEAR, MONTH, DAY, TMOFDAY, RST);
```

```
QAT := MyDTAT.QAT;
```

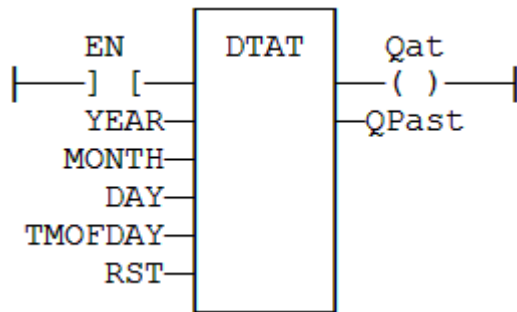
```
QPAST := MyDTAT.QPAST;
```

#### 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 MyDTAT.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:  
 $\text{DAYS} * 24\text{h} + \text{TM}$

For instance, specifying DAYS=1 and TM=6h means a period of 30 hours.

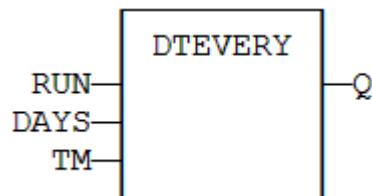
##### 4.21.4.4 ST Language

(\* MyDTEVERY is a declared instance of DTEVERY function block \*)

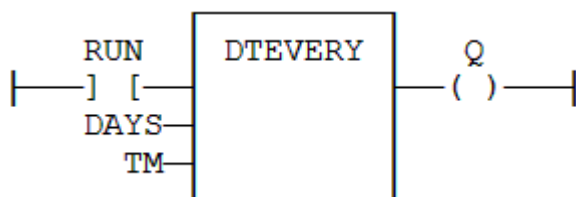
MyDTEVERY (RUN DAYS, TM);

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 Serializeln 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:

<b>1 byte</b>	BOOL, SINT, USINT and BYTE variables
<b>2 bytes</b>	INT, UINT and WORD variables
<b>4 bytes</b>	DINT, UDINT, DWORD and REAL variables
<b>8 bytes</b>	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

<b>En</b>	<b>Description</b>	Execute the function
	<b>Data type</b>	BOOL
	<b>Range</b>	[0,1]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>Frame[]</b>	<b>Description</b>	Source buffer - must be an array.
	<b>Data type</b>	USINT
	<b>Range</b>	[0,+65535]
	<b>Unit</b>	n/a
	<b>Default</b>	n/a
<b>Data</b>	<b>Description</b>	Destination variable to be copied

	<b>Data type</b>	any except STRING
	<b>Range</b>	—
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>Pos</b>	<b>Description</b>	Position in the source buffer
	<b>Data type</b>	DINT
	<b>Range</b>	[0,+65535]
	<b>Unit</b>	n/a
	<b>Default</b>	n/a
<b>BigEndian</b>	<b>Description</b>	TRUE if the frame is encoded with Big Endian format.
	<b>Data type</b>	BOOL
	<b>Range</b>	?
	<b>Unit</b>	?
	<b>Default</b>	?

#### 4.22.2.2 Output

<b>OK</b>	<b>Description</b>	Returns true when the function successfully executes. See <a href="#">Function - General rules</a> .
	<b>Data type</b>	BOOL
	<b>Unit</b>	n/a
<b>NextPos</b>	<b>Description</b>	Position in the source buffer after the extracted data. 0 in case of error (invalid position / buffer size).
	<b>Data type</b>	DINT
	<b>Unit</b>	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:

<b>1 byte</b>	BOOL, SINT, USINT and BYTE variables
<b>2 bytes</b>	INT, UINT and WORD variables
<b>4 bytes</b>	DINT, UDINT, DWORD and REAL variables
<b>8 bytes</b>	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

<b>En</b>	<b>Description</b>	Execute the function
	<b>Data type</b>	BOOL
	<b>Range</b>	[0,1]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>Frame[]</b>	<b>Description</b>	Destination buffer - must be an array.
	<b>Data type</b>	USINT
	<b>Range</b>	[0,+65535]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>Data</b>	<b>Description</b>	Source variable to be copied

	<b>Data type</b>	any except STRING
	<b>Range</b>	—
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>Pos</b>	<b>Description</b>	Position in the destination buffer
	<b>Data type</b>	DINT
	<b>Range</b>	[0,+65535]
	<b>Unit</b>	n/a
	<b>Default</b>	—
<b>BigEndian</b>	<b>Description</b>	TRUE if the frame is encoded with Big Endian format.
	<b>Data type</b>	BOOL
	<b>Range</b>	[0,1]
	<b>Unit</b>	n/a
	<b>Default</b>	—

#### 4.23.2.2 Output

<b>OK</b>	<b>Description</b>	Returns true when the function successfully executes. See <a href="#">Function - General rules</a> .
	<b>Data type</b>	BOOL
	<b>Unit</b>	n/a
<b>NextPos</b>	<b>Description</b>	Position in the destination buffer after the copied data. 0 in case of error (invalid position / buffer size).
	<b>Data type</b>	DINT
	<b>Unit</b>	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

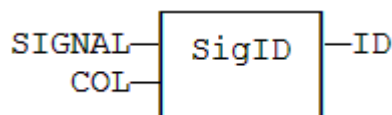
### 4.24.3 Remarks

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.

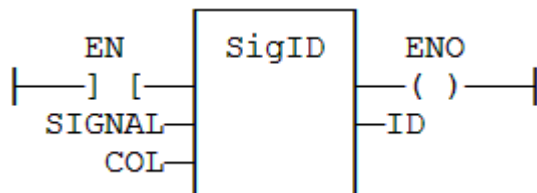
### 4.24.4 ST Language

```
ID := SigID ('MySignal', 'FirstColumn');
```

### 4.24.5 FBD Language



### 4.24.6 FLD 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 <a href="#">SigID</a> 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.

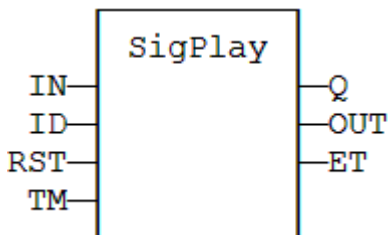
The "TM" input specifies the minimum amount of time in between two changes of the output signal. This parameter is ignored if less than the cycle scan time.

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.

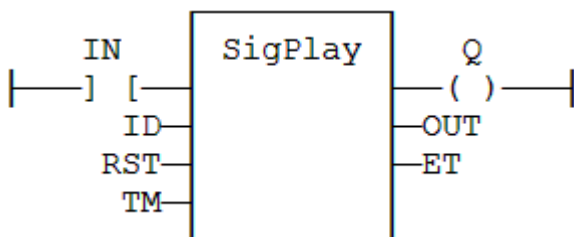
### 4.25.4 ST Language

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

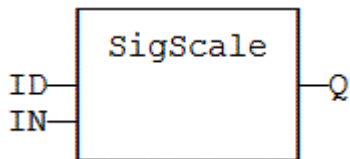
The "ID" argument is the identifier of the "signal" resource. Use the [SigID](#) function to get this value.

This function converts a time value to an 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.

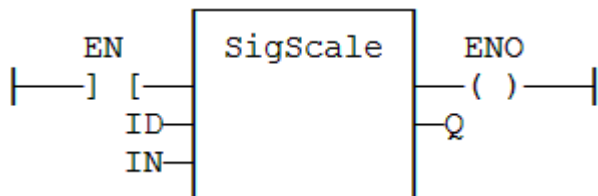
### 4.26.4 ST Language

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.

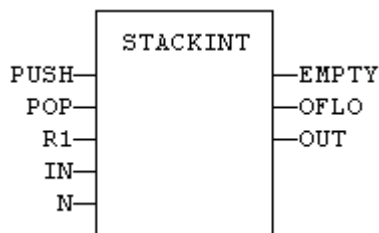
The specified size (N) is taken into account only when the R1 (reset) input is TRUE.

### 4.27.4 ST Language

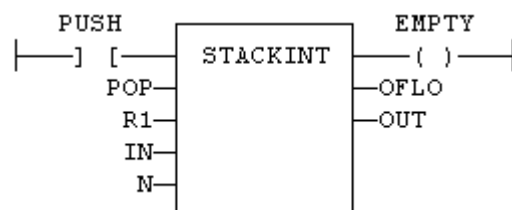
(\* MyStack is a declared instance of STACKINT function block \*)

```
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\\_ALARM](#) [HYSTER](#)

## 4.28 SurfLin

*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* - Get the identifier 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

Some blocks have arguments that refer to a list of variables. For all these blocks, the "list" argument is materialized by a numerical identifier. This function enables you to get the identifier of a list of variables.

Embedded lists of variables are simple ".TXT" text files with one variable name per line (note that you can only declare global variable).

Lists must contain single variables only. Items of arrays and structures must be specified one by one. The length of the list is not limited by the system.

#### ! 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

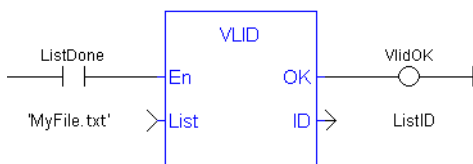
```
ID := VLID ('MyFile.txt');
```

### 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 ID
```



## 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](http://www.kollmorgen.com)  
**Mail:** [support@kollmorgen.com](mailto:support@kollmorgen.com)  
**Tel.:** +1 - 540 - 633 - 3545  
**Fax:** +1 - 540 - 639 - 4162

### Europa KOLLMORGEN Europe GmbH

Pempelfurtstr. 1  
40880 Ratingen, Germany

**Web:** [www.kollmorgen.com](http://www.kollmorgen.com)  
**Mail:** [technik@kollmorgen.com](mailto: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](http://www.kollmorgen.com)  
**Mail:** [contato@kollmorgen.com](mailto:contato@kollmorgen.com)  
**Tel.:** +55 11 4615-6300

### China and SEA KOLLMORGEN

Floor 4, Building 9, No. 518,  
North Fuquan Road, Changning District,  
Shanghai 200335, China

**Web:** [www.kollmorgen.cn](http://www.kollmorgen.cn)  
**Mail:** [sales.china@kollmorgen.com](mailto:sales.china@kollmorgen.com)  
**Tel.:** +86 - 400 661 2802

**KOLLMORGEN**<sup>®</sup>

*Because Motion Matters™*