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

Copyrights

Copyright © 2009-2022 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:

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

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

Disclaimer

The information in this document (Version V published on 12/7/2022) 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.
# Table of Contents

1 Trademarks and Copyrights .............................................................................. 2
2 Table of Contents ......................................................................................... 3
3 Programming Languages .............................................................................. 35
   3.1 Sequential Function Chart (SFC) .............................................................. 35
      3.1.1 SFC Execution at Runtime .............................................................. 35
          3.1.1.1 Divergence ........................................................................... 38
      3.1.2 Hierarchy of SFC Programs ............................................................. 40
      3.1.3 Control an SFC Child Program ....................................................... 40
   3.2 Function Block Diagram (FBD) ............................................................... 41
      3.2.1 Data Flow .................................................................................... 41
      3.2.2 FFLD Symbols ............................................................................ 41
   3.3 Instruction List (IL) ............................................................................. 42
      3.3.1 Comments ................................................................................... 42
      3.3.2 Data Flow ................................................................................... 42
      3.3.3 Evaluation of Expressions .............................................................. 43
      3.3.4 Actions ....................................................................................... 44
   3.4 Structured Text (ST) ............................................................................. 44
      3.4.1 Comments ................................................................................... 44
      3.4.2 Expressions ................................................................................ 45
      3.4.3 Statements ................................................................................ 45
         3.4.3.1 Basic Statements .................................................................. 45
         3.4.3.2 Conditional Statements ......................................................... 45
         3.4.3.3 Loop Statements .................................................................. 46
         3.4.3.4 Other Statements ................................................................ 46
         3.4.3.5 Helpful Features ................................................................ 47
   3.5 Free Form Ladder Diagram (FFLD) ......................................................... 47
      3.5.1 Use of EN Input and ENO Output for Blocks ................................. 47
         3.5.1.1 Examples .......................................................................... 47
      3.5.2 Contacts and Coils ....................................................................... 48
         3.5.2.1 FFLD Contacts .................................................................... 49
         3.5.2.2 FFLD Coils .......................................................................... 50
   4 PLC Standard Libraries ............................................................................. 53
   4.1 Basic Operations .................................................................................... 54
      4.1.1 Data Manipulation ....................................................................... 54
      4.1.2 Control Program Execution ............................................................ 54
         4.1.2.1 Language Features ................................................................ 54
         4.1.2.2 Structured Statements ............................................................ 54
      4.1.3 Assignment := ............................................................................ 54
         4.1.3.1 Inputs .................................................................................. 55
         4.1.3.2 Outputs .............................................................................. 55
         4.1.3.3 Remarks ............................................................................. 55
         4.1.3.4 FBD Language ..................................................................... 55
         4.1.3.5 FFLD Language .................................................................. 55
         4.1.3.6 IL Language ....................................................................... 55
4.1.3.7 ST Language ................................................................. 56
4.1.4 Bit Access ................................................................. 56
4.1.5 Differences Between Functions and Function Blocks ................. 56
4.1.6 Call a Sub-Program ..................................................... 56
  4.1.6.1 FBD and FFLD Languages ......................................... 57
  4.1.6.2 IL Language .......................................................... 57
  4.1.6.3 ST Language ......................................................... 57
4.1.7 CASE OF ELSE END_CASE ........................................... 57
  4.1.7.1 Syntax ................................................................. 57
  4.1.7.2 Remarks ............................................................... 58
  4.1.7.3 FBD Language ........................................................ 58
  4.1.7.4 FFLD Language ........................................................ 58
  4.1.7.5 IL Language .......................................................... 58
  4.1.7.6 ST Language .......................................................... 58
4.1.8 CountOf ...................................................................... 59
  4.1.8.1 Inputs ................................................................. 59
  4.1.8.2 Outputs ............................................................... 59
  4.1.8.3 Remarks ............................................................... 59
  4.1.8.4 FBD Language ........................................................ 59
  4.1.8.5 FFLD Language ........................................................ 59
  4.1.8.6 IL Language .......................................................... 60
  4.1.8.7 ST Language .......................................................... 60
4.1.9 DEC .......................................................................... 60
  4.1.9.1 Inputs ................................................................. 60
  4.1.9.2 Outputs ............................................................... 60
  4.1.9.3 Remarks ............................................................... 60
  4.1.9.4 FBD Language ........................................................ 60
  4.1.9.5 FFLD Language ........................................................ 60
  4.1.9.6 IL Language .......................................................... 61
  4.1.9.7 ST Language .......................................................... 61
4.1.10 EXIT ........................................................................ 61
  4.1.10.1 Remarks ............................................................. 61
  4.1.10.2 FBD Language ...................................................... 61
  4.1.10.3 FFLD Language ...................................................... 61
  4.1.10.4 IL Language .......................................................... 61
  4.1.10.5 ST Language .......................................................... 61
4.1.11 FOR TO BY END_FOR .............................................. 62
  4.1.11.1 Syntax ................................................................. 62
  4.1.11.2 Remarks ............................................................... 62
  4.1.11.3 FBD Language ........................................................ 62
  4.1.11.4 FFLD Language ........................................................ 62
  4.1.11.5 IL Language .......................................................... 62
  4.1.11.6 ST Language .......................................................... 62
4.1.12 IF THEN ELSE ELSIF END_IF ..................................... 63
  4.1.12.1 Syntax ................................................................. 63
  4.1.12.2 Remarks ............................................................... 63
  4.1.12.3 FBD Language ........................................................ 63
4.1.18.6 ST Language ................................................................. 70
4.1.19 RETURN RET RETC RETNC RETCN ................................ 70
  4.1.19.1 Remarks ................................................................. 70
  4.1.19.2 FBD Language .......................................................... 70
  4.1.19.3 FFLD Language ......................................................... 71
  4.1.19.4 IL Language ............................................................. 71
  4.1.19.5 ST Language ............................................................. 71
4.1.20 WAIT / WAIT_TIME ......................................................... 71
  4.1.20.1 Syntax ................................................................. 71
  4.1.20.2 Remarks ................................................................. 72
  4.1.20.3 FBD Language .......................................................... 72
  4.1.20.4 FFLD Language ......................................................... 72
  4.1.20.5 IL Language ............................................................. 73
  4.1.20.6 ST Language ............................................................. 73
4.1.21 WHILE DO END_WHILE .................................................... 73
  4.1.21.1 Syntax ................................................................. 73
  4.1.21.2 Remarks ................................................................. 73
  4.1.21.3 FBD Language .......................................................... 73
  4.1.21.4 FFLD Language ......................................................... 73
  4.1.21.5 IL Language ............................................................. 73
  4.1.21.6 ST Language ............................................................. 73

4.2 Boolean Operations .......................................................... 74
  4.2.1 Standard Operators ....................................................... 74
  4.2.2 Available Blocks .......................................................... 74
  4.2.3 FlipFlop ................................................................. 74
    4.2.3.1 Inputs ................................................................. 75
    4.2.3.2 Outputs ............................................................... 75
    4.2.3.3 Remarks ............................................................... 75
    4.2.3.4 FBD Language ........................................................ 75
    4.2.3.5 FFLD Language ........................................................ 75
    4.2.3.6 IL Language ........................................................... 75
    4.2.3.7 ST Language ........................................................... 75
  4.2.4 f_trig ................................................................. 76
    4.2.4.1 Inputs ................................................................. 76
    4.2.4.2 Outputs ............................................................... 76
    4.2.4.3 Truth Table ........................................................... 76
    4.2.4.4 Remarks ............................................................... 76
    4.2.4.5 FBD Language ........................................................ 76
    4.2.4.6 FFLD Language ........................................................ 76
    4.2.4.7 IL Language ........................................................... 76
    4.2.4.8 ST Language ........................................................... 77
  4.2.5 NOT ................................................................. 77
    4.2.5.1 Inputs ................................................................. 77
    4.2.5.2 Outputs ............................................................... 77
    4.2.5.3 Truth Table ........................................................... 77
    4.2.5.4 Remarks ............................................................... 77
    4.2.5.5 FBD Language ........................................................ 77
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.5.6</td>
<td>FFLD Language</td>
</tr>
<tr>
<td>4.2.5.7</td>
<td>IL Language</td>
</tr>
<tr>
<td>4.2.5.8</td>
<td>ST Language</td>
</tr>
<tr>
<td>4.2.6</td>
<td>QOR</td>
</tr>
<tr>
<td>4.2.6.1</td>
<td>Inputs</td>
</tr>
<tr>
<td>4.2.6.2</td>
<td>Outputs</td>
</tr>
<tr>
<td>4.2.6.3</td>
<td>Remarks</td>
</tr>
<tr>
<td>4.2.6.4</td>
<td>FBD Language</td>
</tr>
<tr>
<td>4.2.6.5</td>
<td>FFLD Language</td>
</tr>
<tr>
<td>4.2.6.6</td>
<td>IL Language</td>
</tr>
<tr>
<td>4.2.6.7</td>
<td>ST Language</td>
</tr>
<tr>
<td>4.2.7</td>
<td>R</td>
</tr>
<tr>
<td>4.2.7.1</td>
<td>Inputs</td>
</tr>
<tr>
<td>4.2.7.2</td>
<td>Outputs</td>
</tr>
<tr>
<td>4.2.7.3</td>
<td>Truth Table</td>
</tr>
<tr>
<td>4.2.7.4</td>
<td>Remarks</td>
</tr>
<tr>
<td>4.2.7.5</td>
<td>FBD Language</td>
</tr>
<tr>
<td>4.2.7.6</td>
<td>FFLD Language</td>
</tr>
<tr>
<td>4.2.7.7</td>
<td>IL Language</td>
</tr>
<tr>
<td>4.2.7.8</td>
<td>ST Language</td>
</tr>
<tr>
<td>4.2.8</td>
<td>RS</td>
</tr>
<tr>
<td>4.2.8.1</td>
<td>Inputs</td>
</tr>
<tr>
<td>4.2.8.2</td>
<td>Outputs</td>
</tr>
<tr>
<td>4.2.8.3</td>
<td>Truth Table</td>
</tr>
<tr>
<td>4.2.8.4</td>
<td>Remarks</td>
</tr>
<tr>
<td>4.2.8.5</td>
<td>FBD Language</td>
</tr>
<tr>
<td>4.2.8.6</td>
<td>FFLD Language</td>
</tr>
<tr>
<td>4.2.8.7</td>
<td>IL Language</td>
</tr>
<tr>
<td>4.2.8.8</td>
<td>ST Language</td>
</tr>
<tr>
<td>4.2.9</td>
<td>r_trig</td>
</tr>
<tr>
<td>4.2.9.1</td>
<td>Inputs</td>
</tr>
<tr>
<td>4.2.9.2</td>
<td>Outputs</td>
</tr>
<tr>
<td>4.2.9.3</td>
<td>Truth Table</td>
</tr>
<tr>
<td>4.2.9.4</td>
<td>Remarks</td>
</tr>
<tr>
<td>4.2.9.5</td>
<td>FBD Language</td>
</tr>
<tr>
<td>4.2.9.6</td>
<td>FFLD Language</td>
</tr>
<tr>
<td>4.2.9.7</td>
<td>IL Language</td>
</tr>
<tr>
<td>4.2.9.8</td>
<td>ST Language</td>
</tr>
<tr>
<td>4.2.10</td>
<td>S</td>
</tr>
<tr>
<td>4.2.10.1</td>
<td>Inputs</td>
</tr>
<tr>
<td>4.2.10.2</td>
<td>Outputs</td>
</tr>
<tr>
<td>4.2.10.3</td>
<td>Truth Table</td>
</tr>
<tr>
<td>4.2.10.4</td>
<td>Remarks</td>
</tr>
<tr>
<td>4.2.10.5</td>
<td>FBD Language</td>
</tr>
<tr>
<td>4.2.10.6</td>
<td>FFLD Language</td>
</tr>
<tr>
<td>4.2.10.7</td>
<td>IL Language</td>
</tr>
<tr>
<td>4.2.10.8</td>
<td>ST Language</td>
</tr>
</tbody>
</table>
## Table of Contents

4.2.11 sema .............................................................. 85
  4.2.11.1 Inputs ......................................................... 85
  4.2.11.2 Outputs ....................................................... 85
  4.2.11.3 Remarks ....................................................... 85
  4.2.11.4 FBD Language ........................................... 85
  4.2.11.5 FFLD Language ........................................... 85
  4.2.11.6 IL Language .............................................. 85
  4.2.11.7 ST Language .............................................. 85
4.2.12 SR ................................................................. 86
  4.2.12.1 Inputs ......................................................... 86
  4.2.12.2 Outputs ....................................................... 86
  4.2.12.3 Truth Table ................................................ 86
  4.2.12.4 Remarks ....................................................... 86
  4.2.12.5 FBD Language ........................................... 86
  4.2.12.6 FFLD Language ........................................... 87
  4.2.12.7 IL Language .............................................. 87
  4.2.12.8 ST Language .............................................. 87
4.2.13 XOR / XORN .................................................. 87
  4.2.13.1 Inputs ......................................................... 87
  4.2.13.2 Outputs ....................................................... 87
  4.2.13.3 Truth Table ................................................ 87
  4.2.13.4 Remarks ....................................................... 88
  4.2.13.5 FBD Language ........................................... 88
  4.2.13.6 FFLD Language ........................................... 88
  4.2.13.7 IL Language .............................................. 88
  4.2.13.8 ST Language .............................................. 88
4.3 Arithmetic Operations .......................................... 89
  4.3.1 Standard Operators ......................................... 89
  4.3.2 Standard Functions ......................................... 89
  4.3.3 Addition + ....................................................... 89
    4.3.3.1 Inputs ..................................................... 89
    4.3.3.2 Outputs .................................................... 89
    4.3.3.3 Remarks .................................................... 90
    4.3.3.4 FBD Language .......................................... 90
    4.3.3.5 FFLD Language .......................................... 90
    4.3.3.6 IL Language ............................................ 90
    4.3.3.7 ST Language ............................................ 90
  4.3.4 Divide / ......................................................... 90
    4.3.4.1 Inputs ..................................................... 91
    4.3.4.2 Outputs .................................................... 91
    4.3.4.3 Remarks .................................................... 91
    4.3.4.4 FBD Language .......................................... 91
    4.3.4.5 FFLD Language .......................................... 91
    4.3.4.6 IL Language ............................................ 91
    4.3.4.7 ST Language ............................................ 91
  4.3.5 NEG - ........................................................... 92
    4.3.5.1 Inputs ..................................................... 92
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3.5.2 Outputs</td>
<td>92</td>
</tr>
<tr>
<td>4.3.5.3 Truth Table</td>
<td>92</td>
</tr>
<tr>
<td>4.3.5.4 Remarks</td>
<td>92</td>
</tr>
<tr>
<td>4.3.5.5 FBD Language</td>
<td>92</td>
</tr>
<tr>
<td>4.3.5.6 FFLD Language</td>
<td>92</td>
</tr>
<tr>
<td>4.3.5.7 IL Language</td>
<td>92</td>
</tr>
<tr>
<td>4.3.5.8 ST Language</td>
<td>93</td>
</tr>
<tr>
<td>4.3.6 limit</td>
<td>93</td>
</tr>
<tr>
<td>4.3.6.1 Inputs</td>
<td>93</td>
</tr>
<tr>
<td>4.3.6.2 Outputs</td>
<td>93</td>
</tr>
<tr>
<td>4.3.6.3 Remarks</td>
<td>93</td>
</tr>
<tr>
<td>4.3.6.4 FBD Language</td>
<td>93</td>
</tr>
<tr>
<td>4.3.6.5 FFLD Language</td>
<td>94</td>
</tr>
<tr>
<td>4.3.6.6 IL Language</td>
<td>94</td>
</tr>
<tr>
<td>4.3.6.7 ST Language</td>
<td>94</td>
</tr>
<tr>
<td>4.3.7 max</td>
<td>94</td>
</tr>
<tr>
<td>4.3.7.1 Inputs</td>
<td>94</td>
</tr>
<tr>
<td>4.3.7.2 Outputs</td>
<td>94</td>
</tr>
<tr>
<td>4.3.7.3 Remarks</td>
<td>94</td>
</tr>
<tr>
<td>4.3.7.4 FBD Language</td>
<td>95</td>
</tr>
<tr>
<td>4.3.7.5 FFLD Language</td>
<td>95</td>
</tr>
<tr>
<td>4.3.7.6 IL Language</td>
<td>95</td>
</tr>
<tr>
<td>4.3.7.7 ST Language</td>
<td>95</td>
</tr>
<tr>
<td>4.3.8 min</td>
<td>95</td>
</tr>
<tr>
<td>4.3.8.1 Inputs</td>
<td>95</td>
</tr>
<tr>
<td>4.3.8.2 Outputs</td>
<td>96</td>
</tr>
<tr>
<td>4.3.8.3 Remarks</td>
<td>96</td>
</tr>
<tr>
<td>4.3.8.4 FBD Language</td>
<td>96</td>
</tr>
<tr>
<td>4.3.8.5 FFLD Language</td>
<td>96</td>
</tr>
<tr>
<td>4.3.8.6 IL Language</td>
<td>96</td>
</tr>
<tr>
<td>4.3.8.7 ST Language</td>
<td>96</td>
</tr>
<tr>
<td>4.3.9 mod / modLR / modR</td>
<td>96</td>
</tr>
<tr>
<td>4.3.9.1 Inputs</td>
<td>97</td>
</tr>
<tr>
<td>4.3.9.2 Outputs</td>
<td>97</td>
</tr>
<tr>
<td>4.3.9.3 Examples</td>
<td>97</td>
</tr>
<tr>
<td>4.3.9.4 Remarks</td>
<td>98</td>
</tr>
<tr>
<td>4.3.9.5 FBD Language</td>
<td>98</td>
</tr>
<tr>
<td>4.3.9.6 FFLD Language</td>
<td>98</td>
</tr>
<tr>
<td>4.3.9.7 IL Language</td>
<td>99</td>
</tr>
<tr>
<td>4.3.9.8 ST Language</td>
<td>99</td>
</tr>
<tr>
<td>4.3.10 Multiply *</td>
<td>99</td>
</tr>
<tr>
<td>4.3.10.1 Inputs</td>
<td>99</td>
</tr>
<tr>
<td>4.3.10.2 Outputs</td>
<td>99</td>
</tr>
<tr>
<td>4.3.10.3 Remarks</td>
<td>99</td>
</tr>
<tr>
<td>4.3.10.4 FBD Language</td>
<td>99</td>
</tr>
<tr>
<td>4.3.10.5 FFLD Language</td>
<td>100</td>
</tr>
<tr>
<td>4.3.10.6 IL Language</td>
<td>100</td>
</tr>
</tbody>
</table>
## Table of Contents

4.3.10.7 ST Language ................................................................. 100
4.3.11 odd ............................................................................. 100
  4.3.11.1 Inputs ..................................................................... 100
  4.3.11.2 Outputs ................................................................... 101
  4.3.11.3 Remarks ................................................................ 101
  4.3.11.4 FBD Language ....................................................... 101
  4.3.11.5 FFLD Language ...................................................... 101
  4.3.11.6 IL Language .......................................................... 101
  4.3.11.7 ST Language .......................................................... 101
4.3.12 SetWithin ..................................................................... 101
  4.3.12.1 Inputs ..................................................................... 102
  4.3.12.2 Outputs ................................................................... 102
  4.3.12.3 Truth Table ........................................................... 102
  4.3.12.4 Remarks ................................................................ 102
4.3.13 Subtraction - ................................................................. 102
  4.3.13.1 Inputs ..................................................................... 102
  4.3.13.2 Outputs ................................................................... 102
  4.3.13.3 Remarks ................................................................ 102
  4.3.13.4 FBD Language ....................................................... 102
  4.3.13.5 FFLD Language ...................................................... 103
  4.3.13.6 IL Language .......................................................... 103
  4.3.13.7 ST Language .......................................................... 103
4.4 Comparison Operations ......................................................... 103
  4.4.1 CMP ............................................................................ 104
    4.4.1.1 Inputs ..................................................................... 104
    4.4.1.2 Outputs ................................................................... 104
    4.4.1.3 Remarks ................................................................ 104
    4.4.1.4 FBD Language ....................................................... 104
    4.4.1.5 FFLD Language ...................................................... 104
    4.4.1.6 IL Language .......................................................... 104
    4.4.1.7 ST Language .......................................................... 105
  4.4.2 GE >= ........................................................................... 105
    4.4.2.1 Inputs ..................................................................... 105
    4.4.2.2 Outputs ................................................................... 105
    4.4.2.3 Remarks ................................................................ 105
    4.4.2.4 FBD Language ....................................................... 105
    4.4.2.5 FFLD Language ...................................................... 106
    4.4.2.6 IL Language .......................................................... 106
    4.4.2.7 ST Language .......................................................... 106
  4.4.3 GT > .............................................................................. 106
    4.4.3.1 Inputs ..................................................................... 106
    4.4.3.2 Outputs ................................................................... 106
    4.4.3.3 Remarks ................................................................ 107
    4.4.3.4 ST Language .......................................................... 107
    4.4.3.5 FBD Language ....................................................... 107
    4.4.3.6 FFLD Language ...................................................... 107
    4.4.3.7 IL Language .......................................................... 107
### 4.4.4 EQ =

<table>
<thead>
<tr>
<th>Section</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Remarks</th>
<th>FBD Language</th>
<th>FFLD Language</th>
<th>IL Language</th>
<th>ST Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.4.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.4.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.4.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.4.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.4.5 NE <>

<table>
<thead>
<tr>
<th>Section</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Remarks</th>
<th>FBD Language</th>
<th>FFLD Language</th>
<th>IL Language</th>
<th>ST Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.5.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.5.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.5.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.5.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.4.6 LE <=

<table>
<thead>
<tr>
<th>Section</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Remarks</th>
<th>FBD Language</th>
<th>FFLD Language</th>
<th>IL Language</th>
<th>ST Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.6.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.6.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.6.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.6.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.4.7 LT <

<table>
<thead>
<tr>
<th>Section</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Remarks</th>
<th>FBD Language</th>
<th>FFLD Language</th>
<th>IL Language</th>
<th>ST Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.7.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.7.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.7.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.7.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.5 Type Conversion Functions

#### 4.5.1 any_to_bool

<table>
<thead>
<tr>
<th>Section</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Remarks</th>
<th>FBD Language</th>
<th>FFLD Language</th>
<th>IL Language</th>
<th>ST Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5.1.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.1.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.1.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.1.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 4.5.2 any_to_dint / any_to_udint

<table>
<thead>
<tr>
<th>Section</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Remarks</th>
<th>FBD Language</th>
<th>FFLD Language</th>
<th>IL Language</th>
<th>ST Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5.2.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.2.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.2.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.2.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.2.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.2.6 IL Language</td>
<td>114</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.2.7 ST Language</td>
<td>115</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.3 any_to_int / any_to_uint</td>
<td>115</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.3.1 Inputs</td>
<td>115</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.3.2 Outputs</td>
<td>115</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.3.3 Remarks</td>
<td>115</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.3.4 FBD Language</td>
<td>115</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.3.5 FFLD Language</td>
<td>115</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.3.6 IL Language</td>
<td>116</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.3.7 ST Language</td>
<td>116</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.4 any_to_int / any_to_uint</td>
<td>116</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.4.1 Inputs</td>
<td>116</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.4.2 Outputs</td>
<td>116</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.4.3 Remarks</td>
<td>116</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.4.4 FBD Language</td>
<td>117</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.4.5 FFLD Language</td>
<td>117</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.4.6 IL Language</td>
<td>117</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.4.7 ST Language</td>
<td>117</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.5 any_to_int / any_to_uint</td>
<td>117</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.5.1 Inputs</td>
<td>117</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.5.2 Outputs</td>
<td>117</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.5.3 Remarks</td>
<td>117</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.5.4 FBD Language</td>
<td>118</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.5.5 FFLD Language</td>
<td>118</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.5.6 IL Language</td>
<td>118</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.5.7 ST Language</td>
<td>118</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.6 any_to_int / any_to_uint</td>
<td>118</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.6.1 Inputs</td>
<td>119</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.6.2 Outputs</td>
<td>119</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.6.3 Remarks</td>
<td>119</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.6.4 FBD Language</td>
<td>119</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.6.5 FFLD Language</td>
<td>119</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.6.6 IL Language</td>
<td>119</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.6.7 ST Language</td>
<td>119</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.7 any_to_time</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.7.1 Inputs</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.7.2 Outputs</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.7.3 Remarks</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.7.4 FBD Language</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.7.5 FFLD Language</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.7.6 IL Language</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.7.7 ST Language</td>
<td>121</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.8 any_to_int / any_to_uint</td>
<td>121</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.8.1 Inputs</td>
<td>121</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.8.2 Outputs</td>
<td>121</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.8.3 Remarks</td>
<td>121</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.8.4 FBD Language</td>
<td>121</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.8.5</td>
<td>FFLD Language</td>
<td>121</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.8.6</td>
<td>IL Language</td>
<td>122</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.8.7</td>
<td>ST Language</td>
<td>122</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.9</td>
<td>any_to_string</td>
<td>122</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.9.1</td>
<td>Inputs</td>
<td>122</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.9.2</td>
<td>Outputs</td>
<td>122</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.9.3</td>
<td>Remarks</td>
<td>122</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.9.4</td>
<td>FBD Language</td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.9.5</td>
<td>FFLD Language</td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.9.6</td>
<td>IL Language</td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.9.7</td>
<td>ST Language</td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.10</td>
<td>NUM_TO_STRING</td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.10.1</td>
<td>Inputs</td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.10.2</td>
<td>Outputs</td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.10.3</td>
<td>Remarks</td>
<td>124</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.11</td>
<td>bcd_to_bin</td>
<td>124</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.11.1</td>
<td>Inputs</td>
<td>124</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.11.2</td>
<td>Outputs</td>
<td>124</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.11.3</td>
<td>Remarks</td>
<td>125</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.11.4</td>
<td>FBD Language</td>
<td>125</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.11.5</td>
<td>FFLD Language</td>
<td>125</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.11.6</td>
<td>IL Language</td>
<td>125</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.11.7</td>
<td>ST Language</td>
<td>125</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.12</td>
<td>bin_to_bcd</td>
<td>125</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.12.1</td>
<td>Inputs</td>
<td>125</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.12.2</td>
<td>Outputs</td>
<td>126</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.12.3</td>
<td>Truth Table</td>
<td>126</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.12.4</td>
<td>Remarks</td>
<td>126</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.12.5</td>
<td>FBD Language</td>
<td>126</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.12.6</td>
<td>FFLD Language</td>
<td>126</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.12.7</td>
<td>IL Language</td>
<td>126</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5.12.8</td>
<td>ST Language</td>
<td>126</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## 4.6 Selectors

### 4.6.1 MUX4

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6.1.1</td>
<td>Inputs</td>
<td>127</td>
</tr>
<tr>
<td>4.6.1.2</td>
<td>Outputs</td>
<td>127</td>
</tr>
<tr>
<td>4.6.1.3</td>
<td>Truth Table</td>
<td>127</td>
</tr>
<tr>
<td>4.6.1.4</td>
<td>Remarks</td>
<td>127</td>
</tr>
<tr>
<td>4.6.1.5</td>
<td>FBD Language</td>
<td>128</td>
</tr>
<tr>
<td>4.6.1.6</td>
<td>FFLD Language</td>
<td>128</td>
</tr>
<tr>
<td>4.6.1.7</td>
<td>IL Language</td>
<td>128</td>
</tr>
<tr>
<td>4.6.1.8</td>
<td>ST Language</td>
<td>128</td>
</tr>
</tbody>
</table>

### 4.6.2 MUX8

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6.2.1</td>
<td>Inputs</td>
<td>129</td>
</tr>
<tr>
<td>4.6.2.2</td>
<td>Outputs</td>
<td>129</td>
</tr>
<tr>
<td>4.6.2.3</td>
<td>Truth Table</td>
<td>129</td>
</tr>
<tr>
<td>4.6.2.4</td>
<td>Remarks</td>
<td>129</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>4.6.2.5</td>
<td>ST Language</td>
<td>130</td>
</tr>
<tr>
<td>4.6.2.6</td>
<td>FBD Language</td>
<td>130</td>
</tr>
<tr>
<td>4.6.2.7</td>
<td>FFLD Language</td>
<td>130</td>
</tr>
<tr>
<td>4.6.2.8</td>
<td>IL Language</td>
<td>130</td>
</tr>
<tr>
<td>4.6.3</td>
<td>SEL</td>
<td>131</td>
</tr>
<tr>
<td>4.6.3.1</td>
<td>Inputs</td>
<td>131</td>
</tr>
<tr>
<td>4.6.3.2</td>
<td>Outputs</td>
<td>131</td>
</tr>
<tr>
<td>4.6.3.3</td>
<td>Truth Table</td>
<td>131</td>
</tr>
<tr>
<td>4.6.3.4</td>
<td>Remarks</td>
<td>131</td>
</tr>
<tr>
<td>4.6.3.5</td>
<td>ST Language</td>
<td>131</td>
</tr>
<tr>
<td>4.6.3.6</td>
<td>FBD Language</td>
<td>131</td>
</tr>
<tr>
<td>4.6.3.7</td>
<td>FFLD Language</td>
<td>131</td>
</tr>
<tr>
<td>4.6.3.8</td>
<td>IL Language</td>
<td>132</td>
</tr>
<tr>
<td>4.7</td>
<td>Registers</td>
<td>133</td>
</tr>
<tr>
<td>4.7.1</td>
<td>Standard Functions</td>
<td>133</td>
</tr>
<tr>
<td>4.7.2</td>
<td>Advanced Function</td>
<td>133</td>
</tr>
<tr>
<td>4.7.3</td>
<td>Bit-to-Bit Functions</td>
<td>133</td>
</tr>
<tr>
<td>4.7.4</td>
<td>Pack / Unpack Functions</td>
<td>133</td>
</tr>
<tr>
<td>4.7.5</td>
<td>Bit Access</td>
<td>134</td>
</tr>
<tr>
<td>4.7.6</td>
<td>Deprecated Functions</td>
<td>134</td>
</tr>
<tr>
<td>4.7.7</td>
<td>and_mask</td>
<td>134</td>
</tr>
<tr>
<td>4.7.7.1</td>
<td>Inputs</td>
<td>134</td>
</tr>
<tr>
<td>4.7.7.2</td>
<td>Outputs</td>
<td>135</td>
</tr>
<tr>
<td>4.7.7.3</td>
<td>Remarks</td>
<td>135</td>
</tr>
<tr>
<td>4.7.7.4</td>
<td>FBD Language</td>
<td>135</td>
</tr>
<tr>
<td>4.7.7.5</td>
<td>FFLD Language</td>
<td>135</td>
</tr>
<tr>
<td>4.7.7.6</td>
<td>IL Language</td>
<td>135</td>
</tr>
<tr>
<td>4.7.7.7</td>
<td>ST Language</td>
<td>135</td>
</tr>
<tr>
<td>4.7.8</td>
<td>HiByte</td>
<td>135</td>
</tr>
<tr>
<td>4.7.8.1</td>
<td>Inputs</td>
<td>136</td>
</tr>
<tr>
<td>4.7.8.2</td>
<td>Outputs</td>
<td>136</td>
</tr>
<tr>
<td>4.7.8.3</td>
<td>Remarks</td>
<td>136</td>
</tr>
<tr>
<td>4.7.8.4</td>
<td>FBD Language</td>
<td>136</td>
</tr>
<tr>
<td>4.7.8.5</td>
<td>FFLD Language</td>
<td>136</td>
</tr>
<tr>
<td>4.7.8.6</td>
<td>IL Language</td>
<td>136</td>
</tr>
<tr>
<td>4.7.8.7</td>
<td>ST Language</td>
<td>136</td>
</tr>
<tr>
<td>4.7.9</td>
<td>LoByte</td>
<td>137</td>
</tr>
<tr>
<td>4.7.9.1</td>
<td>Inputs</td>
<td>137</td>
</tr>
<tr>
<td>4.7.9.2</td>
<td>Outputs</td>
<td>137</td>
</tr>
<tr>
<td>4.7.9.3</td>
<td>Remarks</td>
<td>137</td>
</tr>
<tr>
<td>4.7.9.4</td>
<td>FBD Language</td>
<td>137</td>
</tr>
<tr>
<td>4.7.9.5</td>
<td>FFLD Language</td>
<td>137</td>
</tr>
<tr>
<td>4.7.9.6</td>
<td>IL Language</td>
<td>137</td>
</tr>
<tr>
<td>4.7.9.7</td>
<td>ST Language</td>
<td>137</td>
</tr>
<tr>
<td>4.7.10</td>
<td>HiWord</td>
<td>138</td>
</tr>
<tr>
<td>4.7.10.1</td>
<td>Inputs</td>
<td>138</td>
</tr>
<tr>
<td>4.7.10.2</td>
<td>Outputs</td>
<td>138</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Pages</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>4.7.16.2</td>
<td>Outputs</td>
<td>145</td>
</tr>
<tr>
<td>4.7.16.3</td>
<td>Remarks</td>
<td>145</td>
</tr>
<tr>
<td>4.7.16.4</td>
<td>FBD Language</td>
<td>145</td>
</tr>
<tr>
<td>4.7.16.5</td>
<td>FFLD Language</td>
<td>146</td>
</tr>
<tr>
<td>4.7.16.6</td>
<td>IL Language</td>
<td>146</td>
</tr>
<tr>
<td>4.7.16.7</td>
<td>ST Language</td>
<td>146</td>
</tr>
<tr>
<td>4.7.17</td>
<td>PACK8</td>
<td>146</td>
</tr>
<tr>
<td>4.7.17.1</td>
<td>Inputs</td>
<td>146</td>
</tr>
<tr>
<td>4.7.17.2</td>
<td>Outputs</td>
<td>146</td>
</tr>
<tr>
<td>4.7.17.3</td>
<td>Remarks</td>
<td>146</td>
</tr>
<tr>
<td>4.7.17.4</td>
<td>FBD Language</td>
<td>147</td>
</tr>
<tr>
<td>4.7.17.5</td>
<td>FFLD Language</td>
<td>147</td>
</tr>
<tr>
<td>4.7.17.6</td>
<td>IL Language</td>
<td>147</td>
</tr>
<tr>
<td>4.7.17.7</td>
<td>ST Language</td>
<td>147</td>
</tr>
<tr>
<td>4.7.18</td>
<td>rol</td>
<td>147</td>
</tr>
<tr>
<td>4.7.18.1</td>
<td>Inputs</td>
<td>148</td>
</tr>
<tr>
<td>4.7.18.2</td>
<td>Outputs</td>
<td>148</td>
</tr>
<tr>
<td>4.7.18.3</td>
<td>Remarks</td>
<td>148</td>
</tr>
<tr>
<td>4.7.18.4</td>
<td>FBD Language</td>
<td>148</td>
</tr>
<tr>
<td>4.7.18.5</td>
<td>FFLD Language</td>
<td>148</td>
</tr>
<tr>
<td>4.7.18.6</td>
<td>IL Language</td>
<td>148</td>
</tr>
<tr>
<td>4.7.18.7</td>
<td>ST Language</td>
<td>149</td>
</tr>
<tr>
<td>4.7.19</td>
<td>ror</td>
<td>149</td>
</tr>
<tr>
<td>4.7.19.1</td>
<td>Inputs</td>
<td>149</td>
</tr>
<tr>
<td>4.7.19.2</td>
<td>Outputs</td>
<td>149</td>
</tr>
<tr>
<td>4.7.19.3</td>
<td>Remarks</td>
<td>149</td>
</tr>
<tr>
<td>4.7.19.4</td>
<td>FBD Language</td>
<td>149</td>
</tr>
<tr>
<td>4.7.19.5</td>
<td>FFLD Language</td>
<td>149</td>
</tr>
<tr>
<td>4.7.19.6</td>
<td>IL Language</td>
<td>150</td>
</tr>
<tr>
<td>4.7.19.7</td>
<td>ST Language</td>
<td>150</td>
</tr>
<tr>
<td>4.7.20</td>
<td>RORb / ROR_SINT / ROR_USINT / ROR_BYTE</td>
<td>151</td>
</tr>
<tr>
<td>4.7.20.1</td>
<td>Inputs</td>
<td>151</td>
</tr>
<tr>
<td>4.7.20.2</td>
<td>Outputs</td>
<td>151</td>
</tr>
<tr>
<td>4.7.20.3</td>
<td>Diagram</td>
<td>151</td>
</tr>
<tr>
<td>4.7.20.4</td>
<td>Remarks</td>
<td>151</td>
</tr>
<tr>
<td>4.7.20.5</td>
<td>ST Language</td>
<td>151</td>
</tr>
<tr>
<td>4.7.20.6</td>
<td>FBD Language</td>
<td>151</td>
</tr>
<tr>
<td>4.7.20.7</td>
<td>FFLD Language</td>
<td>151</td>
</tr>
<tr>
<td>4.7.20.8</td>
<td>IL Language</td>
<td>151</td>
</tr>
<tr>
<td>4.7.20.9</td>
<td>See also</td>
<td>151</td>
</tr>
<tr>
<td>4.7.21</td>
<td>RORw / ROR_INT / ROR_UINT / ROR_WORD</td>
<td>152</td>
</tr>
<tr>
<td>4.7.21.1</td>
<td>Inputs</td>
<td>152</td>
</tr>
<tr>
<td>4.7.21.2</td>
<td>Outputs</td>
<td>152</td>
</tr>
<tr>
<td>4.7.21.3</td>
<td>Diagram</td>
<td>152</td>
</tr>
<tr>
<td>4.7.21.4</td>
<td>Remarks</td>
<td>152</td>
</tr>
<tr>
<td>4.7.21.5</td>
<td>ST Language</td>
<td>152</td>
</tr>
<tr>
<td>4.7.21.6</td>
<td>FBD Language</td>
<td>152</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>4.7.21.7</td>
<td>FFLD Language</td>
<td>152</td>
</tr>
<tr>
<td>4.7.21.8</td>
<td>IL Language</td>
<td>152</td>
</tr>
<tr>
<td>4.7.21.9</td>
<td>See also</td>
<td>152</td>
</tr>
<tr>
<td>4.7.22</td>
<td>SetBit</td>
<td>153</td>
</tr>
<tr>
<td>4.7.22.1</td>
<td>Inputs</td>
<td>153</td>
</tr>
<tr>
<td>4.7.22.2</td>
<td>Outputs</td>
<td>153</td>
</tr>
<tr>
<td>4.7.22.3</td>
<td>Remarks</td>
<td>153</td>
</tr>
<tr>
<td>4.7.22.4</td>
<td>FBD Language</td>
<td>153</td>
</tr>
<tr>
<td>4.7.22.5</td>
<td>FFLD Language</td>
<td>153</td>
</tr>
<tr>
<td>4.7.22.6</td>
<td>IL Language</td>
<td>153</td>
</tr>
<tr>
<td>4.7.22.7</td>
<td>ST Language</td>
<td>154</td>
</tr>
<tr>
<td>4.7.23</td>
<td>shl</td>
<td>154</td>
</tr>
<tr>
<td>4.7.23.1</td>
<td>Inputs</td>
<td>154</td>
</tr>
<tr>
<td>4.7.23.2</td>
<td>Outputs</td>
<td>154</td>
</tr>
<tr>
<td>4.7.23.3</td>
<td>Remarks</td>
<td>154</td>
</tr>
<tr>
<td>4.7.23.4</td>
<td>FBD Language</td>
<td>154</td>
</tr>
<tr>
<td>4.7.23.5</td>
<td>FFLD Language</td>
<td>154</td>
</tr>
<tr>
<td>4.7.23.6</td>
<td>IL Language</td>
<td>155</td>
</tr>
<tr>
<td>4.7.23.7</td>
<td>ST Language</td>
<td>155</td>
</tr>
<tr>
<td>4.7.24</td>
<td>shr</td>
<td>155</td>
</tr>
<tr>
<td>4.7.24.1</td>
<td>Inputs</td>
<td>155</td>
</tr>
<tr>
<td>4.7.24.2</td>
<td>Outputs</td>
<td>155</td>
</tr>
<tr>
<td>4.7.24.3</td>
<td>Remarks</td>
<td>155</td>
</tr>
<tr>
<td>4.7.24.4</td>
<td>FBD Language</td>
<td>156</td>
</tr>
<tr>
<td>4.7.24.5</td>
<td>FFLD Language</td>
<td>156</td>
</tr>
<tr>
<td>4.7.24.6</td>
<td>IL Language</td>
<td>156</td>
</tr>
<tr>
<td>4.7.24.7</td>
<td>ST Language</td>
<td>156</td>
</tr>
<tr>
<td>4.7.25</td>
<td>SWAB</td>
<td>156</td>
</tr>
<tr>
<td>4.7.25.1</td>
<td>Inputs</td>
<td>157</td>
</tr>
<tr>
<td>4.7.25.2</td>
<td>Outputs</td>
<td>157</td>
</tr>
<tr>
<td>4.7.25.3</td>
<td>Remarks</td>
<td>157</td>
</tr>
<tr>
<td>4.7.25.4</td>
<td>FBD Language</td>
<td>157</td>
</tr>
<tr>
<td>4.7.25.5</td>
<td>FFLD Language</td>
<td>157</td>
</tr>
<tr>
<td>4.7.25.6</td>
<td>IL Language</td>
<td>157</td>
</tr>
<tr>
<td>4.7.25.7</td>
<td>ST Language</td>
<td>157</td>
</tr>
<tr>
<td>4.7.26</td>
<td>TestBit</td>
<td>158</td>
</tr>
<tr>
<td>4.7.26.1</td>
<td>Inputs</td>
<td>158</td>
</tr>
<tr>
<td>4.7.26.2</td>
<td>Outputs</td>
<td>158</td>
</tr>
<tr>
<td>4.7.26.3</td>
<td>Remarks</td>
<td>158</td>
</tr>
<tr>
<td>4.7.26.4</td>
<td>FBD Language</td>
<td>158</td>
</tr>
<tr>
<td>4.7.26.5</td>
<td>FFLD Language</td>
<td>158</td>
</tr>
<tr>
<td>4.7.26.6</td>
<td>IL Language</td>
<td>158</td>
</tr>
<tr>
<td>4.7.26.7</td>
<td>ST Language</td>
<td>158</td>
</tr>
<tr>
<td>4.7.27</td>
<td>UNPACK8</td>
<td>159</td>
</tr>
<tr>
<td>4.7.27.1</td>
<td>Inputs</td>
<td>159</td>
</tr>
<tr>
<td>4.7.27.2</td>
<td>Outputs</td>
<td>159</td>
</tr>
<tr>
<td>4.7.27.3</td>
<td>Remarks</td>
<td>159</td>
</tr>
</tbody>
</table>
# Table of Contents

## 4.7 FBD Language

| 4.7.27.4 | xor_mask | 160 |
| 4.7.27.5 | FF | 159 |
| 4.7.27.6 | IL | 160 |
| 4.7.27.7 | ST | 160 |

## 4.7.28 xor_mask

| 4.7.28.1 | Inputs | 160 |
| 4.7.28.2 | Outputs | 160 |
| 4.7.28.3 | Remarks | 160 |
| 4.7.28.4 | FBD Language | 160 |
| 4.7.28.5 | FF | 160 |
| 4.7.28.6 | IL | 161 |
| 4.7.28.7 | ST | 161 |

## 4.8 Counters

| 4.8.1 | CTD / CTDr | 161 |
| 4.8.1.1 | Inputs | 161 |
| 4.8.1.2 | Outputs | 162 |
| 4.8.1.3 | Remarks | 162 |
| 4.8.1.4 | FBD Language | 162 |
| 4.8.1.5 | FF | 162 |
| 4.8.1.6 | IL | 162 |
| 4.8.1.7 | ST | 163 |

| 4.8.2 | CTU / CTUr | 163 |
| 4.8.2.1 | Inputs | 163 |
| 4.8.2.2 | Outputs | 163 |
| 4.8.2.3 | Remarks | 163 |
| 4.8.2.4 | FBD Language | 163 |
| 4.8.2.5 | FF | 164 |
| 4.8.2.6 | IL | 164 |
| 4.8.2.7 | ST | 164 |

| 4.8.3 | CTUD / CTUDr | 164 |
| 4.8.3.1 | Inputs | 164 |
| 4.8.3.2 | Outputs | 165 |
| 4.8.3.3 | Remarks | 165 |
| 4.8.3.4 | FBD Language | 165 |
| 4.8.3.5 | FF | 165 |
| 4.8.3.6 | IL | 165 |
| 4.8.3.7 | ST | 166 |

## 4.9 Timers

<p>| 4.9.1 | BLINK | 166 |
| 4.9.1.1 | Inputs | 166 |
| 4.9.1.2 | Outputs | 166 |
| 4.9.1.3 | Time diagram | 166 |
| 4.9.1.4 | Remarks | 167 |
| 4.9.1.5 | ST | 167 |
| 4.9.1.6 | FBD Language | 167 |
| 4.9.1.7 | FF | 167 |
| 4.9.1.8 | IL | 167 |</p>
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.9.7.7</td>
<td>IL Language</td>
<td>176</td>
</tr>
<tr>
<td>4.9.7.8</td>
<td>ST Language</td>
<td>176</td>
</tr>
<tr>
<td>4.9.8</td>
<td>TON</td>
<td>177</td>
</tr>
<tr>
<td>4.9.8.1</td>
<td>Inputs</td>
<td>177</td>
</tr>
<tr>
<td>4.9.8.2</td>
<td>Outputs</td>
<td>177</td>
</tr>
<tr>
<td>4.9.8.3</td>
<td>Time Diagram</td>
<td>177</td>
</tr>
<tr>
<td>4.9.8.4</td>
<td>Remarks</td>
<td>177</td>
</tr>
<tr>
<td>4.9.8.5</td>
<td>FBD Language</td>
<td>177</td>
</tr>
<tr>
<td>4.9.8.6</td>
<td>FFLD Language</td>
<td>177</td>
</tr>
<tr>
<td>4.9.8.7</td>
<td>IL Language</td>
<td>178</td>
</tr>
<tr>
<td>4.9.8.8</td>
<td>ST Language</td>
<td>178</td>
</tr>
<tr>
<td>4.9.9</td>
<td>TP / TPR</td>
<td>178</td>
</tr>
<tr>
<td>4.9.9.1</td>
<td>Inputs</td>
<td>178</td>
</tr>
<tr>
<td>4.9.9.2</td>
<td>Outputs</td>
<td>178</td>
</tr>
<tr>
<td>4.9.9.3</td>
<td>Time Diagram</td>
<td>178</td>
</tr>
<tr>
<td>4.9.9.4</td>
<td>Remarks</td>
<td>179</td>
</tr>
<tr>
<td>4.9.9.5</td>
<td>FBD Language</td>
<td>179</td>
</tr>
<tr>
<td>4.9.9.6</td>
<td>FFLD Language</td>
<td>179</td>
</tr>
<tr>
<td>4.9.9.7</td>
<td>IL Language</td>
<td>179</td>
</tr>
<tr>
<td>4.9.9.8</td>
<td>ST Language</td>
<td>179</td>
</tr>
<tr>
<td>4.10</td>
<td>Mathemetic Operations</td>
<td>180</td>
</tr>
<tr>
<td>4.10.1</td>
<td>abs / absL</td>
<td>180</td>
</tr>
<tr>
<td>4.10.1.1</td>
<td>Inputs</td>
<td>180</td>
</tr>
<tr>
<td>4.10.1.2</td>
<td>Outputs</td>
<td>180</td>
</tr>
<tr>
<td>4.10.1.3</td>
<td>Remarks</td>
<td>180</td>
</tr>
<tr>
<td>4.10.1.4</td>
<td>FBD Language</td>
<td>180</td>
</tr>
<tr>
<td>4.10.1.5</td>
<td>FFLD Language</td>
<td>181</td>
</tr>
<tr>
<td>4.10.1.6</td>
<td>IL Language</td>
<td>181</td>
</tr>
<tr>
<td>4.10.1.7</td>
<td>ST Language</td>
<td>181</td>
</tr>
<tr>
<td>4.10.2</td>
<td>expt</td>
<td>181</td>
</tr>
<tr>
<td>4.10.2.1</td>
<td>Inputs</td>
<td>181</td>
</tr>
<tr>
<td>4.10.2.2</td>
<td>Outputs</td>
<td>181</td>
</tr>
<tr>
<td>4.10.2.3</td>
<td>Remarks</td>
<td>181</td>
</tr>
<tr>
<td>4.10.2.4</td>
<td>FBD Language</td>
<td>182</td>
</tr>
<tr>
<td>4.10.2.5</td>
<td>FFLD Language</td>
<td>182</td>
</tr>
<tr>
<td>4.10.2.6</td>
<td>IL Language</td>
<td>182</td>
</tr>
<tr>
<td>4.10.2.7</td>
<td>ST Language</td>
<td>182</td>
</tr>
<tr>
<td>4.10.3</td>
<td>EXP / EXPL</td>
<td>182</td>
</tr>
<tr>
<td>4.10.3.1</td>
<td>Inputs</td>
<td>182</td>
</tr>
<tr>
<td>4.10.3.2</td>
<td>Outputs</td>
<td>182</td>
</tr>
<tr>
<td>4.10.3.3</td>
<td>Remarks</td>
<td>183</td>
</tr>
<tr>
<td>4.10.3.4</td>
<td>FBD Language</td>
<td>183</td>
</tr>
<tr>
<td>4.10.3.5</td>
<td>FFLD Language</td>
<td>183</td>
</tr>
<tr>
<td>4.10.3.6</td>
<td>IL Language</td>
<td>183</td>
</tr>
<tr>
<td>4.10.3.7</td>
<td>ST Language</td>
<td>183</td>
</tr>
<tr>
<td>4.10.4</td>
<td>LOG / LOGL</td>
<td>183</td>
</tr>
<tr>
<td>4.10.4.1</td>
<td>Inputs</td>
<td>183</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>4.10.4.2</td>
<td>Outputs</td>
<td>183</td>
</tr>
<tr>
<td>4.10.4.3</td>
<td>Remarks</td>
<td>184</td>
</tr>
<tr>
<td>4.10.4.4</td>
<td>ST Language</td>
<td>184</td>
</tr>
<tr>
<td>4.10.4.5</td>
<td>FBD Language</td>
<td>184</td>
</tr>
<tr>
<td>4.10.4.6</td>
<td>FFLD Language</td>
<td>184</td>
</tr>
<tr>
<td>4.10.4.7</td>
<td>IL Language</td>
<td>184</td>
</tr>
<tr>
<td>4.10.5</td>
<td>asLIN / LNL</td>
<td>184</td>
</tr>
<tr>
<td>4.10.5.1</td>
<td>Inputs</td>
<td>184</td>
</tr>
<tr>
<td>4.10.5.2</td>
<td>Outputs</td>
<td>184</td>
</tr>
<tr>
<td>4.10.5.3</td>
<td>Remarks</td>
<td>184</td>
</tr>
<tr>
<td>4.10.5.4</td>
<td>ST Language</td>
<td>185</td>
</tr>
<tr>
<td>4.10.5.5</td>
<td>FBD Language</td>
<td>185</td>
</tr>
<tr>
<td>4.10.5.6</td>
<td>FFLD Language</td>
<td>185</td>
</tr>
<tr>
<td>4.10.5.7</td>
<td>IL Language</td>
<td>185</td>
</tr>
<tr>
<td>4.10.6</td>
<td>POW ** POWL</td>
<td>185</td>
</tr>
<tr>
<td>4.10.6.1</td>
<td>Inputs</td>
<td>185</td>
</tr>
<tr>
<td>4.10.6.2</td>
<td>Outputs</td>
<td>185</td>
</tr>
<tr>
<td>4.10.6.3</td>
<td>Remarks</td>
<td>185</td>
</tr>
<tr>
<td>4.10.6.4</td>
<td>ST Language</td>
<td>185</td>
</tr>
<tr>
<td>4.10.6.5</td>
<td>FBD Language</td>
<td>185</td>
</tr>
<tr>
<td>4.10.6.6</td>
<td>FFLD Language</td>
<td>186</td>
</tr>
<tr>
<td>4.10.6.7</td>
<td>IL Language</td>
<td>186</td>
</tr>
<tr>
<td>4.10.7</td>
<td>ROOT</td>
<td>186</td>
</tr>
<tr>
<td>4.10.7.1</td>
<td>Inputs</td>
<td>186</td>
</tr>
<tr>
<td>4.10.7.2</td>
<td>Outputs</td>
<td>186</td>
</tr>
<tr>
<td>4.10.7.3</td>
<td>Remarks</td>
<td>186</td>
</tr>
<tr>
<td>4.10.7.4</td>
<td>ST Language</td>
<td>186</td>
</tr>
<tr>
<td>4.10.7.5</td>
<td>FBD Language</td>
<td>186</td>
</tr>
<tr>
<td>4.10.7.6</td>
<td>FFLD Language</td>
<td>187</td>
</tr>
<tr>
<td>4.10.7.7</td>
<td>IL Language</td>
<td>187</td>
</tr>
<tr>
<td>4.10.8</td>
<td>ScaleLin</td>
<td>187</td>
</tr>
<tr>
<td>4.10.8.1</td>
<td>Inputs</td>
<td>187</td>
</tr>
<tr>
<td>4.10.8.2</td>
<td>Outputs</td>
<td>187</td>
</tr>
<tr>
<td>4.10.8.3</td>
<td>Truth Table</td>
<td>187</td>
</tr>
<tr>
<td>4.10.8.4</td>
<td>Remarks</td>
<td>188</td>
</tr>
<tr>
<td>4.10.8.5</td>
<td>ST Language</td>
<td>188</td>
</tr>
<tr>
<td>4.10.8.6</td>
<td>FBD Language</td>
<td>188</td>
</tr>
<tr>
<td>4.10.8.7</td>
<td>FFLD Language</td>
<td>188</td>
</tr>
<tr>
<td>4.10.8.8</td>
<td>IL Language</td>
<td>188</td>
</tr>
<tr>
<td>4.10.9</td>
<td>SQRT / SQRTL</td>
<td>188</td>
</tr>
<tr>
<td>4.10.9.1</td>
<td>Inputs</td>
<td>188</td>
</tr>
<tr>
<td>4.10.9.2</td>
<td>Outputs</td>
<td>188</td>
</tr>
<tr>
<td>4.10.9.3</td>
<td>Remarks</td>
<td>188</td>
</tr>
<tr>
<td>4.10.9.4</td>
<td>ST Language</td>
<td>189</td>
</tr>
<tr>
<td>4.10.9.5</td>
<td>FBD Language</td>
<td>189</td>
</tr>
<tr>
<td>4.10.9.6</td>
<td>FFLD Language</td>
<td>189</td>
</tr>
<tr>
<td>4.10.9.7</td>
<td>IL Language</td>
<td>189</td>
</tr>
</tbody>
</table>
4.10.10 trunc / truncL ................................................................................. 190
4.10.10.1 Inputs ......................................................................................... 190
4.10.10.2 Outputs .................................................................................... 190
4.10.10.3 Remarks ................................................................................. 190
4.10.10.4 FBD Language ....................................................................... 190
4.10.10.5 FFLD Language .................................................................... 190
4.10.10.6 IL Language ........................................................................... 190
4.10.10.7 ST Language .......................................................................... 190

4.11 Trigonometric Functions ..................................................................... 191
4.11.1 acos / acosL .................................................................................. 191
4.11.1.1 Inputs ......................................................................................... 191
4.11.1.2 Outputs .................................................................................... 191
4.11.1.3 Remarks ................................................................................. 191
4.11.1.4 FBD Language ....................................................................... 191
4.11.1.5 FFLD Language .................................................................... 191
4.11.1.6 IL Language ........................................................................... 192
4.11.1.7 ST Language .......................................................................... 192
4.11.2 asin / asinL .................................................................................. 192
4.11.2.1 Inputs ......................................................................................... 192
4.11.2.2 Outputs .................................................................................... 192
4.11.2.3 Remarks ................................................................................. 192
4.11.2.4 FBD Language ....................................................................... 192
4.11.2.5 FFLD Language .................................................................... 193
4.11.2.6 IL Language ........................................................................... 193
4.11.2.7 ST Language .......................................................................... 193
4.11.3 atan / atanL .................................................................................. 193
4.11.3.1 Inputs ......................................................................................... 193
4.11.3.2 Outputs .................................................................................... 193
4.11.3.3 Remarks ................................................................................. 193
4.11.3.4 FBD Language ....................................................................... 194
4.11.3.5 FFLD Language .................................................................... 194
4.11.3.6 IL Language ........................................................................... 194
4.11.3.7 ST Language .......................................................................... 194
4.11.4 atan2 / atan2L ............................................................................. 194
4.11.4.1 Inputs ......................................................................................... 194
4.11.4.2 Outputs .................................................................................... 195
4.11.4.3 Remarks ................................................................................. 195
4.11.4.4 FBD Language ....................................................................... 195
4.11.4.5 FFLD Language .................................................................... 195
4.11.4.6 IL Language ........................................................................... 195
4.11.4.7 ST Language .......................................................................... 195
4.11.5 cos / cosL ..................................................................................... 195
4.11.5.1 Inputs ......................................................................................... 196
4.11.5.2 Outputs .................................................................................... 196
4.11.5.3 Remarks ................................................................................. 196
4.11.5.4 FBD Language ....................................................................... 196
4.11.5.5 FFLD Language .................................................................... 196
<p>| 4.11.5.6 | IL Language | 196 |
| 4.11.5.7 | ST Language | 196 |
| 4.11.6 | sin / sinL | 197 |
| 4.11.6.1 | Inputs | 197 |
| 4.11.6.2 | Outputs | 197 |
| 4.11.6.3 | Remarks | 197 |
| 4.11.6.4 | FBD Language | 197 |
| 4.11.6.5 | FFLD Language | 197 |
| 4.11.6.6 | IL Language | 197 |
| 4.11.6.7 | ST Language | 197 |
| 4.11.7 | tan / tanL | 198 |
| 4.11.7.1 | Inputs | 198 |
| 4.11.7.2 | Outputs | 198 |
| 4.11.7.3 | Remarks | 198 |
| 4.11.7.4 | FBD Language | 198 |
| 4.11.7.5 | FFLD Language | 198 |
| 4.11.7.6 | IL Language | 198 |
| 4.11.7.7 | ST Language | 199 |
| 4.11.8 | UseDegrees | 199 |
| 4.11.8.1 | Inputs | 199 |
| 4.11.8.2 | Outputs | 199 |
| 4.11.8.3 | Remarks | 199 |
| 4.11.8.4 | FBD Language | 199 |
| 4.11.8.5 | FFLD Language | 200 |
| 4.11.8.6 | IL Language | 200 |
| 4.11.8.7 | ST Language | 200 |
| 4.12 | String Operations | 200 |
| 4.12.1 | Standard Operators | 200 |
| 4.12.2 | Manage String Tables | 201 |
| 4.12.3 | ArrayToString / ArrayToStringU | 201 |
| 4.12.3.1 | Inputs | 201 |
| 4.12.3.2 | Outputs | 201 |
| 4.12.3.3 | Remarks | 201 |
| 4.12.3.4 | FBD Language | 201 |
| 4.12.3.5 | FFLD Language | 201 |
| 4.12.3.6 | IL Language | 202 |
| 4.12.3.7 | ST Language | 202 |
| 4.12.4 | ascii | 202 |
| 4.12.4.1 | Inputs | 202 |
| 4.12.4.2 | Outputs | 202 |
| 4.12.4.3 | Remarks | 202 |
| 4.12.4.4 | FBD Language | 202 |
| 4.12.4.5 | FFLD Language | 202 |
| 4.12.4.6 | IL Language | 203 |
| 4.12.4.7 | ST Language | 203 |
| 4.12.5 | ATOH | 203 |
| 4.12.5.1 | Inputs | 203 |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outputs</td>
<td></td>
<td>203</td>
</tr>
<tr>
<td>Truth Table</td>
<td></td>
<td>203</td>
</tr>
<tr>
<td>Remarks</td>
<td></td>
<td>203</td>
</tr>
<tr>
<td>FBD Language</td>
<td></td>
<td>204</td>
</tr>
<tr>
<td>FFLD Language</td>
<td></td>
<td>204</td>
</tr>
<tr>
<td>IL Language</td>
<td></td>
<td>204</td>
</tr>
<tr>
<td>ST Language</td>
<td></td>
<td>204</td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
<td>204</td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
<td>204</td>
</tr>
<tr>
<td>Remarks</td>
<td></td>
<td>205</td>
</tr>
<tr>
<td>FBD Language</td>
<td></td>
<td>205</td>
</tr>
<tr>
<td>FFLD Language</td>
<td></td>
<td>205</td>
</tr>
<tr>
<td>IL Language</td>
<td></td>
<td>205</td>
</tr>
<tr>
<td>ST Language</td>
<td></td>
<td>205</td>
</tr>
<tr>
<td>Concat</td>
<td></td>
<td>205</td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
<td>205</td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
<td>205</td>
</tr>
<tr>
<td>Remarks</td>
<td></td>
<td>206</td>
</tr>
<tr>
<td>FBD Language</td>
<td></td>
<td>206</td>
</tr>
<tr>
<td>FFLD Language</td>
<td></td>
<td>206</td>
</tr>
<tr>
<td>IL Language</td>
<td></td>
<td>206</td>
</tr>
<tr>
<td>ST Language</td>
<td></td>
<td>206</td>
</tr>
<tr>
<td>CRC16</td>
<td></td>
<td>206</td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
<td>206</td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
<td>206</td>
</tr>
<tr>
<td>Remarks</td>
<td></td>
<td>206</td>
</tr>
<tr>
<td>FBD Language</td>
<td></td>
<td>207</td>
</tr>
<tr>
<td>FFLD Language</td>
<td></td>
<td>207</td>
</tr>
<tr>
<td>IL Language</td>
<td></td>
<td>207</td>
</tr>
<tr>
<td>ST Language</td>
<td></td>
<td>207</td>
</tr>
<tr>
<td>Delete</td>
<td></td>
<td>207</td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
<td>207</td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
<td>207</td>
</tr>
<tr>
<td>Remarks</td>
<td></td>
<td>208</td>
</tr>
<tr>
<td>FBD Language</td>
<td></td>
<td>208</td>
</tr>
<tr>
<td>FFLD Language</td>
<td></td>
<td>208</td>
</tr>
<tr>
<td>IL Language</td>
<td></td>
<td>208</td>
</tr>
<tr>
<td>ST Language</td>
<td></td>
<td>208</td>
</tr>
<tr>
<td>FIND</td>
<td></td>
<td>208</td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
<td>209</td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
<td>209</td>
</tr>
<tr>
<td>Remarks</td>
<td></td>
<td>209</td>
</tr>
<tr>
<td>ST Language</td>
<td></td>
<td>209</td>
</tr>
<tr>
<td>FBD Language</td>
<td></td>
<td>209</td>
</tr>
<tr>
<td>FFLD Language</td>
<td></td>
<td>209</td>
</tr>
<tr>
<td>IL Language</td>
<td></td>
<td>209</td>
</tr>
</tbody>
</table>
4.12.11 HTOA ............................................................................................................. 210
  4.12.11.1 Inputs ........................................................................................................ 210
  4.12.11.2 Outputs ..................................................................................................... 210
  4.12.11.3 Truth Table .............................................................................................. 210
  4.12.11.4 Remarks ................................................................................................... 210
  4.12.11.5 FBD Language ........................................................................................ 210
  4.12.11.6 FFLD Language ....................................................................................... 210
  4.12.11.7 IL Language ............................................................................................. 210
  4.12.11.8 ST Language ............................................................................................ 211
4.12.12 INSERT .......................................................................................................... 211
  4.12.12.1 Inputs ........................................................................................................ 211
  4.12.12.2 Outputs ..................................................................................................... 211
  4.12.12.3 Remarks ................................................................................................... 211
  4.12.12.4 ST Language ............................................................................................ 211
  4.12.12.5 FBD Language ........................................................................................ 211
  4.12.12.6 FFLD Language ....................................................................................... 211
  4.12.12.7 IL Language ............................................................................................. 212
4.12.13 LEFT .............................................................................................................. 212
  4.12.13.1 Inputs ........................................................................................................ 212
  4.12.13.2 Outputs ..................................................................................................... 212
  4.12.13.3 Remarks ................................................................................................... 212
  4.12.13.4 ST Language ............................................................................................ 212
  4.12.13.5 FBD Language ........................................................................................ 212
  4.12.13.6 FFLD Language ....................................................................................... 212
  4.12.13.7 IL Language ............................................................................................. 213
4.12.14 LoadString ..................................................................................................... 213
  4.12.14.1 Inputs ........................................................................................................ 213
  4.12.14.2 Outputs ..................................................................................................... 213
  4.12.14.3 Remarks ................................................................................................... 213
  4.12.14.4 ST Language ............................................................................................ 213
  4.12.14.5 FBD Language ........................................................................................ 213
  4.12.14.6 FFLD Language ....................................................................................... 213
  4.12.14.7 IL Language ............................................................................................. 213
4.12.15 MID ................................................................................................................ 214
  4.12.15.1 Inputs ........................................................................................................ 214
  4.12.15.2 Outputs ..................................................................................................... 214
  4.12.15.3 Remarks ................................................................................................... 214
  4.12.15.4 ST Language ............................................................................................ 214
  4.12.15.5 FBD Language ........................................................................................ 214
  4.12.15.6 FFLD Language ....................................................................................... 214
  4.12.15.7 IL Language ............................................................................................. 214
4.12.16 MLEN ............................................................................................................ 215
  4.12.16.1 Inputs ........................................................................................................ 215
  4.12.16.2 Outputs ..................................................................................................... 215
  4.12.16.3 Remarks ................................................................................................... 215
  4.12.16.4 ST Language ............................................................................................ 215
  4.12.16.5 FBD Language ........................................................................................ 215
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.12.16.6 FFLD Language</td>
<td>215</td>
</tr>
<tr>
<td>4.12.16.7 IL Language</td>
<td>215</td>
</tr>
<tr>
<td>4.12.17 REPLACE</td>
<td>216</td>
</tr>
<tr>
<td>4.12.17.1 Inputs</td>
<td>216</td>
</tr>
<tr>
<td>4.12.17.2 Outputs</td>
<td>216</td>
</tr>
<tr>
<td>4.12.17.3 Remarks</td>
<td>216</td>
</tr>
<tr>
<td>4.12.17.4 ST Language</td>
<td>216</td>
</tr>
<tr>
<td>4.12.17.5 FBD Language</td>
<td>216</td>
</tr>
<tr>
<td>4.12.17.6 FFLD Language</td>
<td>217</td>
</tr>
<tr>
<td>4.12.17.7 IL Language</td>
<td>217</td>
</tr>
<tr>
<td>4.12.18 RIGHT</td>
<td>218</td>
</tr>
<tr>
<td>4.12.18.1 Inputs</td>
<td>218</td>
</tr>
<tr>
<td>4.12.18.2 Outputs</td>
<td>218</td>
</tr>
<tr>
<td>4.12.18.3 Remarks</td>
<td>218</td>
</tr>
<tr>
<td>4.12.18.4 ST Language</td>
<td>218</td>
</tr>
<tr>
<td>4.12.18.5 FBD Language</td>
<td>218</td>
</tr>
<tr>
<td>4.12.18.6 FFLD Language</td>
<td>218</td>
</tr>
<tr>
<td>4.12.18.7 IL Language</td>
<td>218</td>
</tr>
<tr>
<td>4.12.19 StringTable</td>
<td>220</td>
</tr>
<tr>
<td>4.12.19.1 Inputs</td>
<td>220</td>
</tr>
<tr>
<td>4.12.19.2 Outputs</td>
<td>220</td>
</tr>
<tr>
<td>4.12.19.3 Remarks</td>
<td>220</td>
</tr>
<tr>
<td>4.12.19.4 FBD Language</td>
<td>220</td>
</tr>
<tr>
<td>4.12.19.5 FFLD Language</td>
<td>220</td>
</tr>
<tr>
<td>4.12.19.6 IL Language</td>
<td>220</td>
</tr>
<tr>
<td>4.12.19.7 ST Language</td>
<td>221</td>
</tr>
<tr>
<td>4.12.19.8 String Table Resources</td>
<td>221</td>
</tr>
<tr>
<td>4.12.20 StringToArray / StringToArrayU</td>
<td>222</td>
</tr>
<tr>
<td>4.12.20.1 Inputs</td>
<td>222</td>
</tr>
<tr>
<td>4.12.20.2 Outputs</td>
<td>222</td>
</tr>
<tr>
<td>4.12.20.3 Remarks</td>
<td>222</td>
</tr>
<tr>
<td>4.12.20.4 FBD Language</td>
<td>222</td>
</tr>
<tr>
<td>4.12.20.5 FFLD Language</td>
<td>222</td>
</tr>
<tr>
<td>4.12.20.6 IL Language</td>
<td>222</td>
</tr>
<tr>
<td>4.12.20.7 ST Language</td>
<td>223</td>
</tr>
<tr>
<td>5 PLC Advanced Libraries</td>
<td>224</td>
</tr>
<tr>
<td>5.1 Analog Signal Processing</td>
<td>224</td>
</tr>
<tr>
<td>5.2 Alarm Management</td>
<td>224</td>
</tr>
<tr>
<td>5.3 Data Collections and Serialization</td>
<td>224</td>
</tr>
<tr>
<td>5.4 Data Log</td>
<td>225</td>
</tr>
<tr>
<td>5.5 Special Operations</td>
<td>225</td>
</tr>
<tr>
<td>5.6 Communication</td>
<td>225</td>
</tr>
<tr>
<td>5.7 Others</td>
<td>225</td>
</tr>
<tr>
<td>5.8 Alarm_A</td>
<td>225</td>
</tr>
<tr>
<td>5.8.1 Inputs</td>
<td>226</td>
</tr>
<tr>
<td>5.8.2 Outputs</td>
<td>226</td>
</tr>
<tr>
<td>5.8.3 Remarks</td>
<td>226</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>5.9.1</td>
<td>Alarm_M Inputs</td>
</tr>
<tr>
<td>5.9.2</td>
<td>Alarm_M Outputs</td>
</tr>
<tr>
<td>5.9.3</td>
<td>Alarm_M Remarks</td>
</tr>
<tr>
<td>5.9.3.1</td>
<td>Alarm_M Sequence</td>
</tr>
<tr>
<td>5.9.4</td>
<td>Alarm_M FBD Language</td>
</tr>
<tr>
<td>5.9.5</td>
<td>Alarm_M FFLD Language</td>
</tr>
<tr>
<td>5.9.6</td>
<td>Alarm_M IL Language</td>
</tr>
<tr>
<td>5.9.7</td>
<td>Alarm_M ST Language</td>
</tr>
<tr>
<td>5.10.1</td>
<td>ApplyRecipeColumn Inputs</td>
</tr>
<tr>
<td>5.10.2</td>
<td>ApplyRecipeColumn Outputs</td>
</tr>
<tr>
<td>5.10.3</td>
<td>ApplyRecipeColumn Remarks</td>
</tr>
<tr>
<td>5.10.4</td>
<td>ApplyRecipeColumn FBD Language</td>
</tr>
<tr>
<td>5.10.5</td>
<td>ApplyRecipeColumn FFLD Language</td>
</tr>
<tr>
<td>5.10.6</td>
<td>ApplyRecipeColumn IL Language</td>
</tr>
<tr>
<td>5.10.7</td>
<td>ApplyRecipeColumn ST Language</td>
</tr>
<tr>
<td>5.11.1</td>
<td>AS-interface Functions Interface</td>
</tr>
<tr>
<td>5.11.2</td>
<td>AS-interface Functions Arguments</td>
</tr>
<tr>
<td>5.12.1</td>
<td>average / averageL Inputs</td>
</tr>
<tr>
<td>5.12.2</td>
<td>average / averageL Outputs</td>
</tr>
<tr>
<td>5.12.3</td>
<td>average / averageL Remarks</td>
</tr>
<tr>
<td>5.13.1</td>
<td>CurveLin Inputs</td>
</tr>
<tr>
<td>5.13.2</td>
<td>CurveLin Outputs</td>
</tr>
<tr>
<td>5.13.3</td>
<td>CurveLin Remarks</td>
</tr>
<tr>
<td>5.14.1</td>
<td>derivate Inputs</td>
</tr>
<tr>
<td>5.14.2</td>
<td>derivate Outputs</td>
</tr>
<tr>
<td>5.14.3</td>
<td>derivate Remarks</td>
</tr>
<tr>
<td>5.15.1</td>
<td>EnableEvents Inputs</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Table of Contents

5.15.2 Outputs ........................................................................................................ 236  
5.15.3 Remarks ..................................................................................................... 236  
5.15.4 FBD Language ......................................................................................... 236  
5.15.5 FFLD Language ....................................................................................... 236  
5.15.6 IL Language .............................................................................................. 236  
5.15.7 ST Language ............................................................................................. 236  
5.16 FIFO ............................................................................................................... 237  
5.16.1 Inputs ........................................................................................................ 237  
5.16.2 Outputs ..................................................................................................... 237  
5.16.3 Remarks ..................................................................................................... 237  
5.16.4 FBD Language ......................................................................................... 238  
5.16.5 FFLD Language ....................................................................................... 238  
5.16.6 IL Language .............................................................................................. 238  
5.16.7 ST Language ............................................................................................. 238  
5.17 File Management .......................................................................................... 239  
5.17.1 SD Card Access ......................................................................................... 240  
5.17.2 SD Card Mounting Functions ................................................................... 241  
5.17.2.1 SD_MOUNT .......................................................................................... 241  
5.17.2.2 SD_UNMOUNT .................................................................................... 241  
5.17.2.3 SD_ISREADY ....................................................................................... 242  
5.17.3 File Path Conventions ............................................................................. 243  
5.17.3.1 File Name Warning and Limitations .................................................... 243  
5.17.3.2 Shared Directory Path Conventions .................................................... 244  
5.17.3.3 SD Card Path Conventions ................................................................ 245  
5.17.3.4 USB Flash Drive Path Conventions ..................................................... 245  
5.18 FilterOrder1 ................................................................................................... 246  
5.18.1 Inputs ........................................................................................................ 246  
5.18.2 Outputs ..................................................................................................... 246  
5.18.3 Remarks ..................................................................................................... 246  
5.18.4 Example .................................................................................................... 247  
5.18.5 FBD Language ......................................................................................... 247  
5.18.6 FFLD Language ....................................................................................... 247  
5.18.7 IL Language .............................................................................................. 247  
5.18.8 ST Language ............................................................................................. 247  
5.19 GetSysInfo .................................................................................................... 247  
5.19.1 Inputs ........................................................................................................ 247  
5.19.2 Outputs ..................................................................................................... 247  
5.19.3 Remarks ..................................................................................................... 248  
5.19.4 FBD Language ......................................................................................... 248  
5.19.5 FFLD Language ....................................................................................... 248  
5.19.6 IL Language .............................................................................................. 249  
5.19.7 ST Language ............................................................................................. 249  
5.20 hyster ............................................................................................................. 249  
5.20.1 Inputs ........................................................................................................ 249  
5.20.2 Outputs ..................................................................................................... 249  
5.20.3 Remarks ..................................................................................................... 249  
5.20.4 FBD Language ......................................................................................... 249
5.20.5 FFLD Language ........................................ 250
5.20.6 IL Language ........................................ 250
5.20.7 ST Language ........................................ 250
5.21 integral ........................................ 250
5.21.1 Inputs ........................................ 250
5.21.2 Outputs ........................................ 251
5.21.3 Remarks ........................................ 251
5.21.4 FBD Language ........................................ 251
5.21.5 FFLD Language ........................................ 251
5.21.6 IL Language ........................................ 251
5.21.7 ST Language ........................................ 252
5.22 LIFO ........................................ 252
5.22.1 Inputs ........................................ 252
5.22.2 Outputs ........................................ 252
5.22.3 Remarks ........................................ 252
5.22.4 ST Language ........................................ 253
5.22.5 FBD Language ........................................ 253
5.22.6 FFLD Language ........................................ 253
5.22.7 IL Language ........................................ 253
5.23 lim_alarm ........................................ 254
5.23.1 Inputs ........................................ 254
5.23.2 Outputs ........................................ 254
5.23.3 Remarks ........................................ 254
5.23.4 FBD Language ........................................ 254
5.23.5 FFLD Language ........................................ 255
5.23.6 IL Language ........................................ 255
5.23.7 ST Language ........................................ 255
5.24 LogFileCSV ........................................ 255
5.24.1 Inputs ........................................ 255
5.24.2 Outputs ........................................ 256
5.24.3 Remarks ........................................ 256
5.24.4 FBD Language ........................................ 257
5.24.5 FFLD Language ........................................ 257
5.24.6 IL Language ........................................ 257
5.24.7 ST Language ........................................ 257
5.25 PID ........................................ 258
5.25.1 Inputs ........................................ 258
5.25.2 Outputs ........................................ 258
5.25.3 Remarks ........................................ 259
5.25.3.1 Diagram ........................................ 259
5.25.4 FBD Language ........................................ 259
5.25.5 FFLD Language ........................................ 260
5.25.6 IL Language ........................................ 260
5.25.7 ST Language ........................................ 260
5.26 PWM ........................................ 261
5.26.1 Inputs ........................................ 261
5.26.2 Outputs ........................................ 261
5.26.3 Remarks ................................................................. 261
5.26.4 FBD Language ..................................................... 261
5.26.5 FFLD Language .................................................... 262
5.26.6 IL Language .......................................................... 262
5.26.7 ST Language .......................................................... 262
5.27 rand ................................................................. 262
  5.27.1 Inputs .............................................................. 262
  5.27.2 Outputs ............................................................ 262
  5.27.3 Remarks ........................................................... 262
  5.27.4 FBD Language .................................................... 263
  5.27.5 FFLD Language ................................................... 263
  5.27.6 IL Language ........................................................ 263
  5.27.7 ST Language ........................................................ 263
5.28 RAMP ................................................................. 263
  5.28.1 Inputs .............................................................. 263
  5.28.2 Outputs ............................................................ 263
  5.28.3 Time Diagram .................................................... 263
  5.28.4 Remarks ........................................................... 264
  5.28.5 ST Language ........................................................ 264
  5.28.6 FBD Language .................................................... 264
  5.28.7 FFLD Language ................................................... 264
  5.28.8 IL Language ........................................................ 265
5.29 Real Time Clock Management Functions ............... 265
  5.29.1 Time Zone and Clock Synchronization ................. 265
  5.29.2 Read the Real Time Clock .................................. 265
  5.29.3 Format the Present Date / Time ......................... 266
  5.29.4 Triggering Operations ................................. 266
  5.29.5 day_time ......................................................... 266
    5.29.5.1 Inputs ...................................................... 266
    5.29.5.2 Outputs ..................................................... 266
    5.29.5.3 Remarks .................................................... 267
    5.29.5.4 FBD Language ............................................. 267
    5.29.5.5 FFLD Language ............................................ 267
    5.29.5.6 IL Language ............................................... 267
    5.29.5.7 ST Language ............................................... 267
  5.29.6 DTAt ............................................................ 267
    5.29.6.1 Inputs ...................................................... 268
    5.29.6.2 Outputs ..................................................... 268
    5.29.6.3 Remarks .................................................... 268
    5.29.6.4 FBD Language ............................................. 268
    5.29.6.5 FFLD Language ............................................ 269
    5.29.6.6 IL Language ............................................... 269
    5.29.6.7 ST Language ............................................... 269
  5.29.7 DTCurDate ..................................................... 269
    5.29.7.1 Inputs ...................................................... 269
    5.29.7.2 Outputs ..................................................... 269
    5.29.7.3 ST Language ............................................... 269
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.29.8</td>
<td>DTCurDateTime</td>
<td>270</td>
</tr>
<tr>
<td>5.29.8.1</td>
<td>Inputs</td>
<td>270</td>
</tr>
<tr>
<td>5.29.8.2</td>
<td>Outputs</td>
<td>270</td>
</tr>
<tr>
<td>5.29.8.3</td>
<td>Remarks</td>
<td>270</td>
</tr>
<tr>
<td>5.29.8.4</td>
<td>FBD Language</td>
<td>270</td>
</tr>
<tr>
<td>5.29.8.5</td>
<td>FFLD Language</td>
<td>271</td>
</tr>
<tr>
<td>5.29.8.6</td>
<td>IL Language</td>
<td>271</td>
</tr>
<tr>
<td>5.29.8.7</td>
<td>ST Language</td>
<td>271</td>
</tr>
<tr>
<td>5.29.9</td>
<td>DTCurTime</td>
<td>272</td>
</tr>
<tr>
<td>5.29.9.1</td>
<td>Inputs</td>
<td>272</td>
</tr>
<tr>
<td>5.29.9.2</td>
<td>Output</td>
<td>272</td>
</tr>
<tr>
<td>5.29.9.3</td>
<td>ST Language</td>
<td>272</td>
</tr>
<tr>
<td>5.29.10</td>
<td>DTDay</td>
<td>272</td>
</tr>
<tr>
<td>5.29.10.1</td>
<td>Inputs</td>
<td>272</td>
</tr>
<tr>
<td>5.29.10.2</td>
<td>Outputs</td>
<td>272</td>
</tr>
<tr>
<td>5.29.10.3</td>
<td>ST Language</td>
<td>272</td>
</tr>
<tr>
<td>5.29.11</td>
<td>DTGetNTPServer</td>
<td>273</td>
</tr>
<tr>
<td>5.29.11.1</td>
<td>Inputs</td>
<td>273</td>
</tr>
<tr>
<td>5.29.11.2</td>
<td>Outputs</td>
<td>273</td>
</tr>
<tr>
<td>5.29.11.3</td>
<td>Remarks</td>
<td>273</td>
</tr>
<tr>
<td>5.29.11.4</td>
<td>FBD Language</td>
<td>273</td>
</tr>
<tr>
<td>5.29.11.5</td>
<td>FFLD Language</td>
<td>274</td>
</tr>
<tr>
<td>5.29.11.6</td>
<td>IL Language</td>
<td>274</td>
</tr>
<tr>
<td>5.29.11.7</td>
<td>ST Language</td>
<td>274</td>
</tr>
<tr>
<td>5.29.12</td>
<td>DTGetNTPSync</td>
<td>275</td>
</tr>
<tr>
<td>5.29.12.1</td>
<td>Inputs</td>
<td>275</td>
</tr>
<tr>
<td>5.29.12.2</td>
<td>Outputs</td>
<td>275</td>
</tr>
<tr>
<td>5.29.12.3</td>
<td>Remarks</td>
<td>275</td>
</tr>
<tr>
<td>5.29.12.4</td>
<td>FBD Language</td>
<td>275</td>
</tr>
<tr>
<td>5.29.12.5</td>
<td>FFLD Language</td>
<td>276</td>
</tr>
<tr>
<td>5.29.12.6</td>
<td>IL Language</td>
<td>276</td>
</tr>
<tr>
<td>5.29.12.7</td>
<td>ST Language</td>
<td>276</td>
</tr>
<tr>
<td>5.29.13</td>
<td>DTGetTimeZone</td>
<td>277</td>
</tr>
<tr>
<td>5.29.13.1</td>
<td>Inputs</td>
<td>277</td>
</tr>
<tr>
<td>5.29.13.2</td>
<td>Outputs</td>
<td>277</td>
</tr>
<tr>
<td>5.29.13.3</td>
<td>Remarks</td>
<td>277</td>
</tr>
<tr>
<td>5.29.13.4</td>
<td>FBD Language</td>
<td>277</td>
</tr>
<tr>
<td>5.29.13.5</td>
<td>FFLD Language</td>
<td>277</td>
</tr>
<tr>
<td>5.29.13.6</td>
<td>IL Language</td>
<td>278</td>
</tr>
<tr>
<td>5.29.13.7</td>
<td>ST Language</td>
<td>278</td>
</tr>
<tr>
<td>5.29.14</td>
<td>DTEvery</td>
<td>278</td>
</tr>
<tr>
<td>5.29.14.1</td>
<td>Inputs</td>
<td>278</td>
</tr>
<tr>
<td>5.29.14.2</td>
<td>Outputs</td>
<td>279</td>
</tr>
<tr>
<td>5.29.14.3</td>
<td>Remarks</td>
<td>279</td>
</tr>
<tr>
<td>5.29.14.4</td>
<td>FBD Language</td>
<td>279</td>
</tr>
<tr>
<td>5.29.14.5</td>
<td>FFLD Language</td>
<td>279</td>
</tr>
<tr>
<td>5.29.14.6</td>
<td>IL Language</td>
<td>279</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>5.29.14.7</td>
<td>ST Language</td>
<td>279</td>
</tr>
<tr>
<td>5.29.15</td>
<td>DTFormat</td>
<td>279</td>
</tr>
<tr>
<td>5.29.15.1</td>
<td>Inputs</td>
<td>280</td>
</tr>
<tr>
<td>5.29.15.2</td>
<td>Outputs</td>
<td>280</td>
</tr>
<tr>
<td>5.29.15.3</td>
<td>Remarks</td>
<td>280</td>
</tr>
<tr>
<td>5.29.15.4</td>
<td>FBD Language</td>
<td>280</td>
</tr>
<tr>
<td>5.29.15.5</td>
<td>FFLD Language</td>
<td>280</td>
</tr>
<tr>
<td>5.29.15.6</td>
<td>IL Language</td>
<td>281</td>
</tr>
<tr>
<td>5.29.15.7</td>
<td>ST Language</td>
<td>281</td>
</tr>
<tr>
<td>5.29.16</td>
<td>DTHour</td>
<td>281</td>
</tr>
<tr>
<td>5.29.16.1</td>
<td>Inputs</td>
<td>281</td>
</tr>
<tr>
<td>5.29.16.2</td>
<td>Outputs</td>
<td>281</td>
</tr>
<tr>
<td>5.29.16.3</td>
<td>ST Language</td>
<td>281</td>
</tr>
<tr>
<td>5.29.17</td>
<td>DTListTimeZones</td>
<td>281</td>
</tr>
<tr>
<td>5.29.17.1</td>
<td>Inputs</td>
<td>282</td>
</tr>
<tr>
<td>5.29.17.2</td>
<td>Outputs</td>
<td>282</td>
</tr>
<tr>
<td>5.29.17.3</td>
<td>Remarks</td>
<td>282</td>
</tr>
<tr>
<td>5.29.17.4</td>
<td>FBD Language</td>
<td>282</td>
</tr>
<tr>
<td>5.29.17.5</td>
<td>FFLD Language</td>
<td>282</td>
</tr>
<tr>
<td>5.29.17.6</td>
<td>IL Language</td>
<td>283</td>
</tr>
<tr>
<td>5.29.17.7</td>
<td>ST Language</td>
<td>283</td>
</tr>
<tr>
<td>5.29.18</td>
<td>DTMin</td>
<td>283</td>
</tr>
<tr>
<td>5.29.18.1</td>
<td>Inputs</td>
<td>283</td>
</tr>
<tr>
<td>5.29.18.2</td>
<td>Outputs</td>
<td>283</td>
</tr>
<tr>
<td>5.29.18.3</td>
<td>ST Language</td>
<td>284</td>
</tr>
<tr>
<td>5.29.19</td>
<td>DTMonth</td>
<td>284</td>
</tr>
<tr>
<td>5.29.19.1</td>
<td>Inputs</td>
<td>284</td>
</tr>
<tr>
<td>5.29.19.2</td>
<td>Outputs</td>
<td>284</td>
</tr>
<tr>
<td>5.29.19.3</td>
<td>ST Language</td>
<td>284</td>
</tr>
<tr>
<td>5.29.20</td>
<td>DTM</td>
<td>284</td>
</tr>
<tr>
<td>5.29.20.1</td>
<td>Inputs</td>
<td>284</td>
</tr>
<tr>
<td>5.29.20.2</td>
<td>Outputs</td>
<td>285</td>
</tr>
<tr>
<td>5.29.20.3</td>
<td>ST Language</td>
<td>285</td>
</tr>
<tr>
<td>5.29.21</td>
<td>DTSec</td>
<td>285</td>
</tr>
<tr>
<td>5.29.21.1</td>
<td>Inputs</td>
<td>285</td>
</tr>
<tr>
<td>5.29.21.2</td>
<td>Outputs</td>
<td>285</td>
</tr>
<tr>
<td>5.29.21.3</td>
<td>ST Language</td>
<td>285</td>
</tr>
<tr>
<td>5.29.22</td>
<td>DTSetDateTime</td>
<td>285</td>
</tr>
<tr>
<td>5.29.22.1</td>
<td>Inputs</td>
<td>285</td>
</tr>
<tr>
<td>5.29.22.2</td>
<td>Outputs</td>
<td>286</td>
</tr>
<tr>
<td>5.29.22.3</td>
<td>Remarks</td>
<td>286</td>
</tr>
<tr>
<td>5.29.22.4</td>
<td>FBD Language</td>
<td>286</td>
</tr>
<tr>
<td>5.29.22.5</td>
<td>FFLD Language</td>
<td>287</td>
</tr>
<tr>
<td>5.29.22.6</td>
<td>IL Language</td>
<td>287</td>
</tr>
<tr>
<td>5.29.22.7</td>
<td>ST Language</td>
<td>287</td>
</tr>
<tr>
<td>5.29.23</td>
<td>DTSetNTPServer</td>
<td>288</td>
</tr>
<tr>
<td>5.29.23.1</td>
<td>Inputs</td>
<td>288</td>
</tr>
</tbody>
</table>
5.29.23.2 Outputs .......................................................... 288
5.29.23.3 Remarks .......................................................... 288
5.29.23.4 FBD Language .................................................. 288
5.29.23.5 FFLD Language ............................................... 289
5.29.23.6 IL Language ..................................................... 289
5.29.23.7 ST Language ..................................................... 289
5.29.24 DTSetNTPSync ................................................... 289
  5.29.24.1 Inputs .......................................................... 290
  5.29.24.2 Outputs ........................................................ 290
  5.29.24.3 Remarks ........................................................ 290
  5.29.24.4 FBD Language ............................................... 290
  5.29.24.5 FFLD Language ............................................... 290
  5.29.24.6 IL Language ..................................................... 291
  5.29.24.7 ST Language ..................................................... 291
5.29.25 DTSetTimeZone .................................................. 291
  5.29.25.1 Inputs .......................................................... 291
  5.29.25.2 Outputs ........................................................ 291
  5.29.25.3 Remarks ........................................................ 292
  5.29.25.4 FBD Language ............................................... 292
  5.29.25.5 FFLD Language ............................................... 292
  5.29.25.6 IL Language ..................................................... 292
  5.29.25.7 ST Language ..................................................... 293
5.29.26 DTYear ............................................................ 293
  5.29.26.1 Inputs .......................................................... 293
  5.29.26.2 Outputs ........................................................ 293
  5.29.26.3 ST Language ..................................................... 293
5.29.27 List of Date / Time / NTP ErrorID Codes ....................... 293
5.30 SerializeIn ............................................................ 293
  5.30.1 Inputs .......................................................... 294
  5.30.2 Outputs .......................................................... 294
  5.30.3 Remarks .......................................................... 294
  5.30.4 FBD Language ............................................... 295
  5.30.5 FFLD Language ............................................... 295
  5.30.6 IL Language ..................................................... 295
  5.30.7 ST Language ..................................................... 295
  5.30.8 Arguments ...................................................... 295
    5.30.8.1 Input ...................................................... 295
    5.30.8.2 Output ...................................................... 296
5.31 SerializeOut .......................................................... 296
  5.31.1 Inputs .......................................................... 296
  5.31.2 Outputs .......................................................... 297
  5.31.3 Remarks .......................................................... 297
  5.31.4 FBD Language ............................................... 297
  5.31.5 FFLD Language ............................................... 298
  5.31.6 IL Language ..................................................... 298
  5.31.7 ST Language ..................................................... 298
  5.31.8 Arguments ...................................................... 298
# Table of Contents

5.31.8.1 Input ................................................................. 298  
5.31.8.2 Output ............................................................... 299  
5.32 SigID ................................................................. 299  
5.32.1 Inputs ............................................................. 299  
5.32.2 Outputs ............................................................ 299  
5.32.3 Remarks ............................................................ 299  
5.32.4 ST Language ....................................................... 299  
5.32.5 FBD Language ..................................................... 300  
5.32.6 FFLD Language .................................................... 300  
5.32.7 IL Language ......................................................... 300  
5.33 SigPlay ................................................................. 301  
5.33.1 Inputs ............................................................. 301  
5.33.2 Outputs ............................................................ 301  
5.33.3 Remarks ............................................................ 301  
5.33.4 ST Language ....................................................... 301  
5.33.5 FBD Language ..................................................... 301  
5.33.6 FFLD Language .................................................... 302  
5.33.7 IL Language ......................................................... 302  
5.34 SigScale ................................................................. 303  
5.34.1 Inputs ............................................................. 303  
5.34.2 Outputs ............................................................ 303  
5.34.3 Remarks ............................................................ 303  
5.34.4 FBD Language ..................................................... 303  
5.34.5 FFLD Language .................................................... 303  
5.34.6 IL Language ......................................................... 303  
5.34.7 ST Language ......................................................... 303  
5.35 stackint ................................................................. 304  
5.35.1 Inputs ............................................................. 304  
5.35.2 Outputs ............................................................ 304  
5.35.3 Remarks ............................................................ 304  
5.35.4 FBD Language ..................................................... 304  
5.35.5 FFLD Language .................................................... 305  
5.35.6 IL Language ......................................................... 305  
5.35.7 ST Language ......................................................... 305  
5.36 SurfLin ................................................................. 305  
5.36.1 Inputs ............................................................. 306  
5.36.2 Outputs ............................................................ 306  
5.36.3 Remarks ............................................................ 306  
5.37 VLID ................................................................. 307  
5.37.1 Inputs ............................................................. 307  
5.37.2 Outputs ............................................................ 307  
5.37.3 Remarks ............................................................ 307  
5.37.4 FBD Language ..................................................... 307  
5.37.5 FFLD Language .................................................... 307  
5.37.6 IL Language ......................................................... 308  
5.37.7 ST Language ......................................................... 308  
6 Support and Services .................................................... 309
3 Programming Languages

This section provides information about the syntax, structure, and use of declarations and statements supported by the KAS IDE application language.

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

- "Free Form Ladder Diagram (FFLD)" (➔ p. 47)
- "Function Block Diagram (FBD)" (➔ p. 41)
- "Instruction List (IL)" (➔ p. 42)
- "Sequential Function Chart (SFC)" (➔ p. 35)
- "Structured Text (ST)" (➔ p. 44)

A language must be selected for each program or User-Defined Function Block of the application.

**NOTE**

When using FFLD or FBD languages, review Use of ST Expressions in Graphic Language.

3.1 Sequential Function Chart (SFC)

The SFC language is a state diagram.

- Graphical steps are used to represent stable states.
- Transitions describe the conditions and events that lead to a change of state.
- Using SFC simplifies the programming of sequential operations because it saves a lot of variables and tests just for maintaining the program context.

**IMPORTANT**

Do 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.

These are basic components of an SFC chart:

<table>
<thead>
<tr>
<th>Chart</th>
<th>Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFC Steps</td>
<td>Actions in an SFC Step</td>
</tr>
<tr>
<td>SFC Transitions</td>
<td>Time out on an SFC Step</td>
</tr>
<tr>
<td>Create SFC Parallel Branches</td>
<td>Condition of an SFC Transition</td>
</tr>
<tr>
<td>Jump to an SFC Step</td>
<td></td>
</tr>
</tbody>
</table>

The KAS IDE fully supports SFC programming with several hierarchical levels of charts (e.g., 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.

See these sections for more information:

- "Hierarchy of SFC Programs" (➔ p. 40)
- "Control an SFC Child Program" (➔ p. 40)

3.1.1 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.

- In 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.

**Example**
Execution order:

- Evaluate transitions:
  - 101
  - 102

- Manage steps:
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 these rules:

- A transition is crossed if:
  - Its condition is TRUE and 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.

3.1.1.1 Divergence

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

3.1.1.1.1 Order of Action Block Execution

For a given cycle, if a transition is:

- FALSE, the N-action blocks are evaluated for all active steps waiting on that transition.
- TRUE, the P0 action blocks are evaluated for all active steps waiting on that transition, followed by the P1 and N steps for all steps waiting on that transition.
  - The steps that were waiting on that transition are then marked as active.

Example

Using this SFC:
The order of action block execution for a given cycle is:

<table>
<thead>
<tr>
<th>Incoming State</th>
<th>Evaluating Transition</th>
<th>If Transition is FALSE</th>
<th>If Transition is TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Cycle</td>
<td>N/A</td>
<td>[1] P1</td>
<td>[1] P0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[1] N</td>
<td>[101] P1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[101] N</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[201] P1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[201] N</td>
</tr>
<tr>
<td>Transition 1.</td>
<td>Transition 1</td>
<td>[1] N</td>
<td>[101] P0</td>
</tr>
<tr>
<td>Not yet TRUE.</td>
<td></td>
<td></td>
<td>[101] N</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[201] P1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[201] N</td>
</tr>
<tr>
<td>Passed transition 1.</td>
<td>Transition 101</td>
<td>[101] N</td>
<td>[101] P0</td>
</tr>
<tr>
<td>Transition 101</td>
<td></td>
<td></td>
<td>[201] P1</td>
</tr>
<tr>
<td>not yet TRUE.</td>
<td></td>
<td></td>
<td>[201] N</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[102] P1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[102] N</td>
</tr>
<tr>
<td>Passed transitions 1 and 101</td>
<td>Transition 2</td>
<td>[201] N</td>
<td>[201] P0</td>
</tr>
<tr>
<td>Transition 2</td>
<td></td>
<td></td>
<td>[102] P0</td>
</tr>
<tr>
<td>not yet TRUE.</td>
<td></td>
<td></td>
<td>[102] N</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[3] P1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[3] N</td>
</tr>
</tbody>
</table>

**IMPORTANT**
Execution of SFC in the IEC 61131-3 target is sampled according to the target cycles. When a transition is crossed within a cycle, these steps are activated.
The evaluation of the chart continues in the next cycle. If several consecutive transitions are TRUE within a branch, only one of them is crossed within one target cycle.

**IMPORTANT**

Some runtime systems may not support exclusivity of the transitions within an divergence. See the OEM instructions for more information about SFC support.

### 3.1.2 Hierarchy of SFC Programs

Each SFC program can have one or more child programs.

- Child programs are written in SFC.
- They are started or stopped in the actions of the parent program.
- The number of hierarchy levels must not exceed 19.
- A child program can also have children.
  - When a child program is stopped, its children are also stopped.
  - When a child program is started, it must 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.
    - very useful for designing a complex process and separate operations due to different aspects of the process.
    - Example: It is common to manage the execution modes in a parent program and to handle details of the process operations in child programs.

### 3.1.3 Control an SFC Child Program

Controlling a child program can be simply achieved by specifying the name of the child program as an action block in a step of its parent program.

These are possible qualifiers that can be applied to an action block for handling a child program:

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

Alternatively, use these statements in an action block programmed in ST language.

In this table, "prog" represents the name of the child program:

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

Use the **GSTATUS** function in expressions.

This function returns the current state of a child SFC program:

- **GSTATUS (prog)**
  - Returns the current state of a child SFC program:
    - 0: program is inactive
    - 1: program is active
    - 2: program is suspended
When a child program is started by its parent program, it keeps the "inactive" status until it is executed (further in the cycle). If a child program is started in an SFC chart, GSTATUS returns 1 (active) on the next cycle.

3.2 Function Block Diagram (FBD)

A function block diagram is a data flow between constant expressions or variables and operations represented by rectangular blocks.

- Operations can be basic operations, function calls, or function block calls.
- See Use of ST Expressions in Graphic Language for more information.
- The name of the operation or function, or the type of function block is written within the block rectangle.
- With a function block call, the name of the called instance is written in the header of the block rectangle.

Example

3.2.1 Data Flow

- The data flow represents values of any data type.
  - All connections must be from input and outputs points having the same data type.
- With a Boolean connection, use a connection link terminated by a small circle.
  - This indicates a Boolean negation of the data flow.

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

3.2.2 FFLD Symbols

FFLD symbols can also be entered in FBD diagrams and linked to FBD objects.

- See these sections for information about components of the FFLD language:
  - "FFLD Contacts" (➔ p. 49)
  - "FFLD Coils" (➔ p. 50)
  - Power Rails
- Special vertical lines are available in FBD language for representing the merging of FFLD parallel lines.
  - Such vertical lines represent a OR operation between the connected inputs.

Example of an OR vertical line used in a FBD diagram:
3.3 Instruction List (IL)

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

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

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

KAS IDE allows you to mix ST and IL languages in textual program.

- ST Language is the default language.
- When you enter IL Language instructions, the program must be entered between BEGIN_IL and END_IL keywords.

Example

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

3.3.1 Comments

Comment text can be entered at the end of a line containing an instruction.

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

Example

```plaintext
(* My comment *)
LD a
ST b  (* Store value in d *)
```
first: evaluating an expression (called current result).
then: use the current result for performing actions.

3.3.3 Evaluation of Expressions
The order of instructions in the program is the one used for evaluating expressions, unless parentheses are inserted.

This list is the available instructions for evaluation of expressions:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operand</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Addition +&quot; (→ p. 89)</td>
<td>Numerical</td>
<td>Performs an addition of all inputs. Adds the operand and the current result.</td>
</tr>
<tr>
<td>AND, ANDN, &amp;</td>
<td>Boolean</td>
<td>AND between the operand and the current result.</td>
</tr>
<tr>
<td>&quot;Divide /&quot; (→ p. 90)</td>
<td>Numerical</td>
<td>Performs a division of all inputs. Divide the current result by the operand.</td>
</tr>
<tr>
<td>&quot;EQ =&quot; (→ p. 107)</td>
<td>Numerical</td>
<td>Compares the current result with the operand.</td>
</tr>
<tr>
<td>&quot;Assignment :=&quot; (→ p. 54)</td>
<td>Any type</td>
<td>Loads the operand in the current result.</td>
</tr>
<tr>
<td>Function Call</td>
<td>Functional Arguments</td>
<td>Calls a function.</td>
</tr>
<tr>
<td>&quot;GE &gt;=&quot; (→ p. 105)</td>
<td>Numerical</td>
<td>Compares the current result with the operand.</td>
</tr>
<tr>
<td>&quot;GT &gt;&quot; (→ p. 106)</td>
<td>Numerical</td>
<td>Compares the current result with the operand.</td>
</tr>
<tr>
<td>&quot;LE &lt;=&quot; (→ p. 110)</td>
<td>Numerical</td>
<td>Compares the current result with the operand.</td>
</tr>
<tr>
<td>&quot;LT &lt;&quot; (→ p. 111)</td>
<td>Numerical</td>
<td>Compares the current result with the operand.</td>
</tr>
<tr>
<td>&quot;Multiply *&quot; (→ p. 99)</td>
<td>Numerical</td>
<td>Performs a multiplication of all inputs. Multiply the operand and the current result.</td>
</tr>
<tr>
<td>&quot;NE &lt;&gt;&quot; (→ p. 109)</td>
<td>Numerical</td>
<td>Compares the current result with the operand.</td>
</tr>
<tr>
<td>OR, ORN</td>
<td>Boolean</td>
<td>Performs a logical OR of all inputs. OR between the operand and the current result.</td>
</tr>
<tr>
<td>&quot;Parenthesis ( )&quot; (→ p. 68)</td>
<td></td>
<td>Changes the execution order.</td>
</tr>
<tr>
<td>&quot;Subtraction -&quot; (→ p. 102)</td>
<td>Numerical</td>
<td>Performs a subtraction of all inputs. Subtract the operand from the current result.</td>
</tr>
<tr>
<td>&quot;XOR / XORN&quot; (→ p. 87)</td>
<td>Boolean</td>
<td>XOR between the operand and the current result.</td>
</tr>
</tbody>
</table>

**NOTE**

Instructions suffixed by N uses the Boolean negation of the operand.
3.3.4 Actions
These instructions perform actions according to the value of current result.
Some of these instructions do not need a current result to be evaluated.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operand</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAL</td>
<td>f. block</td>
<td>Calls a function block (no current result needed).</td>
</tr>
<tr>
<td>CALC</td>
<td>f. block</td>
<td>Calls a function block if the current result is TRUE.</td>
</tr>
<tr>
<td>CALNC / CALCN</td>
<td>f. block</td>
<td>Calls a function block if the current result is FALSE.</td>
</tr>
<tr>
<td>JMP</td>
<td>label</td>
<td>Jump to a label - no current result needed.</td>
</tr>
<tr>
<td>J MPC</td>
<td>label</td>
<td>Jump to a label if the current result is TRUE.</td>
</tr>
<tr>
<td>J MPCN / J MP CN</td>
<td>label</td>
<td>Jump to a label if the current result is FALSE.</td>
</tr>
<tr>
<td>R</td>
<td>Boolean</td>
<td>Sets the operand to FALSE if the current result is TRUE.</td>
</tr>
<tr>
<td>RET</td>
<td></td>
<td>Jump to the end of the current program - no current result needed.</td>
</tr>
<tr>
<td>RETC / RET NC / RET CN</td>
<td></td>
<td>Jump to the end of the current program if the current result is TRUE / FALSE.</td>
</tr>
<tr>
<td>S</td>
<td>Boolean</td>
<td>Sets the operand to TRUE if the current result is TRUE.</td>
</tr>
<tr>
<td>ST / STN</td>
<td>Any type</td>
<td>Stores the current result in the operand.</td>
</tr>
</tbody>
</table>

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

NOTE
An IL Language program cannot be called if there is no entry variable or if it's type is complex (e.g., array).

3.4 Structured Text (ST)
ST is a structured literal programming language.

- A ST program is a list of statements.
  - Each statement describes an action and must end with a semi-colon (;).
  - The presentation of the text has no meaning for a ST program. You can insert blank characters and line breaks where you want in the program text.

3.4.1 Comments
Comment text can be entered anywhere in a ST program.

- Comment text:
  - Has no meaning for the execution of the program.
  - Must begin with "(" and end with ")".
  - Can be entered on several lines (i.e., a comment text can include line breaks).
  - Cannot be nested.

You can also use // to add a comment on a single line:

```
//My main comment
(* My comment.*)
```
3.4.2 Expressions
Each statement describes an action and can include evaluation of complex expressions.
An expression is evaluated:
- From the left to the right.
- According to the default priority order of operators.
  - The default priority can be changed using parentheses "Parenthesis ( )" (→ p. 68).
Arguments of an expression can be:
- Declared Variables.
- Constant Expressions.
- Call a Function.

3.4.3 Statements
3.4.3.1 Basic Statements
These are the available basic statements that can be entered in a ST program:
- "Assignment :=" (→ p. 54) (assignment)
  - Call a Function Block
3.4.3.2 Conditional Statements
These are the available conditional statements in ST language:
- "CASE OF ELSE END_CASE" (→ p. 57)

```
CASE iChoice OF
  0: MyString := 'Nothing';
  1 .. 2,5: MyString := 'First case';
  3,4: MyString := 'Second case';
ELSE MyString := 'Other case';
END_CASE;
```
• "IF THEN ELSE ELSIF END_IF" (→ p. 63)
  
  Simple binary switch.
  One or several ELSIF are allowed.
  
  ```
  IF a = b THEN
    c := 0;
  ELSIF a < b THEN
    c := 1;
  ELSE
    c := -1;
  END_IF;
  ```

3.4.3.3 Loop Statements
These are the available statements for describing loops in ST language:

1. **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.

• "FOR TO BY END_FOR" (→ p. 62)
  - Loops with FOR instructions are slow.
    Optimize your code by replacing such iterations with a WHILE statement.

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

• "REPEAT UNTIL END_REPEAT" (→ p. 69)

  Repeat a list of statements.
  Condition is evaluated on loop exit after the statements.
  ```
  iCount := 0;
  REPEAT
    MyVar := MyVar + 1;
    iCount := iCount + 1;
  UNTIL iCount < 100 END_REPEAT;
  ```

• "WHILE DO END_WHILE" (→ p. 73)

  Repeat a list of statements.
  Condition is evaluated on loop entry before the statements.
  ```
  iCount := 0;
  WHILE iCount < 100 DO
    iCount := iCount + 1;
    MyVar := MyVar + 1;
  END_WHILE;
  ```

3.4.3.4 Other Statements
These are some other statements in ST language:

- "WAIT / WAIT_TIME" (→ p. 71) - Suspend the execution.
- "ON" (→ p. 67) - Conditional execution of statements: provides a simpler syntax for checking the rising edge of a Boolean condition.

### 3.4.3.5 Helpful Features
This content is in the Copa-Data content but not ours - is it relevant for us?

There are some features to simplify the programming:

- A click on the "{" keyboard button automatically inserts the "}" symbol, if the previous word is an instance or a function name. The caret is put between the brackets.
- Hint: Function blocks and UDFBs cannot be used in ST Language without an instance name. If a word is located before a function block or a UDFB the feature is not available.
- A click on the "{" keyboard button automatically inserts the "}" symbol. The caret is put between the two [].

**TIP**

ST provides an automatic completion of typed words.
See Auto-completion of Words for more information.

### 3.5 Free Form Ladder Diagram (FFLD)

A Ladder Diagram is a list of rungs.

- Each rung represents a Boolean data flow from a power rail on the left. The power rail represents the TRUE state. The data flow must be understood from the left to the right. Each symbol connected to the rung either changes the rung state or performs an operation.
- These are possible graphic items to be entered in FFLD diagrams:
  - Power Rails
  - Contacts and Coils
  - Operations, Functions and Function blocks, represented by rectangular blocks.
    - See Call a Function or Call a Function Block.
  - LABELS and Jumps JMP JMPC JMPNC JMPCN
  - Use of ST Expressions in Graphic Language

#### 3.5.1 Use of EN Input and 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.

#### 3.5.1.1 Examples

- This is an example of "XOR / XORN" (→ p. 87) with Boolean inputs and outputs and requiring no EN or ENO pin.
• First input is the rung.
• The rung is the output.

This is an example of the "GT >" (→ p. 106) (greater than) with 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.

This is an example of the "SEL" (→ p. 131) with 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.

This is an example of "Addition +" (→ p. 89) 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 has the same value as EN.

### 3.5.2 Contacts and Coils

This is a list of the contact and coil types available:

<table>
<thead>
<tr>
<th>Contacts</th>
<th>Coils</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negative Transition -[N]-</td>
<td>Reset (Unlatch) - (R)-</td>
</tr>
<tr>
<td>Normally Closed -[/]-</td>
<td>De-energize - (/)-</td>
</tr>
<tr>
<td>Normally closed negative transition -[/N]-</td>
<td>Negative transition sensing coil - (N)-</td>
</tr>
<tr>
<td>Normally closed positive transition -[/P]-</td>
<td>Positive transition sensing coil - (P)-</td>
</tr>
<tr>
<td>Normally Open - [ ]-</td>
<td>Energize - ( )-</td>
</tr>
<tr>
<td>Positive Transition -[P]-</td>
<td>Set (Latch) - (S)-</td>
</tr>
</tbody>
</table>
3.5.2.1 FFLD Contacts

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

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

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

**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:

3.5.2.2 FFLD Coils

Coils are basic graphic elements of the FFLD language.

- A coil is associated with a Boolean variable displayed above the graphic symbol.
- A coil performs a change of the associated variable according to the flow on its left-hand side.
- The six possible coil symbols are:
<table>
<thead>
<tr>
<th>Variable</th>
<th>Coils</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negated</td>
<td>boolVariable - (/)-</td>
<td>The associated variable is forced to the negation of the flow on the left of the coil.</td>
</tr>
<tr>
<td>Negative Transition</td>
<td>boolVariable - (N)-</td>
<td>The associated variable is forced to TRUE if the flow on the left changes from TRUE to FALSE (and forced to FALSE in all other cases).</td>
</tr>
<tr>
<td>Normal</td>
<td>boolVariable - ( )-</td>
<td>The associated variable is forced to the value of the flow on the left of the coil.</td>
</tr>
<tr>
<td>Positive Transition</td>
<td>boolVariable - (P)-</td>
<td>The associated variable is forced to TRUE if the flow on the left changes from FALSE to TRUE (and forced to FALSE in all other cases).</td>
</tr>
<tr>
<td>Reset</td>
<td>boolVariable - (R)-</td>
<td>The associated variable is forced to FALSE if the flow on the left is TRUE. No action if the rung state is FALSE.</td>
</tr>
</tbody>
</table>

**Rules for Reset coil animation:**
- Power Flow on left is TRUE:
  - The horizontal lines are red
  - The variable above (R) is black
  - The R and the circle around the R are black
- Power Flow on left is FALSE and variable above reset coil is NOT Energized (OFF):
  - The horizontal lines are black
  - The variable above (R) is black
  - The R and the circle around the R are black
- Power Flow on left is FALSE and variable above reset coil is Energized (ON):
  - The horizontal lines are black
  - The variable above (R) is red
  - The R and the circle around the R are red
<table>
<thead>
<tr>
<th>Variable</th>
<th>Coils</th>
<th>Description</th>
</tr>
</thead>
</table>
| Set      | boolVariable
          -(S)-     | The associated variable is forced to TRUE if the flow on the left is TRUE. No action if the flow is FALSE. |

**Rules for Set coil animation:**
- Power Flow on left is TRUE:
  - The horizontal wires on either side of the (S) are red
  - The variable and the (S) are red
- Power Flow on left is FALSE and the (S) variable is Energized (ON)
  - The horizontal lines on either sided of (S) are black
  - The variable and the (S) are red
- In all other cases:
  - The horizontal wires are black
  - The variable and the (S) are black

💡 **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.

ℹ️ **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.
4 PLC Standard Libraries

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

- "Basic Operations" (→ p. 54)
- "Boolean Operations" (→ p. 74)
- "Arithmetic Operations" (→ p. 89)
- "Comparison Operations" (→ p. 103)
- "Type Conversion Functions" (→ p. 112)
- "Selectors" (→ p. 127)
- "Registers" (→ p. 133)
- "Counters" (→ p. 161)
- "Timers" (→ p. 166)
- "Mathematic Operations" (→ p. 180)
- "Trigonometric Functions" (→ p. 191)
- "String Operations" (→ p. 200)
- "PLC Advanced Libraries" (→ p. 224)

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

4.1.1 Data Manipulation
These are the language features for basic data manipulation:

- "Assignment :=" (→ p. 54)
- "Bit Access" (→ p. 56)
- Call a Function
- Call a Function Block
- "Call a Sub-Program" (→ p. 56)
- "CountOf" (→ p. 59)
- "DEC" (→ p. 60)
- "INC" (→ p. 64)
- "MoveBlock" (→ p. 65)
- "NEG -" (→ p. 92) integer negation (unary operator)
- "Parenthesis ( )" (→ p. 68)

4.1.2 Control Program Execution

4.1.2.1 Language Features
These are the language features to control program execution:

- Jumps JMP JMPC JMPCN
- LABELS
- "RETURN RET RETC RETNC RETCN" (→ p. 70)

4.1.2.2 Structured Statements
These are the structured statements to control program execution:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;CASE OF ELSE END_CASE&quot; (→ p. 57)</td>
<td>Switch to one of various possible statements.</td>
</tr>
<tr>
<td>&quot;EXIT&quot; (→ p. 61)</td>
<td>Exit from a loop instruction.</td>
</tr>
<tr>
<td>&quot;FOR TO BY END_FOR&quot; (→ p. 62)</td>
<td>Execute iterations of statements.</td>
</tr>
<tr>
<td>&quot;IF THEN ELSE ELSIF END_IF&quot; (→ p. 63)</td>
<td>Conditional execution of statements.</td>
</tr>
<tr>
<td>&quot;ON&quot; (→ p. 67)</td>
<td>Conditional execution of statements.</td>
</tr>
<tr>
<td>&quot;REPEAT UNTIL END_REPEAT&quot; (→ p. 69)</td>
<td>Repeat a list of statements.</td>
</tr>
<tr>
<td>&quot;WAIT / WAIT_TIME&quot; (→ p. 71)</td>
<td>Suspends the execution of an ST program.</td>
</tr>
<tr>
<td>&quot;WHILE DO END_WHILE&quot; (→ p. 73)</td>
<td>Repeat a list of statements while a condition is TRUE.</td>
</tr>
</tbody>
</table>

4.1.3 Assignment :=

PLCopen ✔
Operator - Variable assignment.

### 4.1.3.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Any variable or complex expression.</td>
</tr>
</tbody>
</table>

### 4.1.3.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td></td>
<td></td>
<td>Forced variable.</td>
</tr>
</tbody>
</table>

### 4.1.3.3 Remarks

- The output variable and the input expression must have the same type.
- The forced variable cannot have the read-only attribute.
- In the FFLD and FBD languages, the 1 block is available to perform a 1 gain data copy (1 copy).
  - The FFLD instruction loads the first operand.
  - 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.

### 4.1.3.4 FBD Language

![FBD Diagram]

### 4.1.3.5 FFLD Language

- In the FFLD language, the input rung (EN) enables the assignment
- The output rung keeps the state of the input rung.

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

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

### 4.1.3.6 IL Language

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

[Image: FBD Diagram]

[Image: IL Diagram]
4.1.3.7 ST Language

```
Q := IN; (* copy IN into variable Q *)
    Q := (IN1 + (IN2 / IN3)) * 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 *)
```

See Also

Parenthesis ( )

4.1.4 Bit Access

You can directly specify a bit within an integer variable in expressions and diagrams using this 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 these 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 (zero) always represents the less significant bit.

4.1.5 Differences Between Functions and Function Blocks

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

- A **Function** is called once and it performs an action.
  - This is synchronous.

- A **Function Block** (FB) is an instance that has its own set of data.
  - An FB maintains its own, internal machine state and often has an output to indicate when the work is done.
  - An FB is 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 monitor the status of this action, especially via the Done output.

See Also

- Call a Function
- Call a Function Block

4.1.6 Call a Sub-Program

A sub-program is called by another program.
- Unlike function blocks, local variables of a sub-program are not instantiated and 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.

### 4.1.6.1 FBD and FFLD Languages
To call a sub-program in FBD or FFLD languages, insert the block in the diagram and connect its inputs and outputs.

### 4.1.6.2 IL Language
To call a sub-program in the 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 commas.

Alternatively, the CALC, CALCN or CALNC conditional instructions can be used:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAL</td>
<td>Calls the sub-program</td>
</tr>
<tr>
<td>CALC</td>
<td>Calls the sub-program if the current result is TRUE</td>
</tr>
<tr>
<td>CALNC</td>
<td>Calls the sub-program if the current result is FALSE</td>
</tr>
<tr>
<td>CALCN</td>
<td>Same as CALNC</td>
</tr>
</tbody>
</table>

#### Example

```
Op1: CAL MySubProg (i1, i2)
FFLD MySubProg.Q1
ST Res1
FFLD MySubProg.Q2
ST Res2
```

### 4.1.6.3 ST Language
To call a sub-program in ST, you must specify its name, followed by the input parameters written between parentheses and separated by commas.

To have access to an output parameter, use the name of the sub-program followed by a dot . and the name of the 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);
```

### 4.1.7 CASE OF ELSE END_CASE

**Statement** - Switch to one of various possible statements.

#### 4.1.7.1 Syntax
CASE <DINT expression> OF
  <value> :
    <statements>
  <value> , <value> :
    <statements>;
  <value> .. <value> :
    <statements>;
ELSE
  <statements>
END_CASE;

4.1.7.2 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.
  - a list of possible values separated by commas (,).
  - a range of values specified by a “min .. max” interval.
- You must enter space characters before and after the ".." separator.

4.1.7.3 FBD Language

Not available.

4.1.7.4 FFLD Language

Not available.

4.1.7.5 IL Language

Not available.

4.1.7.6 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;

See Also

- EXIT
- "FOR TO BY END_FOR" (p. 62)
4.1.8 CountOf

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

### 4.1.8.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARR</td>
<td>Any</td>
<td></td>
<td></td>
<td></td>
<td>Declared array.</td>
</tr>
</tbody>
</table>

### 4.1.8.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td>Total</td>
<td></td>
<td>number of items</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in</td>
<td></td>
<td>in the array.</td>
</tr>
</tbody>
</table>

### 4.1.8.3 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.
  - This keeps the program independent from the declaration.

### Example

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

### Examples

<table>
<thead>
<tr>
<th>Array</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arr1 [ 0..9 ]</td>
<td>10</td>
</tr>
<tr>
<td>Arr2 [ 0..4 , 0..9 ]</td>
<td>50</td>
</tr>
</tbody>
</table>

### 4.1.8.4 FBD Language

```
CountOf
Arr[]
```

### 4.1.8.5 FFLD Language

- In the 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.
  - The function is executed only if EN is TRUE.
4.1.8.6 IL Language

Not available.

4.1.8.7 ST Language

\[
Q := \text{CountOf} (\text{ARR});
\]

4.1.9 DEC

**Function** - Decrease a numerical variable.

4.1.9.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td>ANY</td>
<td></td>
<td></td>
<td>Numerical variable (increased after call).</td>
</tr>
</tbody>
</table>

4.1.9.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td>ANY</td>
<td></td>
<td>Decreased value.</td>
</tr>
</tbody>
</table>

4.1.9.3 Remarks

- This function is particularly designed for "ST Language" (⇒ p. 61).
  - It allows simplified writing as assigning the result of the function is not mandatory.
- 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, a variable is decreased by 1.0.
- For time values, a variable is decreased by 1 ms.
- The IN input must be directly connected to a variable.
  - It cannot be a constant or complex expression.

4.1.9.4 FBD Language

4.1.9.5 FFLD Language
4.1.9.6 IL Language
Not available.

4.1.9.7 ST Language

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

4.1.10 EXIT

**Statement** - Exit from a loop instruction.

4.1.10.1 Remarks
- The EXIT statement indicates that the current loop (FOR, REPEAT, or WHILE) must be finished.
- The execution continues after the END_FOR, END_REPEAT, or END_WHILE 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.

4.1.10.2 FBD Language
Not available.

4.1.10.3 FFLD Language
Not available.

4.1.10.4 IL Language
Not available.

4.1.10.5 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;
```

See Also
4.1.11 FOR TO BY END_FOF

Statement - Execute iterations of statements.

4.1.11.1 Syntax

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

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).

4.1.11.2 Remarks

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

4.1.11.3 FBD Language

Not available.

4.1.11.4 FFLD Language

Not available.

4.1.11.5 IL Language

Not available.

4.1.11.6 ST Language

iArrayDim := 10;

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

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

See Also
4.1.12 IF THEN ELSE ELSIF END_IF

Statement - Conditional execution of statements.

4.1.12.1 Syntax

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

4.1.12.2 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.

4.1.12.3 FBD Language

Not available.

4.1.12.4 FFLD Language

Not available.

4.1.12.5 IL Language

Not available.

4.1.12.6 ST Language

```plaintext
(* 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;
```
EN_D_IF;

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

See Also
- "CASE OF ELSE END_CASE" (→ p. 57)
- "EXIT" (→ p. 61)
- "FOR TO BY END_FOR" (→ p. 62)
- "REPEAT UNTIL END_REPEAT" (→ p. 69)
- "WHILE DO END_WHILE" (→ p. 73)

4.1.13 INC

Function - Increase a numerical variable.

4.1.13.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td>Numerical variable (increased after call).</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.1.13.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td>Increased value.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.1.13.3 Remarks

- This function is particularly designed for "ST Language" (→ p. 65).
  - It allows simplified writing as assigning the result of the function is not mandatory.
- 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, a variable is decreased by 1.0.
- For time values, a variable is decreased by 1 ms.
  - The IN input must be directly connected to a variable.
  - It cannot be a constant or complex expression.

4.1.13.4 FBD Language
4.1.13.5 FFLD Language

![FFLD Language Diagram]

4.1.13.6 IL Language

Not available.

4.1.13.7 ST Language

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

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

4.1.14 MoveBlock

![PLCopen Check]

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

4.1.14.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DST</td>
<td>ANY (*)</td>
<td></td>
<td></td>
<td></td>
<td>Array containing the destination of the copy.</td>
</tr>
<tr>
<td>NB</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Number of items to be copied.</td>
</tr>
<tr>
<td>PosDST</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Index of the destination in DST.</td>
</tr>
<tr>
<td>PosSRC</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Index of the first character in SRC.</td>
</tr>
<tr>
<td>SRC</td>
<td>ANY (*)</td>
<td></td>
<td></td>
<td></td>
<td>Array containing the source of the copy.</td>
</tr>
</tbody>
</table>

(*) SRC and DST cannot be a STRING.

4.1.14.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE if successful.</td>
</tr>
</tbody>
</table>

4.1.14.3 Remarks

- Arrays of string are not supported by this function.
- 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 this case, the function avoids lost items when source and destination areas overlap.

• This function verifies 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.

### 4.1.14.4 FBD Language

![FBD Diagram]

### 4.1.14.5 FFLD Language

• In the FFLD language, the operation is executed only if the input rung (EN) is TRUE.

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

![FFLD Diagram]

### 4.1.14.6 IL Language

Not available.

### 4.1.14.7 ST Language

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

### 4.1.15 NEG -

**PLCopen** ✓

*Operator* - Performs a negation of the input. (unary operator)

#### 4.1.15.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Numeric value.</td>
</tr>
</tbody>
</table>

#### 4.1.15.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td></td>
<td></td>
<td>Negation of the input.</td>
</tr>
</tbody>
</table>
4.1.15.3 Truth Table

<table>
<thead>
<tr>
<th>IN</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>-123</td>
<td>123</td>
</tr>
</tbody>
</table>

4.1.15.4 Remarks

- In FBD and FFLD language, the block **NEG** can be used.

4.1.15.5 FBD Language

4.1.15.6 FFLD Language

- In the 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.

- The negation is executed only if EN is TRUE.

- ENO keeps the same value as EN.

4.1.15.7 IL Language

Not available.

4.1.15.8 ST Language

- In the ST language, "-" can be followed by a complex Boolean expression between parentheses.
  - The output data type must be the same as the input data type.

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

4.1.16 ON

**Statement** - Conditional execution of statements.

4.1.16.1 Syntax

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

- The `ON` instruction provides a simpler syntax for checking the rising edge of a Boolean condition.
- 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 or sub-program.
- This statement is an extension to the standard and is NOT IEC 61131-3 compliant.

⚠️ CAUTION
This instruction is NOT UDFB safe. Do not use inside UDFBs.

4.1.16.3 FBD Language

Not available.

4.1.16.4 FFLD Language

Not available.

4.1.16.5 IL Language

Not available.

4.1.16.6 ST Language

```st
(* This example counts the rising edges of variable bIN *)
ON bIN DO
   diCount := diCount + 1;
END_DO;
```

4.1.17 Parenthesis ( )

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

4.1.17.1 Remarks

- Parentheses are used in ST and IL languages for changing the default evaluation order of various operations within a complex expression.
- Example: The default evaluation of `2 * 3 + 4` expression in ST language gives a result of 10 because `*` operator has the highest priority.
  - Changing the expression as `2 * (3 + 4)` gives a result of 14.
- Parentheses can be nested in a complex expression.

These are the default evaluation priority order for ST language operations:

<table>
<thead>
<tr>
<th>Priority</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>- NOT</td>
<td>Unary operators</td>
</tr>
<tr>
<td>2</td>
<td>* /</td>
<td>Multiply / Divide</td>
</tr>
<tr>
<td>3</td>
<td>+ -</td>
<td>Add / Subtract</td>
</tr>
<tr>
<td>Priority</td>
<td>Operation</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>4</td>
<td>&lt;&gt; &lt;= &gt;= &lt;&gt;</td>
<td>Comparisons</td>
</tr>
<tr>
<td>5</td>
<td>&amp; AND</td>
<td>Boolean And</td>
</tr>
<tr>
<td>6</td>
<td>OR</td>
<td>Boolean Or</td>
</tr>
<tr>
<td>7</td>
<td>XOR</td>
<td>Exclusive OR</td>
</tr>
</tbody>
</table>

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

4.1.17.2 FBD Language
Not available.

4.1.17.3 FFLD Language
Not available.

4.1.17.4 IL Language

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

4.1.17.5 ST Language

```
Q := (IN1 + (IN2 / IN3)) * IN4;
```

See Also
Assignment :=

4.1.18 REPEAT UNTIL END_REPEAT
Statement - Repeat a list of statements.

4.1.18.1 Syntax

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

4.1.18.2 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 because the input is not refreshed before the next cycle.

### 4.1.18.3 FBD Language

Not available.

### 4.1.18.4 FFLD Language

Not available.

### 4.1.18.5 IL Language

Not available.

### 4.1.18.6 ST Language

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

**See Also**
- "**CASE OF ELSE END_CASE**" (→ p. 57)
- "**EXIT**" (→ p. 61)
- "**FOR TO BY END_FOR**" (→ p. 62)
- "**IF THEN ELSE ELSEIF END_IF**" (→ p. 63)
- "**WHILE DO END_WHILE**" (→ p. 73)

### 4.1.19 RETURN RET RETC RETNC RETCN

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

#### 4.1.19.1 Remarks

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

#### 4.1.19.2 FBD Language

- In the 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 this example, the TON block will not be called if bIgnore is TRUE.*)
4.1.19.3 FFLD Language

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

4.1.19.4 IL Language

These are the meanings 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

4.1.19.5 ST Language

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

See Also

- Jumps JMP JMPC JMPNC JMPCN
- LABELS

4.1.20 WAIT / WAIT_TIME

Statement - Suspends the execution of an ST program.

4.1.20.1 Syntax
4.1.20.2 Remarks

- The `WAIT` statement:
  - Provides an easy way to program a state machine.
    - This avoids the use of complex `CASE` structures.
  - Verifies the attached Boolean expression and takes these actions:
    - 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 have no correspondence in other languages.

- The `WAIT` and `WAIT_TIME` 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.
  - 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.
  - Do not support re-entrancy.
    - 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.
  - 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_TIME` in SFC or in a sub-program called from SFC provokes a compile error.
  - 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.

- This statement is an extension to the standard and is NOT IEC 61131-3 compliant.

⚠️ CAUTION

This instruction is NOT UDFB safe.
Do not use inside UDFBs.

4.1.20.3 FBD Language

Not available.

4.1.20.4 FFLD Language

Not available.
4.1.20.5 IL Language
Not available.

4.1.20.6 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;

4.1.21 WHILE DO END_WHILE

Statement - Repeat a list of statements while a condition is TRUE.

4.1.21.1 Syntax

WHILE <BOOL expression> DO
    <statements>
END_WHILE;

4.1.21.2 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 evaluated 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 because the input is not refreshed before the next cycle.

4.1.21.3 FBD Language
Not available.

4.1.21.4 FFLD Language
Not available.

4.1.21.5 IL Language
Not available.

4.1.21.6 ST Language

iMax := 10;
WHILE iPos < iMax DO
    MyArray[iPos]:=0;
See Also

- "CASE OF ELSE END_CASE" (→ p. 57)
- "EXIT" (→ p. 61)
- "FOR TO BY END_FOR" (→ p. 62)
- "IF THEN ELSE ELSIF END_IF" (→ p. 63)
- "REPEAT UNTIL END_REPEAT" (→ p. 69)

4.2 Boolean Operations

4.2.1 Standard Operators

These are the standard operators for managing Booleans.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND ANDN &amp;</td>
<td>Performs a logical AND of all inputs.</td>
</tr>
<tr>
<td>NOT</td>
<td>Performs a Boolean negation of the input.</td>
</tr>
<tr>
<td>OR / ORN</td>
<td>Performs a logical OR of all inputs.</td>
</tr>
<tr>
<td>&quot;QOR&quot; (→ p. 78)</td>
<td>Counts the number of TRUE inputs.</td>
</tr>
<tr>
<td>R</td>
<td>Force a Boolean output to FALSE.</td>
</tr>
<tr>
<td>S</td>
<td>Force a Boolean output to TRUE.</td>
</tr>
<tr>
<td>XOR / XORN</td>
<td>Performs an exclusive OR of all inputs.</td>
</tr>
</tbody>
</table>

4.2.2 Available Blocks

These are the available blocks for managing Boolean signals:

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_trig</td>
<td>Falling pulse detection.</td>
</tr>
<tr>
<td>FlipFop</td>
<td>Flipflop bistable.</td>
</tr>
<tr>
<td>r_trig</td>
<td>Rising pulse detection.</td>
</tr>
<tr>
<td>RS</td>
<td>Reset dominant bistable.</td>
</tr>
<tr>
<td>sema</td>
<td>Semaphore.</td>
</tr>
<tr>
<td>SR</td>
<td>Set dominant bistable.</td>
</tr>
</tbody>
</table>

4.2.3 FlipFlop

Function Block - Flipflop bistable.
4.2.3.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Swap command (on rising edge).</td>
</tr>
<tr>
<td>RST</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Reset to FALSE.</td>
</tr>
</tbody>
</table>

4.2.3.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Output.</td>
</tr>
</tbody>
</table>

4.2.3.3 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.

4.2.3.4 FBD Language

![FBD Diagram]

4.2.3.5 FFLD Language

![FFLD Diagram]

4.2.3.6 IL Language

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

4.2.3.7 ST Language

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

See Also

- R
- S
- SR
4.2.4 f_trig

**Function Block** - Falling pulse detection.

### 4.2.4.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Boolean signal.</td>
</tr>
</tbody>
</table>

### 4.2.4.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td>TRUE</td>
<td></td>
<td>TRUE when the input changes from TRUE to FALSE.</td>
</tr>
</tbody>
</table>

### 4.2.4.3 Truth Table

<table>
<thead>
<tr>
<th>CLK</th>
<th>CCLK (prev)</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### 4.2.4.4 Remarks

- It is recommended to use declared instances of R_TRIG or F_TRIG function blocks.
- This is to avoid contingencies during an Online Change.

### 4.2.4.5 FBD Language

```

```

### 4.2.4.6 FFLD Language

- In the FFLD language, ]P[ and ]N[ contacts can be used.

```

```

### 4.2.4.7 IL Language

```il
(* MyTrigger is declared as an instance of F_TRIG function block *)
Op1: CAL MyTrigger (CLK)
```
4.2.4.8 ST Language

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

See Also
r_trig

4.2.5 NOT

Operator - Performs a Boolean negation of the input.

4.2.5.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Boolean value.</td>
</tr>
</tbody>
</table>

4.2.5.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Boolean negation of the input.</td>
</tr>
</tbody>
</table>

4.2.5.3 Truth Table

<table>
<thead>
<tr>
<th>IN</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

4.2.5.4 Remarks

4.2.5.5 FBD Language

- In the FBD language, the block "NOT" can be used.
  - Alternatively, you can use a link terminated by a "o" negation.

Example: Explicit use of the NOT block:

Example: Use of a negated link: Q is IN1 AND NOT IN2:

4.2.5.6 FFLD Language
In the FFLD language, negated contacts and coils can be used.

Example: Negated contact: Q is IN1 AND NOT IN2:

\[ Q := \overline{IN1} \land \overline{IN2} \]

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

\[ Q := \overline{(IN1 \land IN2)} \]

### 4.2.5.7 IL Language

- In the IL language, the \( \text{N} \) modifier can be used with instructions FFLD, AND, OR, XOR and ST.
  - It represents a negation of the operand.

```
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) *)
```

### 4.2.5.8 ST Language

- In the ST language, NOT can be followed by a complex Boolean expression between parentheses.

```
Q := \overline{IN};
Q := \overline{(IN1 OR IN2)};
```

### See Also

- \( \text{AND} \) \( \text{ANDN} \) \&
- \( \text{OR} \) \( \text{ORN} \)
- \( \text{XOR} \) \( \text{XORN} \)

### 4.2.6 QOR

**Operator** - Counts the number of TRUE inputs.

#### 4.2.6.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Boolean inputs.</td>
</tr>
</tbody>
</table>

#### 4.2.6.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td></td>
<td></td>
<td>Number of inputs being TRUE.</td>
</tr>
</tbody>
</table>

#### 4.2.6.3 Remarks
The block accepts a non-fixed number of inputs.

4.2.6.4 FBD Language
- The block can have a maximum of 16 inputs.

4.2.6.5 FFLD Language
- The block can have a maximum of 16 inputs.

4.2.6.6 IL Language

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

4.2.6.7 ST Language

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

4.2.7 R
Operator - Force a Boolean output to FALSE.

4.2.7.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Condition.</td>
</tr>
</tbody>
</table>

4.2.7.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Output to be forced.</td>
</tr>
</tbody>
</table>

4.2.7.3 Truth Table

<table>
<thead>
<tr>
<th>RESET</th>
<th>Q (prev)</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
4.2.7.4 Remarks

None

4.2.7.5 FBD Language

- In the FBD language, RS and SR function blocks are preferred.
  - Use "RS" (→ p. 80) or "SR" (→ p. 86) function blocks.
  - (S) and (R) coils can be used.

4.2.7.6 FFLD Language

- In the FFLD languages, they are represented by (S) and (R) coils.

Example: Use of R coil:

\[
\text{RESET} \quad \overline{Q} \quad \overline{Q} \quad \overline{Q}
\]

4.2.7.7 IL Language

- In the IL language, S and R operators are available as standard instructions.

<table>
<thead>
<tr>
<th>Op1: FFLD RESET</th>
</tr>
</thead>
<tbody>
<tr>
<td>R Q</td>
</tr>
<tr>
<td>(* Q is forced to FALSE if RESET is TRUE *)</td>
</tr>
<tr>
<td>(* Q is unchanged if RESET is FALSE *)</td>
</tr>
</tbody>
</table>

4.2.7.8 ST Language

Not available.

See Also

- "RS" (→ p. 80)
- "S" (→ p. 83)
- "SR" (→ p. 86)

4.2.8 RS

Function Block - Reset dominant bistable.

4.2.8.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Condition for forcing to TRUE.</td>
</tr>
<tr>
<td>RESET1</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Condition for forcing to FALSE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Highest priority command.</td>
</tr>
</tbody>
</table>

4.2.8.2 Outputs
4.2.8.3 Truth Table

<table>
<thead>
<tr>
<th>SET</th>
<th>RESET1</th>
<th>Q1 prev</th>
<th>Q1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

4.2.8.4 Remarks

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

4.2.8.5 FBD Language

```
SET
RESET1

Q1
```

4.2.8.6 FFLD Language

```
SET
RESET1

Q1
```

4.2.8.7 IL Language

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

4.2.8.8 ST Language

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

See Also
4.2.9 \texttt{r\_trig}

\begin{itemize}
  \item "R" (\(\rightarrow\) p. 79)
  \item "RS" (\(\rightarrow\) p. 80)
  \item "SR" (\(\rightarrow\) p. 86)
\end{itemize}

\textbf{Function Block} - Rising pulse detection.

### 4.2.9.1 Inputs

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|}
\hline
\textbf{Input} & \textbf{Data Type} & \textbf{Range} & \textbf{Unit} & \textbf{Default} & \textbf{Description} \\
\hline
CLK & BOOL & & & & Boolean signal. \\
\hline
\end{tabular}
\end{table}

### 4.2.9.2 Outputs

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|}
\hline
\textbf{Output} & \textbf{Data Type} & \textbf{Range} & \textbf{Unit} & \textbf{Description} \\
\hline
Q & BOOL & & & TRUE when the input changes from FALSE to TRUE. \\
\hline
\end{tabular}
\end{table}

### 4.2.9.3 Truth Table

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
CLK & CCLK (prev) & Q \\
\hline
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\hline
\end{tabular}
\end{table}

### 4.2.9.4 Remarks

- It is recommended to use declared instances of \texttt{R\_TRIG} or \texttt{F\_TRIG} function blocks.
- This is to avoid contingencies during an Online Change.

### 4.2.9.5 FBD Language

\begin{figure}
\centering
\includegraphics[width=0.5\textwidth]{fig1.png}
\end{figure}

### 4.2.9.6 FFLD Language

- In the FFLD language, \texttt{P[} and \texttt{N[} contacts can be used.
- The input signal is the rung.
- The rung is the output.

\begin{figure}
\centering
\includegraphics[width=0.5\textwidth]{fig2.png}
\end{figure}

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

4.2.9.8 ST Language

See Also
- "f_trig" (→ p. 76)

4.2.10 S

Operator - Force a Boolean output to TRUE.

4.2.10.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Condition.</td>
</tr>
</tbody>
</table>

4.2.10.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Output to be forced.</td>
</tr>
</tbody>
</table>

4.2.10.3 Truth Table

<table>
<thead>
<tr>
<th>SET</th>
<th>Q prev</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

4.2.10.4 Remarks

None

4.2.10.5 FBD Language

- In the FBD language, RS and SR function blocks are preferred.
  - Use "RS" (→ p. 80) or "SR" (→ p. 86) function blocks.
  - (S) and (R) coils can be used.

4.2.10.6 FFLD Language

- In the FFLD languages, they are represented by (S) and (R) coils.
Example: Use of S coil:

\[
\begin{array}{c}
\text{SET} \\
\hline
\end{array}
\]

\[
\begin{array}{c}
\text{Op1: FFLD SET} \\
\text{S Q} \\
\hline
\text{(* Q is forced to TRUE if SET is TRUE *)} \\
\text{(* Q is unchanged if SET is FALSE *)}
\end{array}
\]

### 4.2.10.7 IL Language

- In the IL language, S and R operators are available as standard instructions.

### 4.2.10.8 ST Language

Not available.

**See Also**

- "R" (p. 79)
- "RS" (p. 80)
- "SR" (p. 86)
4.2.11 sema

**Function Block** - Semaphore.

### 4.2.11.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLAIM</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Takes the semaphore.</td>
</tr>
<tr>
<td>RELEASE</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Releases the semaphore.</td>
</tr>
</tbody>
</table>

### 4.2.11.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSY</td>
<td>BOOL</td>
<td>TRUE</td>
<td></td>
<td>TRUE if semaphore is busy.</td>
</tr>
</tbody>
</table>

### 4.2.11.3 Remarks

The function block implements this algorithm:

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

### 4.2.11.4 FBD Language

![FBD Diagram]

### 4.2.11.5 FFLD Language

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

```
CLAIM | SEMA
RELEASE | BUSY
```

### 4.2.11.6 IL Language

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

### 4.2.11.7 ST Language
4.2.12 SR

![PLCopen](image)

**Function Block** - Set dominant bistable.

### 4.2.12.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET1</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Condition for forcing to TRUE. Highest priority command.</td>
</tr>
<tr>
<td>RESET</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Condition for forcing to FALSE.</td>
</tr>
</tbody>
</table>

### 4.2.12.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Output to be forced.</td>
</tr>
</tbody>
</table>

### 4.2.12.3 Truth Table

<table>
<thead>
<tr>
<th>SET1</th>
<th>RESET</th>
<th>Q1 prev</th>
<th>Q1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### 4.2.12.4 Remarks

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

### 4.2.12.5 FBD Language

[Diagram of FBD language]
4.2.12.6 FFLD Language

- The SET1 command is the rung.
- The rung is the output.

\[
\begin{array}{c|c|c}
\text{SET1} & \text{SR} & \text{Q1} \\
\text{RESET} & & \\
\end{array}
\]

4.2.12.7 IL Language

\[
\begin{array}{l}
\text{Op1: \textit{CAL} MySR (SET1, RESET)} \\
\text{FFLD MySR.Q1} \\
\text{ST Q1}
\end{array}
\]

4.2.12.8 ST Language

\[
\begin{array}{l}
\text{MySR (SET1, RESET);} \\
\text{Q1 := MySR.Q1;}
\end{array}
\]

See Also

- "R" (→ p. 79)
- "RS" (→ p. 80)
- "S" (→ p. 83)

4.2.13 XOR / XORN

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

4.2.13.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>First Boolean input.</td>
</tr>
<tr>
<td>IN2</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Second Boolean input.</td>
</tr>
</tbody>
</table>

4.2.13.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Exclusive OR of all inputs.</td>
</tr>
</tbody>
</table>

4.2.13.3 Truth Table

<table>
<thead>
<tr>
<th>IN1</th>
<th>IN2</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
4.2.13.4 Remarks
- The block is called =1 in FBD and FFLD languages.

4.2.13.5 FBD Language

4.2.13.6 FFLD Language
- The first input is the rung.
- The rung is the output.

4.2.13.7 IL Language
- In the 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.

4.2.13.8 ST Language

See Also
- AND ANDN &
- "NOT" (➔ p. 77)
- OR / ORN
4.3 Arithmetic Operations

4.3.1 Standard Operators
These are the standard operators that perform arithmetic operations:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Addition +&quot; (→ p. 89)</td>
<td>Performs an addition of all inputs.</td>
</tr>
<tr>
<td>&quot;Divide /&quot; (→ p. 90)</td>
<td>Performs a division of all inputs.</td>
</tr>
<tr>
<td>&quot;Multiply *&quot; (→ p. 99)</td>
<td>Performs a multiplication of all inputs.</td>
</tr>
<tr>
<td>NEG -</td>
<td>Performs a negation of the input. (unary operator)</td>
</tr>
<tr>
<td>&quot;Subtraction -&quot; (→ p. 102)</td>
<td>Performs a subtraction of all inputs.</td>
</tr>
</tbody>
</table>

4.3.2 Standard Functions
These are the standard functions that perform arithmetic operations:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>limit</td>
<td>Limits a numeric value between low and high bounds.</td>
</tr>
<tr>
<td>max</td>
<td>Get the maximum of two integers.</td>
</tr>
<tr>
<td>min</td>
<td>Get the minimum of two integers.</td>
</tr>
<tr>
<td>mod /</td>
<td>Calculation of modulo.</td>
</tr>
<tr>
<td>modLR /</td>
<td></td>
</tr>
<tr>
<td>modR</td>
<td></td>
</tr>
<tr>
<td>odd</td>
<td>Test if an integer is odd.</td>
</tr>
<tr>
<td>&quot;SetWithin&quot; (→ p. 101)</td>
<td>Force a value when inside an interval.</td>
</tr>
</tbody>
</table>

4.3.3 Addition +
Operator - Performs an addition of all inputs.

4.3.3.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>First input.</td>
</tr>
<tr>
<td>IN2</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Second input.</td>
</tr>
</tbody>
</table>

4.3.3.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td></td>
<td></td>
<td>Result: IN1 + IN2.</td>
</tr>
</tbody>
</table>
4.3.3.3 Remarks
- All inputs and the output must have the same type.
- The addition can be used with strings.
  - The result is the concatenation of the input strings.

4.3.3.4 FBD Language
- In the FBD language, the block can have a maximum of 32 inputs.

4.3.3.5 FFLD Language
- In the FFLD language, the input rung (EN) enables the operation.
  - The output rung (ENO) keeps the same value as the input rung.

4.3.3.6 IL Language
- In the 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.

```
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 *)
```

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

See Also
- Divide /
- Multiply *
- Subtraction -

4.3.4 Divide /
Operator - Performs a division of all inputs.
4.3.4.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>ANY_NUM</td>
<td>First</td>
<td></td>
<td></td>
<td>First input.</td>
</tr>
<tr>
<td>IN2</td>
<td>ANY_NUM</td>
<td>Second</td>
<td></td>
<td></td>
<td>Second input.</td>
</tr>
</tbody>
</table>

4.3.4.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY_NUM</td>
<td></td>
<td></td>
<td>Result: IN1 / IN2</td>
</tr>
</tbody>
</table>

4.3.4.3 Remarks

- All inputs and the output must have the same type.

4.3.4.4 FBD Language

- In the FBD language, the block can have a maximum of 32 inputs.

```
IN1
IN2
/
Q
```

4.3.4.5 FFLD Language

- In the FFLD language, the input rung (EN) enables the operation.
  - The output rung (ENO) keeps the same value as the input rung.

```
[EN]
IN1
IN2
/
Q
[ENO]
```

4.3.4.6 IL Language

- In the 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.

```
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 *)
```

4.3.4.7 ST Language

```
Q := IN1 / IN2;
```
See Also

- Addition +
- Multiply *
- Subtraction -

4.3.5 NEG -

Operator - Performs a negation of the input. (unary operator)

4.3.5.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Numeric value.</td>
</tr>
</tbody>
</table>

4.3.5.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td></td>
<td></td>
<td>Negation of the input.</td>
</tr>
</tbody>
</table>

4.3.5.3 Truth Table

<table>
<thead>
<tr>
<th>IN</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>-123</td>
<td>123</td>
</tr>
</tbody>
</table>

4.3.5.4 Remarks

- In FBD and FFLD language, the block NEG can be used.

4.3.5.5 FBD Language

4.3.5.6 FFLD Language

- In the 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.

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

4.3.5.7 IL Language
4.3.5.8 ST Language

- In the ST language, "-" can be followed by a complex Boolean expression between parentheses.
- The output data type must be the same as the input data type.

\[
Q := - \text{IN}; \\
Q := - (\text{IN1} + \text{IN2});
\]

4.3.6 limit

**Function** - Limits a numeric value between low and high bounds.

### 4.3.6.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMIN</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Low bound.</td>
</tr>
<tr>
<td>IN</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Input value.</td>
</tr>
<tr>
<td>IMAX</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>High bound.</td>
</tr>
</tbody>
</table>

### 4.3.6.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td>IMIN</td>
<td></td>
<td>IMIN if IN &lt; IMIN; IMAX if IN &gt; IMAX; IN otherwise.</td>
</tr>
</tbody>
</table>

### 4.3.6.3 Remarks

4.3.6.3.1 Function Diagram

![Function Diagram](image)

### 4.3.6.4 FBD Language

![FBD Language Diagram](image)
4.3.6.5 FFLD Language

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

```
  EN [ ] LIMIT [ ]
  IMIN [ ] IMAX [ ]
  IN [ ]
  Q [ ]
  ENO ( )
```

4.3.6.6 IL Language

- In the IL language, the first input must be loaded before the function call.
  - Other inputs are operands of the function, separated by a coma.

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

4.3.6.7 ST Language

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

See Also

- max
- min
- mod / modLR / modR
- odd

4.3.7 max

**PLCopen**

Function - Get the maximum of two integers.

4.3.7.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>First input.</td>
</tr>
<tr>
<td>IN2</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Second input.</td>
</tr>
</tbody>
</table>

4.3.7.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td></td>
<td></td>
<td>IN1 if IN1 &gt; IN2; IN2 otherwise.</td>
</tr>
</tbody>
</table>

4.3.7.3 Remarks
None

4.3.7.4 FBD Language

\[\text{MAX} \quad \text{IN1} \rightarrow \text{Q} \quad \text{IN2} \]

4.3.7.5 FFLD Language

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

\[\text{EN} \quad \text{MAX} \quad \text{ENO} \quad \text{IN1} \rightarrow \text{Q} \quad \text{IN2} \]

4.3.7.6 IL Language

- In the IL language, the first input must be loaded before the function call.
  - The second input is the operand of the function.

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

4.3.7.7 ST Language

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

See Also

- "limit" (➔ p. 93)
- "min" (➔ p. 95)
- "mod / modLR / modR" (➔ p. 96)
- "odd" (➔ p. 100)

4.3.8 min

```python
PLCopen ✓
```

**Function** - Get the minimum of two integers.

4.3.8.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>First input.</td>
</tr>
<tr>
<td>IN2</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Second input.</td>
</tr>
</tbody>
</table>
4.3.8.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td>IN1 if IN1 &lt; IN2; IN2 otherwise.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.3.8.3 Remarks
None

4.3.8.4 FBD Language

```
MIN
IN1
IN2
```

4.3.8.5 FFLD Language
- In the FFLD language, the input rung (EN) enables the operation.
  - The output rung keeps the state of the input rung.
  - The comparison is executed only if EN is TRUE.
  - ENO has the same value as EN.

```
EN
IN1
IN2
MIN
ENO
```

4.3.8.6 IL Language
- In the IL language, the first input must be loaded before the function call.
  - The second input is the operand of the function.

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

4.3.8.7 ST Language

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

See Also
- "limit" (→ p. 93)
- "max" (→ p. 94)
- "mod / modLR / modR" (→ p. 96)
- "odd" (→ p. 100)

4.3.9 mod / modLR / modR

[PLCopen] [✓]
**Function** - Calculation of modulo.
The modulo is calculated as:

\[ Q = \text{IN} - \text{Trunc}(\text{IN}/\text{BASE}) \times \text{BASE} \]

where \( \text{Trunc}(x) \) calculates the truncated (rounded toward zero) value of \( x \).

**Note**
If \( \text{BASE} = 0 \), then \( Q \) will return 0.
If \( \text{IN} = 0 \), then \( Q \) will return 0.

### 4.3.9.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>mod = DINT</td>
<td></td>
<td></td>
<td></td>
<td>Input value.</td>
</tr>
<tr>
<td></td>
<td>modR = REAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>modLR = LREAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BASE</td>
<td>mod = DINT</td>
<td></td>
<td></td>
<td></td>
<td>Base of the modulo.</td>
</tr>
<tr>
<td></td>
<td>modR = REAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>modLR = LREAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.3.9.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>mod = DINT</td>
<td></td>
<td></td>
<td>Modulo: rest of the integer division (IN / BASE).</td>
</tr>
<tr>
<td></td>
<td>modR = REAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>modLR = LREAL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.3.9.3 Examples

**MOD Examples**

<table>
<thead>
<tr>
<th>IN</th>
<th>BASE</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>-5</td>
<td>7</td>
<td>-5</td>
</tr>
<tr>
<td>-7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>-11</td>
<td>7</td>
<td>-4</td>
</tr>
<tr>
<td>11</td>
<td>-7</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>-7</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>-7</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>-7</td>
<td>0</td>
</tr>
<tr>
<td>-5</td>
<td>-7</td>
<td>-5</td>
</tr>
</tbody>
</table>
### MODR / MODLR Examples

<table>
<thead>
<tr>
<th>IN</th>
<th>BASE</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>-7</td>
<td>-7</td>
<td>0</td>
</tr>
<tr>
<td>-11</td>
<td>-7</td>
<td>-4</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### 4.3.9.4 Remarks

None

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

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

### 4.3.9.5 FBD Language

```
MOD
IN
BASE
Q
```

### 4.3.9.6 FFLD Language
• In the FFLD language, the input rung (EN) enables the operation.
  • The output rung keeps the state of the input rung.
  • The comparison is executed only if EN is TRUE.
  • ENO has the same value as EN.

4.3.9.7 IL Language
• In the IL language, the first input must be loaded before the function call.
  • The second input is the operand of the function.

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

4.3.9.8 ST Language
```
Q := MOD (IN, BASE);
```

See Also
• "limit" (➔ p. 93)
• "max" (➔ p. 94)
• "min" (➔ p. 95)
• "odd" (➔ p. 100)

4.3.10 Multiply *
Operator - Performs a multiplication of all inputs.

4.3.10.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>ANY_NUM</td>
<td></td>
<td></td>
<td></td>
<td>First input.</td>
</tr>
<tr>
<td>IN2</td>
<td>ANY_NUM</td>
<td></td>
<td></td>
<td></td>
<td>Second input.</td>
</tr>
</tbody>
</table>

4.3.10.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY_NUM</td>
<td></td>
<td></td>
<td>Result: IN1 * IN2.</td>
</tr>
</tbody>
</table>

4.3.10.3 Remarks
• All inputs and the output must have the same type.

4.3.10.4 FBD Language
• In the FBD language, the block can have a maximum of 32 inputs.

4.3.10.5 FFLD Language
• The multiplication is executed only if EN is TRUE.
• In the FFLD language, the input rung (EN) enables the operation.
  • The output rung (ENO) keeps the same value as the input rung.
• ENO is equal to EN.

4.3.10.6 IL Language
• In the 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.

    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 *)

4.3.10.7 ST Language

    Q := IN1 * IN2;

See Also
• Addition +
• Divide /
• Subtraction -

4.3.11 odd

   PLCopen ✓

Function - Test if an integer is odd.

4.3.11.1 Inputs
### 4.3.11.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE</td>
<td>TRUE if IN is odd.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>FALSE</td>
<td>FALSE if IN is even.</td>
</tr>
</tbody>
</table>

### 4.3.11.3 Remarks

None

### 4.3.11.4 FBD Language

```
IN   ODD   Q
```

### 4.3.11.5 FFLD Language

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

```
EN  ODD   Q
```

### 4.3.11.6 IL Language

- In the IL language, the first input must be loaded before the function call.
- The second input is the operand of the function.

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

### 4.3.11.7 ST Language

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

### See Also

- "limit" (→ p. 93)
- "max" (→ p. 94)
- min
- "mod / modLR / modR" (→ p. 96)

### 4.3.12 SetWithin
Function - Force a value when inside an interval.

### 4.3.12.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Input.</td>
</tr>
<tr>
<td>MIN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Low limit of the interval.</td>
</tr>
<tr>
<td>MAX</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>High limit of the interval.</td>
</tr>
<tr>
<td>VAL</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Value to apply when inside the interval.</td>
</tr>
</tbody>
</table>

### 4.3.12.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td></td>
<td></td>
<td>Result.</td>
</tr>
</tbody>
</table>

### 4.3.12.3 Truth Table

<table>
<thead>
<tr>
<th>In</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN &lt; MIN</td>
<td>IN</td>
</tr>
<tr>
<td>IN &gt; MAX</td>
<td>IN</td>
</tr>
<tr>
<td>MIN &lt; IN &lt; MAX</td>
<td>VAL</td>
</tr>
</tbody>
</table>

### 4.3.12.4 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.

### 4.3.13 Subtraction - Operator - Performs a subtraction of all inputs.

### 4.3.13.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>ANY_NUM / TIME</td>
<td></td>
<td></td>
<td></td>
<td>First input.</td>
</tr>
<tr>
<td>IN2</td>
<td>ANY_NUM / TIME</td>
<td></td>
<td></td>
<td></td>
<td>Second input.</td>
</tr>
</tbody>
</table>

### 4.3.13.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY_NUM / TIME</td>
<td></td>
<td></td>
<td>Result: IN1 - IN2.</td>
</tr>
</tbody>
</table>

### 4.3.13.3 Remarks

- All inputs and the output must have the same type.

### 4.3.13.4 FBD Language
4.3.13.5 **FFLD Language**
- In the FFLD language, the input rung (EN) enables the operation.
  - The output rung (ENO) keeps the same value as the input rung.
  - The subtraction is executed only if EN is TRUE.
  - ENO is equal to EN.

![Diagram of FFLD operation](image)

4.3.13.6 **IL Language**
- In the 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.

```plaintext
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 *)
```

4.3.13.7 **ST Language**

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

**See Also**
- Addition +
- Divide /
- Multiply *

4.4 **Comparison Operations**

![PLCopen](image)

These are the standard operators and blocks that perform comparisons:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LT &lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>LE &lt;=</td>
<td>Less than or equal to</td>
</tr>
</tbody>
</table>
### 4.4.1 CMP

**PLCopen**

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

#### 4.4.1.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>First value.</td>
</tr>
<tr>
<td>IN2</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Second value.</td>
</tr>
</tbody>
</table>

#### 4.4.1.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>BOOL</td>
<td>TRUE</td>
<td></td>
<td>TRUE if IN1 = IN2.</td>
</tr>
<tr>
<td>GT</td>
<td>BOOL</td>
<td>TRUE</td>
<td></td>
<td>TRUE if IN1 &gt; IN2.</td>
</tr>
<tr>
<td>LT</td>
<td>BOOL</td>
<td>TRUE</td>
<td></td>
<td>TRUE if IN1 &lt; IN2.</td>
</tr>
</tbody>
</table>

#### 4.4.1.3 Remarks

None

#### 4.4.1.4 FBD Language

![FBD Diagram]

#### 4.4.1.5 FFLD Language

- In the FFLD language, the rung input (EN) validates the operation.
- The rung output is the result of LT (lower than) comparison.
- The comparison is executed only if EN is TRUE.

![FFLD Diagram]

#### 4.4.1.6 IL Language
4.4.1.7  ST Language

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

See Also

- EQ =
- GE >=
- GT >
- LE <=
- LT <
- NE <>

4.4.2  GE >=

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

4.4.2.1  Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>ANY_NUM</td>
<td></td>
<td></td>
<td></td>
<td>First input.</td>
</tr>
<tr>
<td>IN2</td>
<td>ANY_NUM</td>
<td></td>
<td></td>
<td></td>
<td>Second input.</td>
</tr>
</tbody>
</table>

4.4.2.2  Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE if IN1 &gt;= IN2.</td>
</tr>
</tbody>
</table>

4.4.2.3  Remarks

- Both inputs must have the same type.
- Comparisons can be used with strings.
  - In this case, the lexical order is used for comparing the input strings.
  - Example: ABC is less than ZX; ABCD is greater than ABC.

4.4.2.4  FBD Language
4.4.2.5 FFLD Language

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

4.4.2.6 IL Language

- In the 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.

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

4.4.2.7 ST Language

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

See Also

- CMP
- EQ =
- GT >
- LE <=
- LT <
- NE <>

4.4.3 GT >

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

4.4.3.1 Inputs

IN1 : ANY  First input.
IN2 : ANY  Second input.

4.4.3.2 Outputs

Q : BOOL  TRUE if IN1 > IN2
4.4.3.3 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".

4.4.3.4 ST Language

\[ Q := \text{IN1} > \text{IN2}; \]

4.4.3.5 FBD Language

\[ \text{IN1} \gg \text{IN2} \rightarrow Q \]

4.4.3.6 FFLD Language

\[ \text{EN} \rightarrow (\text{IN1} > \text{IN2}) \rightarrow Q \]

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

4.4.3.7 IL Language

\[ \text{Op1: FFLD IN1 GT IN2 ST Q} \quad (* \text{Q is true if IN1 > IN2} *) \]

See Also
- CMP
- EQ =
- GE >=
- LE <=
- LT <
- NE <>

4.4.4 EQ =

\[ \text{PLCopen } \checkmark \]

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

4.4.4.1 Inputs
## Input Data Type Range Unit Default Description

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>First input.</td>
</tr>
<tr>
<td>IN2</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Second input.</td>
</tr>
</tbody>
</table>

### 4.4.4.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE</td>
<td>TRUE if IN1 = IN2.</td>
</tr>
</tbody>
</table>

### 4.4.4.3 Remarks

- Both inputs must have the same type.
- Comparisons can be used with strings.
  - In that case, the lexical order is used for comparing the input strings.
  - Example: 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.

### 4.4.4.4 FBD Language

```
IN1 := IN2
```

### 4.4.4.5 FFLD Language

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

```
EN [ ] = ( )
```

### 4.4.4.6 IL Language

- In the 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.

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

### 4.4.4.7 ST Language

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

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

4.4.5.1 Inputs
IN1 : ANY    First input.
IN2 : ANY    Second input.

4.4.5.2 Outputs
Q : BOOL    TRUE if IN1 is not equal to IN2.

4.4.5.3 Remarks
Both inputs must have the same data 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 data 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 has the resolution of the target cycle and test can be unsafe as some values can never be reached.

4.4.5.4 FBD Language

```
       <>
    IN1      Q
    IN2
```

4.4.5.5 FFLD Language
(*) The comparison is executed only if EN is TRUE. *)

```
EN  [<> { }  Q
   ]
IN1
IN2
```

4.4.5.6 IL Language

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

\[ Q := \text{IN1} <> \text{IN2}; \]

See Also
- CMP
- EQ =
- GE >=
- GT >
- LE <=
- LT <

4.4.6 LE <=

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

4.4.6.1 Inputs
IN1 : ANY  First input.
IN2 : ANY  Second input.

4.4.6.2 Outputs
Q : BOOL  TRUE if IN1 <= IN2.

4.4.6.3 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".

4.4.6.4 ST Language
Q := IN1 <= IN2;

4.4.6.5 FBD Language

4.4.6.6 FFLD Language
(* The comparison is executed only if EN is TRUE. *)

\[ \text{EN} \rightarrow [\text{IN1} <] \rightarrow Q \]
4.4.6.7 IL Language

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

See Also
- CMP
- EQ =
- GE >=
- GT >
- LT <
- NE <>

4.4.7 LT <

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

4.4.7.1 Inputs
IN1 : ANY    First input.
IN2 : ANY    Second input.

4.4.7.2 Outputs
Q : BOOL     TRUE if IN1 < IN2.

4.4.7.3 Remarks
Both inputs must have the same data 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 datatype.

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".

4.4.7.4 FBD Language

```
IN1 < Q
IN2
```

4.4.7.5 FFLD Language

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

```
<table>
<thead>
<tr>
<th></th>
<th>EN</th>
<th>IN1</th>
<th>IN2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```

4.4.7.6 IL Language
4.4.7.7  ST Language

\[ Q := \text{IN1} < \text{IN2}; \]

See Also
- CMP
- EQ =
- GE >=
- GT >
- LE <=
- NE <>

4.5  Type Conversion Functions

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;any_to_bool&quot; ( \to p. 112)</td>
<td>Converts to Boolean.</td>
</tr>
<tr>
<td>&quot;any_to_dint / any_to_udint&quot; (\to p. 114)</td>
<td>Converts to integer (32-bit - default).</td>
</tr>
<tr>
<td>&quot;any_to_int / any_to_uint&quot; (\to p. 115)</td>
<td>Converts to 16-bit integer.</td>
</tr>
<tr>
<td>&quot;any_to_lint / any_to_uint&quot; (\to p. 116)</td>
<td>Converts to long (64-bit) integer.</td>
</tr>
<tr>
<td>&quot;any_to_lreal&quot; (\to p. 117)</td>
<td>Converts to double precision real.</td>
</tr>
<tr>
<td>&quot;any_to_real&quot; (\to p. 118)</td>
<td>Converts to real.</td>
</tr>
<tr>
<td>&quot;any_to_sint / any_to_usint&quot; (\to p. 121)</td>
<td>Converts to small (8-bit) integer.</td>
</tr>
<tr>
<td>&quot;any_to_string&quot; (\to p. 122)</td>
<td>Converts to character string.</td>
</tr>
<tr>
<td>&quot;any_to_time&quot; (\to p. 120)</td>
<td>Converts to time.</td>
</tr>
<tr>
<td>&quot;NUM_TO_STRING&quot; (\to p. 123)</td>
<td>Converts a number to a string.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;bcd_to_bin&quot; ( \to p. 124)</td>
<td>Converts a BCD value to a binary value.</td>
</tr>
<tr>
<td>&quot;bin_to_bcd&quot; ( \to p. 125)</td>
<td>Converts a binary value to a BCD value.</td>
</tr>
</tbody>
</table>

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

4.5.1  any_to_bool

**PLCopen**: ✓

**Operator** - Converts the input into Boolean value.

4.5.1.1  Inputs
### 4.5.1.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td>Value</td>
<td></td>
<td>Value converted to Boolean.</td>
</tr>
</tbody>
</table>

### 4.5.1.3 Remarks

- For DINT, REAL, and TIME input data types, the result is FALSE if the input is 0 (zero).
  - The result is TRUE in all other cases.
- For STRING inputs, the output is TRUE if the input string is not empty.
  - The output is FALSE if the string is empty.

### 4.5.1.4 FBD Language

```
IN     ANY_TO_BOOL
Q
```

### 4.5.1.5 FFLD Language

- In the FFLD language, the conversion is executed only if the input rung (EN) is TRUE.
  - The output rung is the result of the conversion.
  - The output rung is FALSE if the EN is FALSE.

```
EN IN     ANY_TO_BOOL
[ ] [ ] [ ] [ ] [ ] Q
```

### 4.5.1.6 IL Language

- In the IL Language, the any_to_bool function converts the current result.

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

### 4.5.1.7 ST Language

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

### See Also

- `any_to_dint / any_to_udint`
- `any_to_int / any_to_uint`
- `any_to_lint / any_to_uuint`
- `any_to_lreal`
- `any_to_real`
- `any_to_sint / any_to_usint`
any_to_string
any_to_time

4.5.2 any_to_dint / any_to_udint

**Operator** - Converts the input into integer value.
Can be unsigned with any_to_udint.

### 4.5.2.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Input value.</td>
</tr>
</tbody>
</table>

### 4.5.2.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td></td>
<td></td>
<td>Value converted to a signed double integer. (32-bit).</td>
</tr>
<tr>
<td>Q</td>
<td>UDINT</td>
<td></td>
<td></td>
<td>Value converted to an unsigned double integer. (32-bit).</td>
</tr>
</tbody>
</table>

### 4.5.2.3 Remarks

- For BOOL input data types, the output is 0 (zero) 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 input data types, the output is the number represented by the string or 0 (zero) if the string does not represent a valid number.

### 4.5.2.4 FBD Language

```
IN    any_to_dint
    Q
```

### 4.5.2.5 FFLD Language

- In the 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.

```
EN [ ] [ ]
IN    any_to_dint       ENO
    ( ) [ ]
    Q
```

### 4.5.2.6 IL Language

- In the IL Language, the any_to_udint converts the current result.

```
Op1: FFLD    IN
     any_to_dint
     ST  Q
```
4.5.2.7 ST Language

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

See Also

- any_to_bool
- any_to_int / any_to_uint
- any_to_lint / any_to_ulint
- any_to_lreal
- any_to_real
- any_to_sint / any_to_usint
- any_to_string
- any_to_time

4.5.3 any_to_int / any_to_uint

**Operator** - Converts the input into 16-bit integer value.

Can be unsigned with any_to_uint.

4.5.3.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Input value.</td>
</tr>
</tbody>
</table>

4.5.3.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>INT</td>
<td></td>
<td></td>
<td>Value converted to a signed integer. (16-bit).</td>
</tr>
<tr>
<td>Q</td>
<td>UINT</td>
<td></td>
<td></td>
<td>Value converted to an unsigned integer. (16-bit).</td>
</tr>
</tbody>
</table>

4.5.3.3 Remarks

- For BOOL input data types, the output is 0 (zero) 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 input data types, the output is the number represented by the string or 0 (zero) if the string does not represent a valid number.

4.5.3.4 FBD Language

```
ANY_TO_INT
  
IN  Q
```

4.5.3.5 FFLD Language

- In the 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.
4.5.3.6 IL Language

- In the IL Language, the `any_to_int` converts the current result.

```
Op1: FFLD IN
    ANY_TO_INT
    ST Q
```

4.5.3.7 ST Language

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

See Also

- `any_to_bool`
- `any_to_dint / any_to_udint`
- `any_to_lint / any_to_ulint`
- `any_to_lreal`
- `any_to_real`
- `any_to_sint / any_to_usint`
- `any_to_string`
- `any_to_time`

4.5.4 `any_to_lint / any_to_ulint`

**Operator** - Converts the input into long (64-bit) integer value.

Can be unsigned with `any_to_ulint`.

### 4.5.4.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Input value.</td>
</tr>
</tbody>
</table>

### 4.5.4.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>LINT</td>
<td></td>
<td></td>
<td>Value converted to long (64-bit) integer.</td>
</tr>
<tr>
<td>Q</td>
<td>ULINT</td>
<td></td>
<td></td>
<td>Value converted to long (64-bit) unsigned integer.</td>
</tr>
</tbody>
</table>

### 4.5.4.3 Remarks

- For BOOL input data types, the output is 0 (zero) 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 input data types, the output is the number represented by the string or 0 (zero) if the string does not represent a valid number.

4.5.4.4 FBD Language

\[
\begin{array}{c}
\text{IN} \\
\text{EN} \rightarrow \text{Q}
\end{array}
\]

4.5.4.5 FFLD Language

- In the 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.

\[
\begin{array}{c}
\text{EN} \\
\text{IN} \rightarrow \text{Q}
\end{array}
\]

4.5.4.6 IL Language

- In IL Language, the \text{any_to_lint} converts the current result.

\[
\begin{array}{c}
\text{Op1: FFLD IN} \\
\text{ANY_TO_LINT} \\
\text{ST Q}
\end{array}
\]

4.5.4.7 ST Language

\[
\begin{array}{c}
\text{Q := ANY_TO_LINT (IN)};
\end{array}
\]

See Also

- \text{any_to_bool}
- \text{any_to_dint} / \text{any_to_udint}
- \text{any_to_int} / \text{any_to_uint}
- \text{any_to_lreal}
- \text{any_to_real}
- \text{any_to_sint} / \text{any_to_usint}
- \text{any_to_string}
- \text{any_to_time}

4.5.5 any_to_lreal

Operator - Converts the input into double precision floating point real value.

4.5.5.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td>ANY</td>
<td>null</td>
<td>null</td>
<td>Input value.</td>
</tr>
</tbody>
</table>

4.5.5.2 Outputs
### 4.5.5.3 Remarks

- For BOOL input data types, the output is 0.0 or 1.0.
- For DINT input data types, the output is the same number.
- For TIME input data types, the result is the number of milliseconds.
- For STRING input data types, the output is the number represented by the string or 0.0 if the string does not represent a valid number.

### 4.5.5.4 FBD Language

```
IN ANY_TO_LREAL Q
```

### 4.5.5.5 FFLD Language

- In the 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.

```
EN [ ] [ ANY_TO_LREAL ] ( [ ] )
  IN ANY_TO_LREAL
  Q
```

### 4.5.5.6 IL Language

- In IL Language, the `any_to_lreal` converts the current result.

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

### 4.5.5.7 ST Language

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

### See Also

- `any_to_bool`
- `any_to_dint / any_to_udint`
- `any_to_int / any_to_uint`
- `any_to_lint / any_to_ulint`
- `any_to_real`
- `any_to_sint / any_to_usint`
- `any_to_string`
- `any_to_time`

### 4.5.6 any_to_real

PLCopen ✅
**Operator** - Converts the input into single precision floating point real value.

### 4.5.6.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Input value.</td>
</tr>
</tbody>
</table>

### 4.5.6.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Value converted to single precision floating point real.</td>
</tr>
</tbody>
</table>

### 4.5.6.3 Remarks

- For BOOL input data types, the output is 0.0 or 1.0.
- For DINT input data types, the output is the same number.
- For TIME input data types, the result is the number of milliseconds.
- For STRING input data types, the output is the number represented by the string or 0.0 if the string does not represent a valid number.

### 4.5.6.4 FBD Language

```
IN ANY_TO_REAL Q
```

### 4.5.6.5 FFLD Language

- In the 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.

```
EN [ ANY_TO_REAL ] Q
```

### 4.5.6.6 IL Language

- In the IL Language, the `any_to_real` converts the current result.

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

### 4.5.6.7 ST Language

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

See Also

- `any_to_bool`
- `any_to_dint / any_to_udint`
- `any_to_int / any_to_uint`
4.5.7 any_to_time

**Operator** - Converts the input into time value.

### 4.5.7.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Input value.</td>
</tr>
</tbody>
</table>

### 4.5.7.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>TIME</td>
<td></td>
<td></td>
<td>Value converted to time.</td>
</tr>
</tbody>
</table>

### 4.5.7.3 Remarks

- For BOOL input data types, the output is t#0ms or t#1ms.
- For DINT or REAL input data type, the output is the time represented by the input number as a number of milliseconds.
- For STRING input data types, the output is the time represented by the string or t#0ms if the string does not represent a valid time.

### 4.5.7.4 FBD Language

```
IN  ANY_TO_TIME  Q
```

### 4.5.7.5 FFLD Language

- In the 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.

```
EN [ ANY_TO_TIME ( ) ]  ENO
  | IN     | Q   |
```

### 4.5.7.6 IL Language

- In the IL Language, the `any_to_time` converts the current result.

```
Op1: FFLD  IN
     ANY_TO_TIME
     ST  Q
```
### 4.5.7.7 ST Language

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

**See Also**

- `any_to_bool`
- `any_to_dint / any_to_udint`
- `any_to_int / any_to_uint`
- `any_to_lint / any_to_ulint`
- `any_to_lreal`
- `any_to_real`
- `any_to_sint / any_to_usint`
- `any_to_string`

### 4.5.8 any_to_sint / any_to_usint

**Operator** - Converts the input into a short (8 bit) integer value.

Can be unsigned with `any_to_usint`.

#### 4.5.8.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Input value.</td>
</tr>
</tbody>
</table>

#### 4.5.8.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>SINT</td>
<td></td>
<td></td>
<td>Value converted to a signed short integer. (8-bit).</td>
</tr>
<tr>
<td>Q</td>
<td>USINT</td>
<td></td>
<td></td>
<td>Value converted to an unsigned short integer. (8-bit).</td>
</tr>
</tbody>
</table>

#### 4.5.8.3 Remarks

- For BOOL input data types, the output is 0 (zero) 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 input data types, the output is the number represented by the string or 0 (zero) if the string does not represent a valid number.

#### 4.5.8.4 FBD Language

```
IN   ANY_TO_SINT
     Q
```

#### 4.5.8.5 FFLD Language

- In the 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.
4.5.8.6 IL Language

- In the IL Language, the `any_to_sint` converts the current result.

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

4.5.8.7 ST Language

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

See Also

- `any_to_bool`
- `any_to_dint / any_to_udint`
- `any_to_int / any_to_uint`
- `any_to_lint / any_to_ulint`
- `any_to_lreal`
- `any_to_real`
- `any_to_string`
- `any_to_time`

4.5.9 any_to_string

ANY_TO_STRING

**Operator** - Converts the input into string value.

4.5.9.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Input value.</td>
</tr>
</tbody>
</table>

4.5.9.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>STRING</td>
<td></td>
<td></td>
<td>Value converted to a string.</td>
</tr>
</tbody>
</table>

4.5.9.3 Remarks

- For BOOL input data types, the output is
  - 0 for FALSE.
  - 1 for TRUE.
- For DINT, REAL, or TIME input data types, the output is the string representation of the input
This is a number of milliseconds for TIME inputs.

### 4.5.9.4 FBD Language

```
IN    ANY_TO_STRING
     Q
```

### 4.5.9.5 FFLD Language

- In the 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.

```
EN    [ ] ANY_TO_STRING { } ENO
     IN    Q
```

### 4.5.9.6 IL Language

- In the IL language, the `any_to_string` function converts the current result.

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

### 4.5.9.7 ST Language

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

#### See Also

- `any_to_bool`
- `any_to_dint / any_to_udint`
- `any_to_int / any_to_uint`
- `any_to_lint / any_to_ulint`
- `any_to_lreal`
- `any_to_real`
- `any_to_sint / any_to_usint`
- `any_to_time`

### 4.5.10 NUM_TO_STRING

- **Function**: Converts a number into string value.

#### 4.5.10.1 Inputs

- **IN**: ANY Input number.
- **WIDTH**: DINT Length of the output string (see remarks).
- **DIGITS**: DINT Number of digits after decimal point.

#### 4.5.10.2 Outputs
4.5.10.3 Remarks

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

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

```
Q := NUM_TO_STRING (1.333333, 0, 2); (* Q is '1.33' *)
```

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

```
Q := NUM_TO_STRING (123.4, 8, 2); (* Q is ' 123.40' *)
```

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

```
Q := NUM_TO_STRING (123.4, -8, 2); (* Q is '123.40 ' *)
```

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

```
Q := NUM_TO_STRING (1.333333, 3, 0); (* Q is ' 1' *)
```

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

```
Q := NUM_TO_STRING (1.333333, 0, 1); (* Q is '1.3' *)
```

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

```
Q := NUM_TO_STRING (1234, 3, 0); (* Q is '***' *)
```

### 4.5.11 bcd_to_bin

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

#### 4.5.11.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Integer value in BCD.</td>
</tr>
</tbody>
</table>

#### 4.5.11.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td></td>
<td></td>
<td>Value converted to integer or 0 (zero) if IN is not a valid positive BCD value.</td>
</tr>
</tbody>
</table>
4.5.11.2.1 Truth Table

<table>
<thead>
<tr>
<th>IN</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>0 (invalid)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16 (16#10)</td>
<td>10</td>
</tr>
<tr>
<td>15 (16#0F)</td>
<td>0 (invalid)</td>
</tr>
</tbody>
</table>

4.5.11.3 Remarks
- The input must:
  - be positive.
  - represent a valid BCD value.

4.5.11.4 FBD Language

```
IN  BCD_TO_BIN  Q
```

4.5.11.5 FFLD Language
- In the 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 [ BCD_TO_BIN Q ] ENO
```

4.5.11.6 IL Language
- In the IL language, the input must be loaded in the current result before calling the function.

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

4.5.11.7 ST Language

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

See Also
- `bin_to_bcd`

4.5.12 `bin_to_bcd`

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

4.5.12.1 Inputs
### 4.5.12.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td></td>
<td></td>
<td>Value converted to BCD or 0 if IN is less than 0.</td>
</tr>
</tbody>
</table>

### 4.5.12.3 Truth Table

<table>
<thead>
<tr>
<th>IN</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>0 (invalid)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>16 (16#10)</td>
</tr>
<tr>
<td>22</td>
<td>34 (16#22)</td>
</tr>
</tbody>
</table>

### 4.5.12.4 Remarks
- The input must be positive.

### 4.5.12.5 FBD Language

```
IN -- BIN_TO_BCD -- Q
```

### 4.5.12.6 FFLD Language
- In the 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.

```
EN [ -- ] IN [ BIN_TO_BCD ] ENO [ ( ) ] Q
```

### 4.5.12.7 IL Language
- In the IL language, the input must be loaded in the current result before calling the function.

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

### 4.5.12.8 ST Language

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

See Also

`bcd_to_bin`
4.6 Selectors
These are the standard functions that perform data selection:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SEL&quot; (➔ p. 131)</td>
<td>2 integer inputs</td>
</tr>
<tr>
<td>&quot;MUX4&quot; (➔ p. 127)</td>
<td>4 integer inputs</td>
</tr>
<tr>
<td>&quot;MUX8&quot; (➔ p. 129)</td>
<td>8 integer inputs</td>
</tr>
</tbody>
</table>

4.6.1 MUX4

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

4.6.1.1 Inputs

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>DINT</td>
<td>Selection command.</td>
</tr>
<tr>
<td>IN0</td>
<td>ANY</td>
<td>First input.</td>
</tr>
<tr>
<td>IN1</td>
<td>ANY</td>
<td>Second input.</td>
</tr>
<tr>
<td>IN2</td>
<td>ANY</td>
<td>Third input.</td>
</tr>
<tr>
<td>IN3</td>
<td>ANY</td>
<td>Last input.</td>
</tr>
</tbody>
</table>

4.6.1.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td>IN0 or IN1 ... or IN3 depending on K. See &quot;Truth Table&quot; (➔ p. 127).</td>
</tr>
</tbody>
</table>

4.6.1.3 Truth Table

<table>
<thead>
<tr>
<th>K</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IN0</td>
</tr>
<tr>
<td>1</td>
<td>IN1</td>
</tr>
<tr>
<td>2</td>
<td>IN2</td>
</tr>
<tr>
<td>3</td>
<td>IN3</td>
</tr>
<tr>
<td>Other</td>
<td>0</td>
</tr>
</tbody>
</table>

4.6.1.4 Remarks

- In FFLD language, the input rung (EN) enables the selection.
- The output rung keeps the state of 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 commas.

### 4.6.1.5 FBD Language

![FBD Diagram]

### 4.6.1.6 FFLD Language

(* the selection is performed only if EN is TRUE. *)

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

![FFLD Diagram]

### 4.6.1.7 IL Language

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

### 4.6.1.8 ST Language

```
Q := MUX4 (K, IN0, IN1, IN2, IN3);
```

See Also

- "MUX8" (➔ p. 129)
- "SEL" (➔ p. 131)
4.6.2 MUX8

Function - Select one of the inputs - 8 inputs.

4.6.2.1 Inputs

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>DINT</td>
<td>Selection command.</td>
</tr>
<tr>
<td>IN0</td>
<td>ANY</td>
<td>First input.</td>
</tr>
<tr>
<td>IN1</td>
<td>ANY</td>
<td>Second input.</td>
</tr>
<tr>
<td>IN2</td>
<td>ANY</td>
<td>Third input.</td>
</tr>
<tr>
<td>IN3</td>
<td>ANY</td>
<td>Fourth input.</td>
</tr>
<tr>
<td>IN4</td>
<td>ANY</td>
<td>Fifth input.</td>
</tr>
<tr>
<td>IN5</td>
<td>ANY</td>
<td>Sixth input.</td>
</tr>
<tr>
<td>IN6</td>
<td>ANY</td>
<td>Seventh input.</td>
</tr>
<tr>
<td>IN7</td>
<td>ANY</td>
<td>Last input.</td>
</tr>
</tbody>
</table>

4.6.2.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td>IN0 or IN1 ... or IN7 depending on K. See &quot;Truth Table&quot; (p. 129).</td>
</tr>
</tbody>
</table>

4.6.2.3 Truth Table

<table>
<thead>
<tr>
<th>K</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IN0</td>
</tr>
<tr>
<td>1</td>
<td>IN1</td>
</tr>
<tr>
<td>2</td>
<td>IN2</td>
</tr>
<tr>
<td>3</td>
<td>IN3</td>
</tr>
<tr>
<td>4</td>
<td>IN4</td>
</tr>
<tr>
<td>5</td>
<td>IN5</td>
</tr>
<tr>
<td>6</td>
<td>IN6</td>
</tr>
<tr>
<td>7</td>
<td>IN7</td>
</tr>
<tr>
<td>Other</td>
<td>0</td>
</tr>
</tbody>
</table>

4.6.2.4 Remarks

- In FFLD language, the input rung (EN) enables the selection.
  - The output rung keeps the state of 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 commas.
4.6.2.5 ST Language

\[ Q := \text{MUX8}(K, \text{IN0}, \text{IN1}, \text{IN2}, \text{IN3}, \text{IN4}, \text{IN5}, \text{IN6}, \text{IN7}); \]

4.6.2.6 FBD Language

![FBD Diagram]

4.6.2.7 FFLD Language

(* the selection is performed only if EN is TRUE *)

(* ENO has the same value as EN *)

![FFLD Diagram]

4.6.2.8 IL Language

Not available.

Op1: LD SELECT

MUX8 IN1, IN2, IN3, IN4, IN5, IN6, IN7, IN8

ST Q

See Also

- "MUX4" (→ p. 127)
- "SEL" (→ p. 131)
4.6.3 SEL

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

### 4.6.3.1 Inputs

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>BOOL</td>
<td>Selection command.</td>
</tr>
<tr>
<td>IN0</td>
<td>ANY</td>
<td>First input.</td>
</tr>
<tr>
<td>IN1</td>
<td>ANY</td>
<td>Second input.</td>
</tr>
</tbody>
</table>

### 4.6.3.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Q      | ANY       | • IN0 if G is FALSE  
          |            | • IN1 if G is TRUE |

### 4.6.3.3 Truth Table

<table>
<thead>
<tr>
<th>SELECT</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IN0</td>
</tr>
<tr>
<td>1</td>
<td>IN1</td>
</tr>
</tbody>
</table>

### 4.6.3.4 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 commas.

### 4.6.3.5 ST Language

```plaintext
Q := SEL (G, IN0, IN1);
```

### 4.6.3.6 FBD Language

![FBD Diagram](image)

### 4.6.3.7 FFLD Language
(* the input rung is the selector *)
(* ENO has the same value as SELECT *)

4.6.3.8 IL Language

Op1: LD SELECT
     SEL IN1, IN2
     ST Q

See Also

- "MUX4" (→ p. 127)
- "MUX8" (→ p. 129)
4.7 Registers

4.7.1 Standard Functions
These are the standard functions for managing 8- to 32-bit registers:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;rol&quot; (→ p. 147)</td>
<td>Rotate bits of a register to the left.</td>
</tr>
<tr>
<td>&quot;ror&quot; (→ p. 149)</td>
<td>Rotate bits of a register to the right.</td>
</tr>
<tr>
<td>&quot;shl&quot; (→ p. 154)</td>
<td>Shift bits of a register to the left.</td>
</tr>
<tr>
<td>&quot;shr&quot; (→ p. 155)</td>
<td>Shift bits of a register to the right.</td>
</tr>
</tbody>
</table>

4.7.2 Advanced Function
This is the advanced function for register manipulation:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;MBshift&quot; (→ p. 142)</td>
<td>Multi-byte shift / rotate.</td>
</tr>
</tbody>
</table>

4.7.3 Bit-to-Bit Functions
These functions enable bit-to-bit operations on a 8- to 32-bit integers:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;and_mask&quot; (→ p. 134)</td>
<td>Performs a bit-to-bit Boolean AND between two integer values.</td>
</tr>
<tr>
<td>&quot;not_mask&quot; (→ p. 144)</td>
<td>Performs a bit-to-bit Boolean negation of an integer value.</td>
</tr>
<tr>
<td>&quot;or_mask&quot; (→ p. 145)</td>
<td>Performs a bit-to-bit Boolean OR between two integer values.</td>
</tr>
<tr>
<td>&quot;xor_mask&quot; (→ p. 160)</td>
<td>Performs a bit to bit exclusive OR between two integer values.</td>
</tr>
</tbody>
</table>

4.7.4 Pack / Unpack Functions
These functions enable pack / unpack 8-, 16-, and 32-bit registers:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HiByte</td>
<td>Get the highest byte of a word.</td>
</tr>
<tr>
<td>HiWord</td>
<td>Get the highest word of a double word.</td>
</tr>
<tr>
<td>LoByte</td>
<td>Get the lowest byte of a word.</td>
</tr>
</tbody>
</table>

#### 4.7.5 Bit Access
These functions provide bit access in 8- to 32-bit integers:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoWord</td>
<td>Get the lowest word of a double word.</td>
</tr>
<tr>
<td>MakeDWord</td>
<td>Builds a double word as the concatenation of two words.</td>
</tr>
<tr>
<td>MakeWord</td>
<td>Builds a word as the concatenation of two bytes.</td>
</tr>
<tr>
<td>PACK8</td>
<td>Pack bits in a byte.</td>
</tr>
<tr>
<td>UNPACK8</td>
<td>Extract bits from a byte.</td>
</tr>
</tbody>
</table>

#### 4.7.6 Deprecated Functions
These functions have been deprecated.
- They are available for backwards compatibility only.
- The previous functions should be used for all current and future development.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND_WORD</td>
<td>AND_BYTE</td>
</tr>
<tr>
<td>NOT_WORD</td>
<td>NOT_BYTE</td>
</tr>
<tr>
<td>OR_WORD</td>
<td>OR_BYTE</td>
</tr>
<tr>
<td>ROLB</td>
<td>RORB</td>
</tr>
<tr>
<td>ROLW</td>
<td>RORW</td>
</tr>
<tr>
<td>SHLB</td>
<td>SHRB</td>
</tr>
<tr>
<td>SHLW</td>
<td>SHRW</td>
</tr>
<tr>
<td>XOR_WORD</td>
<td>XOR_BYTE</td>
</tr>
</tbody>
</table>

#### 4.7.7 and_mask

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>and_mask</td>
<td>Performs a bit-to-bit Boolean AND between two integer values.</td>
</tr>
</tbody>
</table>

##### 4.7.7.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>First input.</td>
</tr>
<tr>
<td>MSK</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Second input. (AND mask)</td>
</tr>
</tbody>
</table>
4.7.7.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td>ANY</td>
<td>ANY</td>
<td>AND mask between IN and MSK inputs.</td>
</tr>
</tbody>
</table>

4.7.7.3 Remarks

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

4.7.7.4 FBD Language

```
IN AND_MASK MSK
IN  AND_MASK  MSK
```

4.7.7.5 FFLD Language

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

```
EN \[\]
IN AND_MASK MSK
IN  AND_MASK  MSK
ENO \(\)q
```

4.7.7.6 IL Language

- In the 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.

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

4.7.7.7 ST Language

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

See Also

- not_mask
- or_mask
- xor_mask

4.7.8 HiByte

```
PLCopen ✓
```
**Function** - Get the highest byte of a word.

### 4.7.8.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>UINT</td>
<td>16-bit register.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.7.8.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>USINT</td>
<td>Highest significant byte.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.7.8.3 Remarks

None

### 4.7.8.4 FBD Language

```
IN HiByte Q
```

### 4.7.8.5 FFLD Language

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

```
EN [IN] [HiByte] ENO ( ) Q
```

### 4.7.8.6 IL Language

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

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

### 4.7.8.7 ST Language

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

**See Also**

- HiWord
- LoByte
- LoWord
4.7.9 LoByte

**Function** - Get the lowest byte of a word.

### 4.7.9.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>UINT</td>
<td>16-bit</td>
<td></td>
<td></td>
<td>16-bit register.</td>
</tr>
</tbody>
</table>

### 4.7.9.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>USINT</td>
<td></td>
<td></td>
<td>Lowest significant byte.</td>
</tr>
</tbody>
</table>

### 4.7.9.3 Remarks

None

### 4.7.9.4 FBD Language

```
IN   LoByte  Q
```

### 4.7.9.5 FFLD Language

- In the 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.

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

```
EN  LoByte  ENO
[   ] [    ] [    ]
IN  (    ) Q
```

### 4.7.9.6 IL Language

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

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

### 4.7.9.7 ST Language
Q := LOBYTE (IN);

See Also
- HiByte
- HiWord
- LoWord
- MakeDWord
- MakeWord

4.7.10 HiWord

Function - Get the highest word of a double word.

4.7.10.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>UDINT</td>
<td>32-bit register.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.7.10.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>UINT</td>
<td>Highest significant word.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.7.10.3 Remarks
None

4.7.10.4 FBD Language

4.7.10.5 FFLD Language
- In the 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.
- The function is executed only if EN is TRUE.
- ENO keeps the same value as EN.

4.7.10.6 IL Language
- In the IL language, the input must be loaded in the current result before calling the function.
4.7.10.7 ST Language

Q := HIWORD (IN);

See Also
- HiByte
- LoByte
- LoWord
- MakeDWord
- MakeWord

4.7.11 LoWord

Function - Get the lowest word of a double word.

4.7.11.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>UDINT</td>
<td>32-bit register.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.7.11.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>UINT</td>
<td>Lowest significant word.</td>
</tr>
</tbody>
</table>

4.7.11.3 Remarks

None

4.7.11.4 FBD Language

<table>
<thead>
<tr>
<th>IN</th>
<th>LoWord</th>
<th>Q</th>
</tr>
</thead>
</table>

4.7.11.5 FFLD Language

- In the 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.
- The function is executed only if EN is TRUE.
  - ENO keeps the same value as EN.
4.7.11.6 IL Language

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

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

4.7.11.7 ST Language

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

See Also

- HiByte
- HiWord
- LoByte
- MakeDWord
- MakeWord

4.7.12 MakeDWord

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

### 4.7.12.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HI</td>
<td>USINT</td>
<td></td>
<td></td>
<td></td>
<td>Highest significant word.</td>
</tr>
<tr>
<td>LO</td>
<td>USINT</td>
<td></td>
<td></td>
<td></td>
<td>Lowest significant word.</td>
</tr>
</tbody>
</table>

### 4.7.12.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>UINT</td>
<td></td>
<td></td>
<td>32-bit register.</td>
</tr>
</tbody>
</table>

### 4.7.12.3 Remarks

None

### 4.7.12.4 FBD Language
4.7.12.5  FFLD Language

- In the 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.

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

4.7.12.6  IL Language

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

```il
Op1: LD HI
     MADEWORD LO
     ST Q
```

4.7.12.7  ST Language

```st
Q := MADEWORD (HI, LO);
```

See Also

- HiByte
- HiWord
- LoByte
- LoWord
- MakeWord

4.7.13  MakeWord

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

4.7.13.1  Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HI</td>
<td>USINT</td>
<td></td>
<td></td>
<td></td>
<td>Highest significant byte.</td>
</tr>
<tr>
<td>LO</td>
<td>USINT</td>
<td></td>
<td></td>
<td></td>
<td>Lowest significant byte.</td>
</tr>
</tbody>
</table>
4.7.13.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>UINT</td>
<td>16-bit register.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.7.13.3 Remarks
None

4.7.13.4 FBD Language

![FBD Diagram]

4.7.13.5 FFLD Language
- In the 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.
- The function is executed only if EN is TRUE.
- ENO keeps the same value as EN.

![FFLD Diagram]

4.7.13.6 IL Language
- In the IL language, the input must be loaded in the current result before calling the function.

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

4.7.13.7 ST Language
```
Q := MAKEWORD (HI, LO);
```

See Also
- HiByte
- HiWord
- LoByte
- LoWord
- MakeDWord

4.7.14 MBshift
**Function** - Multi-byte shift / rotate.

### 4.7.14.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer</td>
<td>SINT / USINT</td>
<td>Array</td>
<td></td>
<td></td>
<td>Array of bytes.</td>
</tr>
<tr>
<td>Pos</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Base position in the array.</td>
</tr>
<tr>
<td>NbByte</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Number of bytes to be shifted or rotated.</td>
</tr>
<tr>
<td>NbShift</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Number of shifts or rotations.</td>
</tr>
<tr>
<td>ToRight</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td></td>
<td></td>
<td>• TRUE for right.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• TRUE for rotate.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• FALSE for left.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• FALSE for shift.</td>
</tr>
<tr>
<td>Rotate</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td></td>
<td></td>
<td>• TRUE for rotate.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• FALSE for shift.</td>
</tr>
<tr>
<td>InBit</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td></td>
<td></td>
<td>Bit to be introduced in a shift.</td>
</tr>
</tbody>
</table>

### 4.7.14.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE if successful.</td>
</tr>
</tbody>
</table>

### 4.7.14.3 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.

### 4.7.14.4 FBD Language

```
Buffer
Pos
NbByte
NbShift
ToRight
Rotate
InBit
   NBShift
   Q
```

### 4.7.14.5 FFLD Language

- In the FFLD language, the input rung (EN) enables the operation.
- The rung output is the result ("Q").
- The function is called only if EN is TRUE.
4.7.14.6 IL Language

Not available.

4.7.14.7 ST Language

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

4.7.15 not_mask

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

4.7.15.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Integer input.</td>
</tr>
</tbody>
</table>

4.7.15.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td></td>
<td></td>
<td>Bit to bit negation of the input.</td>
</tr>
</tbody>
</table>

4.7.15.3 Remarks

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

4.7.15.4 FBD Language

```
IN -- NOT_MASK -- Q
```

4.7.15.5 FFLD Language

- In the FFLD language, the input rung (EN) enables the operation.
  - The output rung keeps the state of the input rung.
  - The function is executed only if EN is TRUE.
  - ENO has the same value as EN.
4.7.15.6 IL Language

- In the 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.

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

4.7.15.7 ST Language

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

See Also

- and_mask
- or_mask
- xor_mask

4.7.16 or_mask

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

4.7.16.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td>ANY</td>
<td></td>
<td></td>
<td>First input.</td>
</tr>
<tr>
<td>MSK</td>
<td>ANY</td>
<td>ANY</td>
<td></td>
<td></td>
<td>Second input. (OR mask)</td>
</tr>
</tbody>
</table>

4.7.16.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td>ANY</td>
<td></td>
<td>OR mask between IN and MSK inputs.</td>
</tr>
</tbody>
</table>

4.7.16.3 Remarks

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

4.7.16.4 FBD Language

```
IN  OR_MASK
    Q
```
4.7.16.5 FFLD Language

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

![FFLD Language Diagram]

4.7.16.6 IL Language

- In the 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.

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

4.7.16.7 ST Language

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

See Also

- `and_mask`
- `not_mask`
- `xor_mask`

4.7.17 PACK8

**PLCopen**

**Function** - Pack bits in a byte.

4.7.17.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN0</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Less significant bit.</td>
</tr>
<tr>
<td>IN7</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Highest significant bit.</td>
</tr>
</tbody>
</table>

4.7.17.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>USINT</td>
<td></td>
<td></td>
<td>Byte built with input bits.</td>
</tr>
</tbody>
</table>

4.7.17.3 Remarks

None
4.7.17.4 FBD Language

```
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7
```

4.7.17.5 FFLD Language

- In the FFLD language, the input rung is the IN0 input.
  - The output rung (ENO) keeps the same value as the input rung.
- ENO keeps the same value as EN.

```
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7
```

4.7.17.6 IL Language

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

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

4.7.17.7 ST Language

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

See Also

UNPACK8

4.7.18 rol

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

### 4.7.18.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Register.</td>
</tr>
<tr>
<td>NBR</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Number of rotations (each rotation is 1 bit).</td>
</tr>
</tbody>
</table>

### 4.7.18.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td></td>
<td></td>
<td>Rotated register.</td>
</tr>
</tbody>
</table>

### 4.7.18.3 Remarks

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

### 4.7.18.3.1 Diagram

![Diagram of Rotate bits](image)

### 4.7.18.4 FBD Language

```
IN ROL Q
NBR
```

### 4.7.18.5 FFLD Language

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

```
EN [ ] ROL ( | ) ENO
IN
NBR
```

### 4.7.18.6 IL Language

- In the IL language, the first input must be loaded before the function call.
  - The second input is the operand of the function.

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

\[ Q := \text{ROL} \ (\text{IN}, \ NBR); \]

See Also

- "ror" (\(\rightarrow\) p. 149)
- "shr" (\(\rightarrow\) p. 155)
- These other links were in the Copa-Data topic but not ours - should they be added: SHRb ROLb SHLw SHRw ROLw

4.7.19 ror

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

4.7.19.1  Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Register.</td>
</tr>
<tr>
<td>NBR</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Number of rotations (each rotation is 1 bit).</td>
</tr>
</tbody>
</table>

4.7.19.2  Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td></td>
<td></td>
<td>Rotated register.</td>
</tr>
</tbody>
</table>

4.7.19.3  Remarks

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

4.7.19.3.1  Diagram

```
   NBR
  \|
```

4.7.19.4  FBD Language

```
IN ROR Q
```

4.7.19.5  FFLD Language

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

### 4.7.19.6 IL Language
- In the IL language, the first input must be loaded before the function call.
  - The second input is the operand of the function.

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

### 4.7.19.7 ST Language
```
Q := ROR (IN, NBR);
```

See Also
- "rol" (→ p. 147)
- "shl" (→ p. 154)
- "shr" (→ p. 155)
- These other links were in the Copa-Data topic but not ours - should they be added:
  SHRb ROLb SHLw SHRw ROLw
4.7.20 RORb / ROR_SINT / ROR_USINT / ROR_BYTE

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

4.7.20.1 Inputs

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

4.7.20.2 Outputs

Q : SINT    Rotated register

4.7.20.3 Diagram

4.7.20.4 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.

4.7.20.5 ST Language

Q := RORb (IN, NBR);

4.7.20.6 FBD Language

4.7.20.7 FFLD Language

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

4.7.20.8 IL Language

Op1: FFLD IN
    RORb NBR
    ST Q

4.7.20.9 See also

SHL SHLb SHLw SHR SHRb SHRw ROL ROLb ROLw ROR RORb RORw
4.7.21 RORw / ROR_INT / ROR_UINT / ROR_WORD

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

4.7.21.1 Inputs

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

4.7.21.2 Outputs

| Q    | INT | Rotated register         |

4.7.21.3 Diagram

![Diagram showing rotation of bits]

4.7.21.4 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.

4.7.21.5 ST Language

Q := RORw (IN, NBR);

4.7.21.6 FBD Language

![FBD diagram showing rotation]

4.7.21.7 FFLD Language

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

(* ENO has the same value as EN *)

4.7.21.8 IL Language

Op1: FFLD IN
RORw NBR
ST Q

4.7.21.9 See also

SHL  SHR  ROL  ROR  SHLb  SHRb  ROLb  RORb  SHLw  SHRw  ROLw
4.7.22 SetBit

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

### 4.7.22.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td>8-bit</td>
<td></td>
<td>64-bit</td>
<td>8- to 64-bit integer register.</td>
</tr>
<tr>
<td>BIT</td>
<td>DINT</td>
<td>Bit number</td>
<td></td>
<td>0</td>
<td>Bit number (0 = less significant bit).</td>
</tr>
<tr>
<td>VAL</td>
<td>BOOL</td>
<td>Bit value</td>
<td></td>
<td></td>
<td>Bit value to apply.</td>
</tr>
</tbody>
</table>

### 4.7.22.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td>Modified</td>
<td></td>
<td>Modified register.</td>
</tr>
</tbody>
</table>

### 4.7.22.3 Remarks

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

### 4.7.22.4 FBD Language

```
IN
BIT
VAL
SetBit
Q
```

### 4.7.22.5 FFLD Language

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

```
EN
IN
BIT
VAL
SetBit
ENO
Q
```

### 4.7.22.6 IL Language

Not available.
4.7.22.7  ST Language

\[ Q := \text{SETBIT} \ (\text{IN}, \ 	ext{BIT}, \ \text{VAL}); \]

See Also
"TestBit" (→ p. 158)

4.7.23 shl

PLCopen  

Function - Shift bits of a register to the left.

4.7.23.1  Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Register.</td>
</tr>
<tr>
<td>NBS</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Number of shifts (each shift is 1 bit).</td>
</tr>
</tbody>
</table>

4.7.23.2  Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td></td>
<td></td>
<td>Shifted register.</td>
</tr>
</tbody>
</table>

4.7.23.3  Remarks

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

4.7.23.3.1  Diagram

4.7.23.4  FBD Language

4.7.23.5  FFLD Language

- In the FFLD language, the input rung (EN) enables the operation.
  - The output rung keeps the state of the input rung.
  - The shift is executed only if EN is TRUE.
  - ENO has the same value as EN.
4.7.23.6 IL Language

- In the IL language, the first input must be loaded before the function call.
  - The second input is the operand of the function.

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

4.7.23.7 ST Language

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

See Also

- "rol" (→ p. 147)
- "ror" (→ p. 149)
- "shr" (→ p. 155)

These other links were in the Copa-Data topic but not ours - should they be added:
- SHRb ROLb SHLw SHRw ROLw

4.7.24 SHR

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

### 4.7.24.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Register.</td>
</tr>
<tr>
<td>NBS</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Number of shifts (each shift is 1 bit).</td>
</tr>
</tbody>
</table>

### 4.7.24.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td></td>
<td></td>
<td>Shifted register.</td>
</tr>
</tbody>
</table>

### 4.7.24.3 Remarks

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

4.7.24.3.1 Diagram

- these bullets are in the Copa-Data topic but not ours - should we add it?
- If the option **SHR: do not duplicate the most significant bit** is checked in the Project settings
Advanced box, then the most significant bit is set to FALSE.

- If the option is not checked, then the most significant bit is duplicated:

```
nbs
\[\]
0
```

### 4.7.24.4 FBD Language

```
in
shr
nbs
q
```

### 4.7.24.5 FFLD Language

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

```
en
[ ]
shr
[ ]
eno
in
nbs
q
```

### 4.7.24.6 IL Language

- In the IL language, the first input must be loaded before the function call.
  - The second input is the operand of the function.

```
op1: ld in
shr nbs
st q
```

### 4.7.24.7 ST Language

```
q := shr (in, nbs);
```

### See Also

- "rol" (➡ p. 147)
- "ror" (➡ p. 149)
- "shl" (➡ p. 154)
- These other links were in the Copa-Data topic but not ours - should they be added:
  - SHRb ROLb SHLw SHRw ROLw

### 4.7.25 SWAB

**PLCopen**

**Function** - Swap the bytes of an integer.
4.7.25.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td>No range</td>
<td>N/A</td>
<td>No default</td>
<td>Any signed or unsigned integer.</td>
</tr>
</tbody>
</table>

4.7.25.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td>No range</td>
<td>N/A</td>
<td>Swapped value.</td>
</tr>
</tbody>
</table>

4.7.25.3 Remarks

Supported data types are:
- DINT
- DWORD
- INT
- LINT
- LWORD
- SINT
- UDINT
- UINT
- ULINT
- USINT
- WORD

- SINT and USINT inputs result in the same output value because they are only 1 byte wide.
- If the function is called for another data type, the output takes the value of the input.

Examples

<table>
<thead>
<tr>
<th>Type</th>
<th>IN</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>16#1A2B</td>
<td>16#2B1A</td>
</tr>
<tr>
<td>DWORD</td>
<td>16#1A2B3C4D</td>
<td>16#4D3C2B1A</td>
</tr>
</tbody>
</table>

4.7.25.4 FBD Language

```
SWAB

IN Q
```

4.7.25.5 FFLD Language

```
SWAB

En OK

IN Q
```

4.7.25.6 IL Language

Not available.

4.7.25.7 ST Language

```
swappedValue := SWAB(value);
```
4.7.26 TestBit

Function - Test a bit of an integer register.

4.7.26.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td>8- to 64-bit integer register.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIT</td>
<td>DINT</td>
<td>Bit number (0 = less significant bit).</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.7.26.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Bit value.</td>
</tr>
</tbody>
</table>

4.7.26.3 Remarks

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

4.7.26.4 FBD Language

```
IN
BIT
TestBit
Q
```

4.7.26.5 FFLD Language

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

```
EN] [)
IN
BIT
TestBit
Q
```

4.7.26.6 IL Language

Not available.

4.7.26.7 ST Language

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

See Also
"SetBit" (→ p. 153)

4.7.27 UNPACK8

**Function Block** - Extract bits from a byte.

### 4.7.27.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>USINT</td>
<td>8-bit</td>
<td></td>
<td></td>
<td>8-bit register.</td>
</tr>
</tbody>
</table>

### 4.7.27.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q0</td>
<td>BOOL</td>
<td>Less</td>
<td></td>
<td>Less significant bit.</td>
</tr>
<tr>
<td>Q7</td>
<td>BOOL</td>
<td>Highest</td>
<td></td>
<td>Highest significant bit.</td>
</tr>
</tbody>
</table>

### 4.7.27.3 Remarks

- The operation is executed only in the input rung (EN) is TRUE.

### 4.7.27.4 FBD Language

```
    IN
    "Unpack8"
    Q0
    Q1
    Q2
    Q3
    Q4
    Q5
    Q6
    Q7
```

### 4.7.27.5 FFLD Language

- In the FFLD language, the output rung is the Q0 output.
- The operation is performed if EN = TRUE.
4.7.27.6 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

4.7.27.7 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;

See Also

"PACK8" (→ p. 146)

4.7.28 xor_mask

PLCopen

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

4.7.28.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ANY</td>
<td>First input.</td>
</tr>
<tr>
<td>MSK</td>
<td>ANY</td>
<td>Second input. (XOR mask)</td>
</tr>
</tbody>
</table>

4.7.28.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>ANY</td>
<td>Exclusive OR mask between IN and MSK inputs.</td>
</tr>
</tbody>
</table>

4.7.28.3 Remarks

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

4.7.28.4 FBD Language
4.7.28.5 FFLD Language

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

```
<table>
<thead>
<tr>
<th>EN</th>
<th>XOR_MASK</th>
<th>ENO</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>MSK</td>
<td>Q</td>
</tr>
</tbody>
</table>
```

4.7.28.6 IL Language

- In the IL language, the input must be loaded in the current result before calling the function.
  - The other input is the operands of the function.

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

4.7.28.7 ST Language

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

See Also

- "and_mask" (→ p. 134)
- "not_mask" (→ p. 144)
- "or_mask" (→ p. 145)

4.8 Counters

These are the standard blocks for managing counters:

<table>
<thead>
<tr>
<th>Function Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;CTD / CTDr&quot; (→ p. 161)</td>
<td>Down Counter</td>
</tr>
<tr>
<td>&quot;CTU / CTUr&quot; (→ p. 163)</td>
<td>Up counter</td>
</tr>
<tr>
<td>&quot;CTUD / CTUDr&quot; (→ p. 164)</td>
<td>Up / Down Counter</td>
</tr>
</tbody>
</table>

4.8.1 CTD / CTDr

![PLCopen](https://kdn.kollmorgen.com)

Function Block - Down counter.
4.8.1.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Enable counting. Counter is decreased on each call when CD is TRUE.</td>
</tr>
<tr>
<td>LOAD</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Re-load command. Counter is set to PV when called with LOAD to TRUE.</td>
</tr>
<tr>
<td>PV</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Programmed maximum value.</td>
</tr>
</tbody>
</table>

4.8.1.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CV</td>
<td>DINT</td>
<td></td>
<td></td>
<td>Current value of the counter.</td>
</tr>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE when counter is empty (i.e., when CV = 0).</td>
</tr>
</tbody>
</table>

4.8.1.3 Remarks

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

4.8.1.4 FBD Language

```
CD, LOAD, PV

CTD

Q, CV
```

4.8.1.5 FFLD Language

- In the FFLD language, CD is the input rung.
- The output rung is the Q output.

```
CD, LOAD, PV

CTD

Q, CV
```

4.8.1.6 IL Language

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

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

See Also
- CTU / CTUr
- CTUD / CTUDr

4.8.2 CTU / CTUr

Function Block - Up counter.

4.8.2.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CU</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Enable counting. Counter is increased on each call when CU is TRUE.</td>
</tr>
<tr>
<td>PV</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Programmed maximum value.</td>
</tr>
<tr>
<td>RESET</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Reset command. Counter is reset to 0 when called with RESET to TRUE.</td>
</tr>
</tbody>
</table>

4.8.2.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CV</td>
<td>DINT</td>
<td></td>
<td></td>
<td>Current value of the counter.</td>
</tr>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE when counter is full (i.e., when CV = PV).</td>
</tr>
</tbody>
</table>

4.8.2.3 Remarks

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

4.8.2.4 FBD Language

LD MyCounter.CV
   CV
4.8.2.5 FFLD Language

- In the FFLD language, CU is the input rung.
  - The output rung is the Q output.

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Enable counting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Counter is decreased on each call when CD is TRUE.</td>
</tr>
</tbody>
</table>
### 4.8.3.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CV</td>
<td>DINT</td>
<td></td>
<td></td>
<td>Current value of the counter.</td>
</tr>
<tr>
<td>QD</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE when counter is empty (i.e., when CV = 0).</td>
</tr>
<tr>
<td>QU</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE when counter is full (i.e., when CV = PV).</td>
</tr>
</tbody>
</table>

### 4.8.3.3 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 the R_TRIG or F_TRIG function blocks for counting pulses of CU or CD input signals.
- 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.

### 4.8.3.4 FBD Language

```
CU  ---  CTUD  ---  QU
CD  ---  QD  ---  CV
RESET  ---  CV
LOAD  ---  ( )
```

### 4.8.3.5 FFLD Language

- In FFLD language, CU is the input rung.
  - The output rung is the QU output.

```
CU --- CTUD[QU] --- QD --- CV
CD --- QU
RESET --- ( )
LOAD --- CV
PV ---
```

### 4.8.3.6 IL Language
4.8.3.7 ST Language

MyCounter (CU, CD, RESET, LOAD, PV);
QU := MyCounter.QU;
QD := MyCounter.QD;
CV := MyCounter.CV;

See Also
"CTU / CTUr" (➔ p. 163)

4.9 Timers

These are the standard functions for managing timers:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;BLINK&quot; (➔ p. 166)</td>
<td>Blinker</td>
</tr>
<tr>
<td>&quot;BlinkA&quot; (➔ p. 167)</td>
<td>Asymmetric blinker</td>
</tr>
<tr>
<td>&quot;PLS&quot; (➔ p. 169)</td>
<td>Pulse signal generator</td>
</tr>
<tr>
<td>&quot;TMD&quot; (➔ p. 171)</td>
<td>Down-counting stop watch</td>
</tr>
<tr>
<td>&quot;TMU / TMUsec&quot; (➔ p. 173)</td>
<td>Up-counting stop watch (seconds)</td>
</tr>
<tr>
<td>&quot;TOF / TOFR&quot; (➔ p. 175)</td>
<td>Off timer</td>
</tr>
<tr>
<td>&quot;TON&quot; (➔ p. 177)</td>
<td>On timer</td>
</tr>
<tr>
<td>&quot;TP / TPR&quot; (➔ p. 178)</td>
<td>Pulse timer</td>
</tr>
</tbody>
</table>

4.9.1 BLINK

Function Block - Blinker.

4.9.1.1 Inputs

RUN : BOOL   Enabling command
CYCLE : TIME Blinking period

4.9.1.2 Outputs

Q : BOOL   Output blinking signal

4.9.1.3 Time diagram
4.9.1.4 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.

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

4.9.1.6 FBD Language

4.9.1.7 FFLD Language

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

4.9.2 BlinkA

Function Block - Asymmetric blinker.

4.9.2.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Enabling command.</td>
</tr>
<tr>
<td>TM0</td>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
<td>Duration of FALSE state on output.</td>
</tr>
<tr>
<td>TM1</td>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
<td>Duration of TRUE state on output.</td>
</tr>
</tbody>
</table>
4.9.2.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Output blinking signal.</td>
</tr>
</tbody>
</table>

4.9.2.3 Time Diagram

4.9.2.4 Remarks
- The output signal is FALSE when the RUN input is FALSE.

4.9.2.5 FBD Language

```
BlinkA
RUN
TM0
TM1
```

4.9.2.6 FFLD Language
- In the FFLD language, the input rung is the IN command.
- The output rung is the Q output.

```
RUN [ ] [BlinkA Q ]
TM0
TM1
```

4.9.2.7 IL Language

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

4.9.2.8 ST Language

168
(* MyBlinker is a declared instance of BLINKA function block. *)
MyBlinker (RUN, TM0, TM1);
Q := MyBlinker.Q;

See Also
- TOF / TOFR
- TON
- TP / TPR

4.9.3 PLS

Function Block - Pulse signal generator

4.9.3.1 Inputs
RUN : BOOL    Enabling command.
CYCLE : TIME  Signal period.

4.9.3.2 Outputs
Q : BOOL      Output pulse signal.

4.9.3.3 Time diagram

4.9.3.4 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.

4.9.3.5 ST Language

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

4.9.3.6 FBD Language

4.9.3.7 FFLD Language
RUN → PLS → Q ( ) → CYCLE
4.9.3.8 IL Language

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

See Also

- TOF / TOFR
- TON
- TP / TPR

4.9.4 Sig_Gen

Function Block - Generator of pseudo-analogical Signal

4.9.4.1 Inputs
RUN : BOOL  Enabling command
PERIOD : TIME  Signal period
MAXIMUM : DINT  Maximum growth during the signal period

4.9.4.2 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

4.9.4.3 FFLD Language

4.9.5 TMD

Function Block - Down-counting stop watch.
4.9.5.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>The time counts when this input is TRUE.</td>
</tr>
<tr>
<td>RST</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Timer is reset to 0 (zero) when this input is TRUE.</td>
</tr>
<tr>
<td>PT</td>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
<td>Programmed time.</td>
</tr>
</tbody>
</table>

4.9.5.2 Outputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Timer elapsed output signal.</td>
</tr>
<tr>
<td>ET</td>
<td>TIME</td>
<td></td>
<td></td>
<td>Elapsed time.</td>
</tr>
</tbody>
</table>

4.9.5.3 Time Diagram

![Time Diagram](image)

4.9.5.4 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.

4.9.5.5 FBD Language

```
IN RST PT

TMD Q ST
```

4.9.5.6 FFLD Language

```
IN RST PT

TMD Q ET
```
4.9.5.7 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

4.9.5.8 ST Language

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

See Also

"TMU / TMUsec" (➜ p. 173)

4.9.6 TMU / TMUsec

Function Block - Up-counting stop watch.

4.9.6.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>The time counts when this input is TRUE.</td>
</tr>
<tr>
<td>RST</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Timer is reset to 0 (zero) when this input is TRUE.</td>
</tr>
<tr>
<td>PT</td>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
<td>Programmed time. (TMU)</td>
</tr>
<tr>
<td>PTsec</td>
<td>UDWORD</td>
<td></td>
<td></td>
<td></td>
<td>Programmed time. (TMU - seconds)</td>
</tr>
</tbody>
</table>

4.9.6.2 Outputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Timer elapsed output signal.</td>
</tr>
<tr>
<td>ET</td>
<td>TIME</td>
<td></td>
<td></td>
<td>Elapsed time. (TMU)</td>
</tr>
<tr>
<td>ETsec</td>
<td>UDWORD</td>
<td></td>
<td></td>
<td>Elapsed time. (TMU - seconds)</td>
</tr>
</tbody>
</table>

4.9.6.3 Time Diagram
4.9.6.4 Remarks

TMUsec is identical to TMU except that the parameter is a number of seconds.

- 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.

4.9.6.5 FBD Language

```
IN
RST
PT
```

4.9.6.6 FFLD Language

```
IN
RST
PT
```

4.9.6.7 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" (→ p. 171)

4.9.7 TOF / TOFR

Function Block - Off timer.

4.9.7.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Timer command.</td>
</tr>
<tr>
<td>PT</td>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
<td>Programmed time.</td>
</tr>
<tr>
<td>RST</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Reset (TOFR only).</td>
</tr>
</tbody>
</table>

4.9.7.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Timer elapsed output signal.</td>
</tr>
<tr>
<td>ET</td>
<td>TIME</td>
<td></td>
<td></td>
<td>Elapsed time.</td>
</tr>
</tbody>
</table>

4.9.7.3 Time Diagram

4.9.7.4 Remarks

- TOFR is same as TOF but has an extra input for resetting the timer.
- 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 (zero).
- The output signal is set to TRUE when the IN input rises to TRUE.
  - It is reset to FALSE when the programmed time is elapsed.

4.9.7.5 FBD Language
4.9.7.6 FFLD Language

- In the FFLD language, the input rung is the IN command.
- The output rung is the Q output. - CAN THIS BE USED INSTEAD?
- The output rung is Q the output signal.

4.9.7.7 IL Language

\[
\begin{align*}
\text{Op1: } & \text{CAL MyTimer (IN, PT)} \\
& \text{FFLD MyTimer.Q} \\
& \text{ST Q} \\
& \text{FFLD MyTimer.ET} \\
& \text{ST ET}
\end{align*}
\]

4.9.7.8 ST Language

\[
\begin{align*}
\text{(* MyTimer is a declared instance of TOF function block *)} \\
\text{MyTimer (IN, PT);} \\
\text{Q := MyTimer.Q;} \\
\text{ET := MyTimer.ET;}
\end{align*}
\]

See Also

- "BLINK" (\(\rightarrow\) p. 166)
- "TON" (\(\rightarrow\) p. 177)
- "TP / TPR" (\(\rightarrow\) p. 178)
4.9.8 TON

**Function Block** - On timer.

### 4.9.8.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Timer command.</td>
</tr>
<tr>
<td>PT</td>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
<td>Programmed time.</td>
</tr>
</tbody>
</table>

### 4.9.8.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Timer elapsed output signal.</td>
</tr>
<tr>
<td>ET</td>
<td>TIME</td>
<td></td>
<td></td>
<td>Elapsed time.</td>
</tr>
</tbody>
</table>

### 4.9.8.3 Time Diagram

![Time Diagram](image)

### 4.9.8.4 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 (zero).

- The output signal is set to TRUE when programmed time is elapsed.
  - It is reset to FALSE when the input command falls.

### 4.9.8.5 FBD Language

```
IN
TON
Q
PT
ET
```

### 4.9.8.6 FFLD Language

- In the FFLD language, the input rung is the IN command.
  - The output rung is the Q output. - CAN THIS BE USED INSTEAD?
  - The output rung is Q the output signal.
4.9.8.7 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

4.9.8.8 ST Language

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

See Also
- "BLINK" (→ p. 166)
- "TOF / TOFR" (→ p. 175)
- "TP / TPR" (→ p. 178)

4.9.9 TP / TPR

Function Block - Pulse timer.

4.9.9.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Timer command.</td>
<td></td>
</tr>
<tr>
<td>PT</td>
<td>TIME</td>
<td></td>
<td></td>
<td>Programmed time.</td>
<td></td>
</tr>
<tr>
<td>RST</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Reset (TPR only).</td>
<td></td>
</tr>
</tbody>
</table>

4.9.9.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Timer elapsed output signal.</td>
</tr>
<tr>
<td>ET</td>
<td>TIME</td>
<td></td>
<td></td>
<td>Elapsed time.</td>
</tr>
</tbody>
</table>

4.9.9.3 Time Diagram
4.9.9.4 Remarks

- TPR is same as TP but has an extra input for resetting the timer.
- 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 (zero) but 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.

4.9.9.5 FBD Language

```
IN  TP  Q  
PT  ET  
```

4.9.9.6 FFLD Language

- In the FFLD language, the input rung is the IN command.
  - The output rung is the Q output. - CAN THIS BE USED INSTEAD?
  - The output rung is Q the output signal.

```
IN  [  ]  Q  
PT  TP  ET  
```

4.9.9.7 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
```

4.9.9.8 ST Language

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

- "BLINK" (→ p. 166)
- "TOF / TOFR" (→ p. 175)
- "TON" (→ p. 177)

### 4.10 Mathematic Operations

These are the standard functions that perform mathematic calculation:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;s asILN / LNL&quot; (→ p. 184)</td>
<td>Natural logarithm</td>
</tr>
<tr>
<td>abs / absL</td>
<td>Absolute value</td>
</tr>
<tr>
<td>&quot;EXP / EXPL&quot; (→ p. 182)</td>
<td>Power</td>
</tr>
<tr>
<td>expt</td>
<td></td>
</tr>
<tr>
<td>POW ** POWL</td>
<td></td>
</tr>
<tr>
<td>LOG / LOGL</td>
<td>Logarithm</td>
</tr>
<tr>
<td>&quot;ROOT&quot; (→ p. 186)</td>
<td>Root extraction</td>
</tr>
<tr>
<td>ScaleLin</td>
<td>Scaling - linear conversion</td>
</tr>
<tr>
<td>SQRT / SQRTL</td>
<td>Square root</td>
</tr>
<tr>
<td>trunc / truncL</td>
<td>Integer part</td>
</tr>
</tbody>
</table>

#### 4.10.1 abs / absL

Function - Returns the absolute value of the input.

##### 4.10.1.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td></td>
<td>Any value.</td>
</tr>
</tbody>
</table>

##### 4.10.1.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td>Result: Absolute value of IN.</td>
</tr>
</tbody>
</table>

##### 4.10.1.3 Remarks

None

##### 4.10.1.4 FBD Language
4.10.1.5 FFLD Language

- In the 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.

4.10.1.6 IL Language

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

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

4.10.1.7 ST Language

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

See Also

- LOG / LOGL
- POW ** POWL
- SQRT / SQRTL
- trunc / truncL

4.10.2 expt

**Function** - Calculates a power.

4.10.2.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>Real value.</td>
</tr>
<tr>
<td>EXP</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Exponent.</td>
</tr>
</tbody>
</table>

4.10.2.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Result: IN at the EXP power</td>
</tr>
</tbody>
</table>

4.10.2.3 Remarks

The exponent (second input of the function) must be the operand of the function.
4.10.2.4 FBD Language

![FBD Diagram]

4.10.2.5 FFLD Language

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

![FFLD Diagram]

4.10.2.6 IL Language

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

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

4.10.2.7 ST Language

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

See Also

- abs / absL
- LOG / LOGL
- SQRT / SQRTL
- trunc / truncL

4.10.3 EXP / EXPL

![PLCopen Icon]

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

### 4.10.3.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td></td>
<td>Real value.</td>
</tr>
</tbody>
</table>

### 4.10.3.2 Outputs
<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>REAL/LREAL</td>
<td></td>
<td></td>
<td>Result: Natural exponential of IN.</td>
</tr>
</tbody>
</table>

4.10.3.3 Remarks
None

4.10.3.4 FBD Language

![FBD Diagram]

4.10.3.5 FFLD Language
- In the 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.
  - The function is executed only if EN is TRUE.
  - ENO has the same value as EN.

4.10.3.6 IL Language
- In the IL language, the first input must be loaded before the function call.

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

4.10.3.7 ST Language
```
Q := EXP (IN);
```

4.10.4 LOG / LOGL

![PLCopen Check]

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

4.10.4.1 Inputs
IN : REAL/LREAL  Real value.

4.10.4.2 Outputs
Q : REAL/LREAL  Result: logarithm (base 10) of IN.
4.10.4.3 Remarks

- In the 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 the IL language, the first input must be loaded before the function call.

4.10.4.4 ST Language

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

4.10.4.5 FBD Language

```
IN   LOG   Q
```

4.10.4.6 FFLD Language

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

```
EN   LOG ( )   ENO
IN   Q
```

4.10.4.7 IL Language

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

See Also

- abs / absL
- POW ** POWL
- SQRT / SQRTL
- trunc / truncL

4.10.5’s asILN / LNL

```
 PLCopen ✓
```

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

4.10.5.1 Inputs

IN : REAL/LREAL  Real value.

4.10.5.2 Outputs

Q : REAL/LREAL  Result: natural logarithm of IN.

4.10.5.3 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.

### 4.10.5.4 ST Language

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

### 4.10.5.5 FBD Language

```
IN --LN-> Q
```

### 4.10.5.6 FFLD Language

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

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

```
EN | LN | ENO
IN | Q  |
```

### 4.10.5.7 IL Language

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

### 4.10.6 POW ** POWL

**PLCopen** ✓

*Function* - Calculates a power.

#### 4.10.6.1 Inputs

IN : REAL/LREAL  
Real value.

EXP : REAL/LREAL  
Exponent.

#### 4.10.6.2 Outputs

Q : REAL/LREAL  
Result: IN at the ‘EXP’ power.

#### 4.10.6.3 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.

#### 4.10.6.4 ST Language

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

#### 4.10.6.5 FBD Language
4.10.6.6 FFLD Language

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

4.10.6.7 IL Language

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

See Also

- abs / absL
- LOG / LOGL
- SQRT / SQRTL
- trunc / truncL

4.10.7 ROOT

Function - Calculates the Nth root of the input.

4.10.7.1 Inputs

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

4.10.7.2 Outputs

Q : REAL Result: Nth root of IN.

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

4.10.7.4 ST Language

Q := ROOT (IN, N);

4.10.7.5 FBD Language
4.10.7.6 FFLD Language

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

4.10.7.7 IL Language

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

4.10.8 ScaleLin

Function - Scaling - linear conversion.

4.10.8.1 Inputs

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>REAL</td>
<td>Real value.</td>
</tr>
<tr>
<td>IMIN</td>
<td>REAL</td>
<td>Minimum input value.</td>
</tr>
<tr>
<td>IMAX</td>
<td>REAL</td>
<td>Minimum input value.</td>
</tr>
<tr>
<td>OMIN</td>
<td>REAL</td>
<td>Minimum output value.</td>
</tr>
<tr>
<td>OMAX</td>
<td>REAL</td>
<td>Minimum output value.</td>
</tr>
</tbody>
</table>

4.10.8.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>REAL</td>
<td>Result: OMIN + IN * (OMAX - OMIN) / (IMAX - IMIN).</td>
</tr>
</tbody>
</table>

4.10.8.3 Truth Table

<table>
<thead>
<tr>
<th>Inputs</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMIN &gt;=</td>
<td>IN</td>
</tr>
<tr>
<td>IMAX</td>
<td></td>
</tr>
<tr>
<td>IN &lt;</td>
<td>OMIN</td>
</tr>
<tr>
<td>IMIN</td>
<td></td>
</tr>
<tr>
<td>IN &gt;</td>
<td>OMAX</td>
</tr>
<tr>
<td>IMAX</td>
<td></td>
</tr>
</tbody>
</table>
### 4.10.8.4 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.

### 4.10.8.5 ST Language

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

### 4.10.8.6 FBD Language

![FBD Diagram]

### 4.10.8.7 FFLD Language

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

(* ENO keeps the same value as EN *)

![FFLD Diagram]

### 4.10.8.8 IL Language

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

### 4.10.9 SQRT / SQRTL

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

#### 4.10.9.1 Inputs

- **IN**: REAL/LREAL Real value.

#### 4.10.9.2 Outputs

- **Q**: REAL/LREAL Result: square root of IN.

#### 4.10.9.3 Remarks
• In FFLD language, the operation is executed only if the input rung (EN) is TRUE.
  • The output rung (ENO) keeps the same value as the input rung.
• In the IL language, the first input must be loaded before the function call.

### 4.10.9.4 ST Language

\[
Q := \text{SQRT} \ (\text{IN});
\]

### 4.10.9.5 FBD Language

![FBD Diagram](image)

### 4.10.9.6 FFLD Language

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

### 4.10.9.7 IL Language

\[
\text{Op1: LD IN SQRT ST Q} \ (* \ Q \ is: \ \text{SQRT} \ (\text{IN}) \ *)
\]

**See Also**

• "abs / absL" (⇒ p. 180)
• "LOG / LOGL" (⇒ p. 183)
• "POW ** POWL" (⇒ p. 185)
• "trunc / truncL" (⇒ p. 190)
4.10.10 trunc / truncL

Function - Truncates the decimal part of the input.

4.10.10.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td></td>
<td>Real value.</td>
</tr>
</tbody>
</table>

4.10.10.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td>Result: Integer part of N.</td>
</tr>
</tbody>
</table>

4.10.10.3 Remarks

None

4.10.10.4 FBD Language

```
IN  TRUNC  Q
```

4.10.10.5 FFLD Language

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

```
EN [ IN  TRUNC  Q ]
```

4.10.10.6 IL Language

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

```
Op1: LD IN
     TRUNC
     ST Q  (* Q is the integer part of IN *)
```

4.10.10.7 ST Language

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

See Also
4.11 Trigonometric Functions

These are the standard functions for trigonometric calculation:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;acos / acosL&quot; (→ p. 191)</td>
<td>Arc-cosine</td>
</tr>
<tr>
<td>&quot;asin / asinL&quot; (→ p. 192)</td>
<td>Arc-sine</td>
</tr>
<tr>
<td>&quot;atan / atanL&quot; (→ p. 193)</td>
<td>Arc-tangent</td>
</tr>
<tr>
<td>&quot;atan2 / atan2L&quot; (→ p. 194)</td>
<td>Arc-tangent of Y / X</td>
</tr>
<tr>
<td>&quot;cos / cosL&quot; (→ p. 195)</td>
<td>Cosine</td>
</tr>
<tr>
<td>&quot;sin / sinL&quot; (→ p. 197)</td>
<td>Sine</td>
</tr>
<tr>
<td>&quot;tan / tanL&quot; (→ p. 198)</td>
<td>Tangent</td>
</tr>
</tbody>
</table>

See Also
"UseDegrees" (→ p. 199)

4.11.1 acos / acosL

Function - Calculate an arc-cosine.

4.11.1.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td></td>
<td>Real value.</td>
</tr>
</tbody>
</table>

4.11.1.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td></td>
<td>Result: arc-cosine of IN.</td>
</tr>
</tbody>
</table>

4.11.1.3 Remarks
None

4.11.1.4 FBD Language

```
  ACOS
Q
```

4.11.1.5 FFLD Language
In the 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.

4.11.1.6 IL Language

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

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

4.11.1.7 ST Language

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

See Also

- asin / asinL
- atan / atanL
- atan2 / atan2L
- cos / cosL
- sin / sinL
- tan / tanL

4.11.2 asin / asinL

Function - Calculate an arc-sine.

4.11.2.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td></td>
<td>Real value.</td>
</tr>
</tbody>
</table>

4.11.2.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td>Result: arc-sine of IN.</td>
</tr>
</tbody>
</table>

4.11.2.3 Remarks

None

4.11.2.4 FBD Language
4.11.2.5 FFLD Language

- In the 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.

4.11.2.6 IL Language

- In the IL language, the first input must be loaded before the function call.

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

4.11.2.7 ST Language

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

See Also

- "acos / acosL" (→ p. 191)
- "atan / atanL" (→ p. 193)
- "atan2 / atan2L" (→ p. 194)
- "cos / cosL" (→ p. 195)
- "sin / sinL" (→ p. 197)
- "tan / tanL" (→ p. 198)

4.11.3 atan / atanL

Function - Calculate an arc-tangent.

4.11.3.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td></td>
<td>Real value.</td>
</tr>
</tbody>
</table>

4.11.3.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td>Result: arc-tangent of IN.</td>
</tr>
</tbody>
</table>

4.11.3.3 Remarks
None

4.11.3.4 FBD Language

```
IN ATAN Q
```

4.11.3.5 FFLD Language

- In the 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.

```
EN [ ] ATAN ( )
IN Q ENO
```

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

4.11.3.6 IL Language

- In the IL language, the first input must be loaded before the function call.

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

4.11.3.7 ST Language

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

See Also

- acos / acosL
- asin / asinL
- atan2 / atan2L
- cos / cosL
- sin / sinL
- tan / tanL

4.11.4 atan2 / atan2L

PLCopen ✓

Function - Calculate arc-tangent of \( Y/X \).

4.11.4.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td></td>
<td>Real value.</td>
</tr>
<tr>
<td>Y</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td></td>
<td>Real value.</td>
</tr>
</tbody>
</table>
4.11.4.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td>Result: arc-tangent of X / Y.</td>
</tr>
</tbody>
</table>

4.11.4.3 Remarks
None

4.11.4.4 FBD Language

\[
\text{ATAN2} \\
\text{Y} \quad \rightarrow \quad \text{Q} \\
\text{X}
\]

4.11.4.5 FFLD Language

- In the 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.

\[
\text{EN} \quad [ \quad \text{ATAN2} \quad \text{Y} \quad \rightarrow \quad \text{Q} \quad \text{ENO} \quad ] \\
\text{X}
\]

4.11.4.6 IL Language

- In the IL language, the first input must be loaded before the function call.

```
Op1: LD Y
     ATAN2 X
     ST Q  (* Q is: ATAN2 (Y / X) *)
```

4.11.4.7 ST Language

is there an ST Language for this?

See Also

- acos / acosL
- asin / asinL
- 4.11.3 atan / atanL
- cos / cosL
- sin / sinL
- tan / tanL

4.11.5 cos / cosL

Kollmorgen | kdn.kollmorgen.com | December 2022 195
Function - Calculate a cosine.

4.11.5.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td></td>
<td>Real value.</td>
</tr>
</tbody>
</table>

4.11.5.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td>Result: cosine of IN.</td>
</tr>
</tbody>
</table>

4.11.5.3 Remarks

None

4.11.5.4 FBD Language

![FBD Diagram]

4.11.5.5 FFLD Language

- In the 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.
  - The function is executed only if EN is TRUE.

![FFLD Diagram]

4.11.5.6 IL Language

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

```il
Op1: LD  IN
    COS
    ST  Q  (* Q is: COS (IN) *)
```

4.11.5.7 ST Language

```st
Q := COS (IN);
```

See Also

- acos / acosL
- asin / asinL
- atan / atanL
- atan2 / atan2L
- sin / sinL
- tan / tanL
4.11.6 sin / sinL

Function - Calculate a sine.

4.11.6.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td></td>
<td>Real value.</td>
</tr>
</tbody>
</table>

4.11.6.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td>Result: Sine of IN.</td>
</tr>
</tbody>
</table>

4.11.6.3 Remarks

None

4.11.6.4 FBD Language

```
IN  SIN  Q
```

4.11.6.5 FFLD Language

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

```
EN   SIN   ENO
    IN   Q
```

4.11.6.6 IL Language

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

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

4.11.6.7 ST Language

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

See Also
4.11.7 tan / tanL

**Function** - Calculate a tangent.

### 4.11.7.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td></td>
<td>Real value.</td>
</tr>
</tbody>
</table>

### 4.11.7.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>REAL / LREAL</td>
<td></td>
<td></td>
<td>Result: Tangent of IN.</td>
</tr>
</tbody>
</table>

### 4.11.7.3 Remarks

None

### 4.11.7.4 FBD Language

```
IN  TAN   Q
```

### 4.11.7.5 FFLD Language

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

```
EN   [ ]
IN  TAN   ENO
     ( )
     Q
```

### 4.11.7.6 IL Language

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

```
Op1: LD  IN
     TAN
     ST  Q   (* Q is: TAN (IN) *)
```
4.11.7.7 ST Language

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

See Also
- ACOS
- asin / asinL
- ATAN
- ATAN2
- COS
- SIN

4.11.8 UseDegrees

![PLCopen](https://kollmorgen.com)

**Function** - Sets the unit for angles in all trigonometric functions.

4.11.8.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- If TRUE, turn all trigonometric functions to use degrees.
- If FALSE, turn all trigonometric functions to use radians (default).

4.11.8.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td>TRUE</td>
<td>TRUE if functions use degrees before the call.</td>
</tr>
</tbody>
</table>

4.11.8.3 Remarks

This function sets the working unit for these functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;acos / acosL&quot; (→ p. 191)</td>
<td>Arc-cosine</td>
</tr>
<tr>
<td>&quot;asin / asinL&quot; (→ p. 192)</td>
<td>Arc-sine</td>
</tr>
<tr>
<td>&quot;atan / atanL&quot; (→ p. 193)</td>
<td>Arc-tangent</td>
</tr>
<tr>
<td>&quot;atan2 / atan2L&quot; (→ p. 194)</td>
<td>Arc-tangent of Y / X</td>
</tr>
<tr>
<td>&quot;cos / cosL&quot; (→ p. 195)</td>
<td>Cosine</td>
</tr>
<tr>
<td>&quot;sin / sinL&quot; (→ p. 197)</td>
<td>Sine</td>
</tr>
<tr>
<td>&quot;tan / tanL&quot; (→ p. 198)</td>
<td>Tangent</td>
</tr>
</tbody>
</table>

4.11.8.4 FBD Language
4.11.8.5 FFLD Language

- The first input is the rung.
- The rung is the output.

```
IN [ ] UseDegrees ( ) Q
```

4.11.8.6 IL Language

```
Op1: LD IN
UseDegrees
ST Q
```

4.11.8.7 ST Language

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

4.12 String Operations

4.12.1 Standard Operators

These are the standard operators and functions that manage character strings:

<table>
<thead>
<tr>
<th>Functions and Operators</th>
<th>Operator / Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Addition +&quot; (→ p. 89)</td>
<td>Concatenation of strings.</td>
</tr>
<tr>
<td>&quot;ArrayToString / ArrayToStringU&quot; (→ p. 201)</td>
<td>Copies elements of an SINT array to a STRING.</td>
</tr>
<tr>
<td>&quot;ascii&quot; (→ p. 202)</td>
<td>Get the ASCII code of a character within a string.</td>
</tr>
<tr>
<td>&quot;ATOH&quot; (→ p. 203)</td>
<td>Converts string to integer using hexadecimal basis.</td>
</tr>
<tr>
<td>&quot;char&quot; (→ p. 204)</td>
<td>Build a single character string.</td>
</tr>
<tr>
<td>&quot;concat&quot; (→ p. 205)</td>
<td>Concatenation of strings.</td>
</tr>
<tr>
<td>&quot;CRC16&quot; (→ p. 206)</td>
<td>CRC16 calculation.</td>
</tr>
<tr>
<td>&quot;delete&quot; (→ p. 207)</td>
<td>Delete characters in a string.</td>
</tr>
<tr>
<td>&quot;FIND&quot; (→ p. 208)</td>
<td>Find characters in a string.</td>
</tr>
<tr>
<td>&quot;HTOA&quot; (→ p. 210)</td>
<td>Converts integer to string using hexadecimal basis.</td>
</tr>
<tr>
<td>&quot;INSERT&quot; (→ p. 211)</td>
<td>Insert characters in a string.</td>
</tr>
<tr>
<td>&quot;LEFT&quot; (→ p. 212)</td>
<td>Extract a part of a string on the left.</td>
</tr>
<tr>
<td>&quot;MID&quot; (→ p. 214)</td>
<td>Extract a part of a string.</td>
</tr>
<tr>
<td>&quot;MLEN&quot; (→ p. 215)</td>
<td>Get string length.</td>
</tr>
<tr>
<td>&quot;REPLACE&quot; (→ p. 216)</td>
<td>Replace characters in a string.</td>
</tr>
</tbody>
</table>
### 4.12.2 Manage String Tables

These functions are for managing string tables as resources:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;LoadString&quot; (→ p. 213)</td>
<td>Load a string from the active string table.</td>
</tr>
<tr>
<td>&quot;StringTable&quot; (→ p. 220)</td>
<td>Select the active string table resource.</td>
</tr>
</tbody>
</table>

### 4.12.3 ArrayToString / ArrayToStringU

**Function** - Copy an array of SINT to a STRING.

#### 4.12.3.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNT</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Number of characters to be copied.</td>
</tr>
<tr>
<td>DST</td>
<td>STRING</td>
<td></td>
<td></td>
<td></td>
<td>Destination STRING.</td>
</tr>
<tr>
<td>SRC</td>
<td>SINT</td>
<td></td>
<td></td>
<td></td>
<td>Source array of SINT small integers. USINT for ArrayToStringU.</td>
</tr>
</tbody>
</table>

#### 4.12.3.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td></td>
<td></td>
<td>Number of characters copied.</td>
</tr>
</tbody>
</table>

#### 4.12.3.3 Remarks

- 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 adjusts the COUNT number if necessary.

#### 4.12.3.4 FBD Language

```
ArrayToString
  Src
  Count
  Q
```

#### 4.12.3.5 FFLD Language

- In the 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.
4.12.3.6 IL Language
Not available.

4.12.3.7 ST Language

```plaintext
Q := ArrayToString (SRC, DST, COUNT);
```

See Also
`StringToArray / StringToArrayU`

4.12.4 ascii

| Function | Get the ASCII code of a character within a string. |

4.12.4.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>STRING</td>
<td></td>
<td></td>
<td></td>
<td>Input string.</td>
</tr>
<tr>
<td>POS</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Position of the character within the string. The first valid position is 1.</td>
</tr>
</tbody>
</table>

4.12.4.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
<td>DINT</td>
<td></td>
<td></td>
<td>ASCII code of the selected character or 0 (zero) if position is invalid.</td>
</tr>
</tbody>
</table>

4.12.4.3 Remarks
None

4.12.4.4 FBD Language

4.12.4.5 FFLD Language

- In the FFLD language, the input rung (EN) enables the operation.
- The output rung (ENO) keeps the same value as the input rung.
4.12.4.6 IL Language

- In the 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.

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

4.12.4.7 ST Language

```
CODE := ASCII (IN, POS);
```

See Also:
char

4.12.5 ATOH

Function - Converts string to integer using hexadecimal basis.

4.12.5.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>STRING</td>
<td></td>
<td></td>
<td></td>
<td>String representing an integer in hexadecimal format.</td>
</tr>
</tbody>
</table>

4.12.5.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td></td>
<td></td>
<td>Integer represented by the string.</td>
</tr>
</tbody>
</table>

4.12.5.3 Truth Table

<table>
<thead>
<tr>
<th>IN</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>' '</td>
<td>0</td>
</tr>
<tr>
<td>'12'</td>
<td>18</td>
</tr>
<tr>
<td>'a0'</td>
<td>160</td>
</tr>
<tr>
<td>'A0zzz'</td>
<td>160</td>
</tr>
</tbody>
</table>

4.12.5.4 Remarks
The function is case insensitive.
The result is 0 (zero) for an empty string.
The conversion stops before the first invalid character.

4.12.5.5 FBD Language

4.12.5.6 FFLD Language

In the 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.

4.12.5.7 IL Language

In the IL language, the first input must be loaded before the function call.

4.12.5.8 ST Language

Q := ATOH (IN);

See Also
HTOA

4.12.6 char

Function - Builds a single character string.

4.12.6.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>ASCII code of the specified character.</td>
</tr>
</tbody>
</table>

4.12.6.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>STRING</td>
<td></td>
<td></td>
<td>STRING containing only the specified character.</td>
</tr>
</tbody>
</table>
4.12.6.3 Remarks
None

4.12.6.4 FBD Language

```
CODE   CHAR   Q
```

4.12.6.5 FFLD Language
- In the FFLD language, the input rung (EN) enables the operation.
- The output rung (ENO) keeps the same value as the input rung.

```
EN [ ]
CODE   CHAR   ENO ( )
```

4.12.6.6 IL Language
- In the IL language, the input parameter (CODE) must be loaded in the current result before calling the function.

```
Op1:
LD   CODE
CHAR
ST   Q
```

4.12.6.7 ST Language

```
Q := CHAR (COD);e
```

See Also
ascii

4.12.7 concat

PLCopen

Function - Concatenate strings.

4.12.7.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN_1</td>
<td>STRING</td>
<td>Any</td>
<td></td>
<td></td>
<td>Any string variable or constant expression.</td>
</tr>
<tr>
<td>IN_N</td>
<td>STRING</td>
<td>Any</td>
<td></td>
<td></td>
<td>Any string variable or constant expression.</td>
</tr>
</tbody>
</table>

4.12.7.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>STRING</td>
<td></td>
<td></td>
<td>Concatenation of all inputs.</td>
</tr>
</tbody>
</table>
4.12.7.3 Remarks

- In the FBD or FFLD languages, the block can have up to 16 inputs.
- In the IL or ST languages, the function accepts a variable number of inputs (at least 2).
- Use the + operator to concatenate strings.

4.12.7.4 FBD Language

```
IN1  IN2  Q
  Concat
```

4.12.7.5 FFLD Language

```
EN  [ ]  [ ]
IN1  IN2
  Concat
  ENO
```

4.12.7.6 IL Language

```
Op1: FFLD  'AB'
       CONCAT  'CD', 'E'
       ST  Q  (* Q is now 'ABCDE' *)
```

4.12.7.7 ST Language

```
Q := CONCAT ('AB', 'CD', 'E');
(* now Q is 'ABCDE' *)
```

4.12.8 CRC16

**Function** - calculates a CRC16 on the characters of a string.

4.12.8.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>STRING</td>
<td></td>
<td></td>
<td></td>
<td>Character string.</td>
</tr>
</tbody>
</table>

4.12.8.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>INT</td>
<td></td>
<td></td>
<td>CRC16 calculated on all the characters of the string.</td>
</tr>
</tbody>
</table>

4.12.8.3 Remarks
• The function calculates a Modbus CRC16, initialized at 16#FFFF value.

4.12.8.4 FBD Language

4.12.8.5 FFLD Language

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

4.12.8.6 IL Language

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

4.12.8.7 ST Language

Q := CRC16 (IN);

4.12.9 delete

Function - Delete characters in a string.

4.12.9.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>STRING</td>
<td></td>
<td></td>
<td></td>
<td>Character string.</td>
</tr>
<tr>
<td>NBC</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Number of characters to be deleted.</td>
</tr>
<tr>
<td>POS</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Position of the first deleted character. The first character position is 1.</td>
</tr>
</tbody>
</table>

4.12.9.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>STRING</td>
<td></td>
<td></td>
<td>Modified string.</td>
</tr>
</tbody>
</table>

Kollmorgen | kdn.kollmorgen.com | December 2022 | 207
4.12.9.3 Remarks

- The first valid character position is 1.

4.12.9.4 FBD Language

```
IN  DELETE  Q
NBC
POS
```

4.12.9.5 FFLD Language

- In the 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.
  - The function is executed only if EN is TRUE.

```
EN  [ ]  DELETE  Q  ( )
IN
NBC
POS
```

4.12.9.6 IL Language

- In the IL language, 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.

```
Op1: LD IN
     DELETE NBC, POS
     ST Q
```

4.12.9.7 ST Language

```
Q := DELETE (IN, NBC, POS);
```

See Also

- Addition +
- FIND
- INSERT
- LEFT
- MID
- MLEN
- REPLACE
- RIGHT

4.12.10 FIND [PLCopen ✔]

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

| IN   | STRING | Character string. |
| STR  | STRING | Specific characters to search for within the STRING. |

4.12.10.2 Outputs

| POS  | DINT | Position of the first character of STR in IN, or 0 if not found. |

4.12.10.3 Remarks

The first valid character position is 1. A return value of 0 means that the STR substring has not been found. The return value can be used with other string functions such as MID and RIGHT. The 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.

4.12.10.4 ST Language

POS := FIND (IN, STR);

4.12.10.5 FBD Language

![FBD Diagram]

4.12.10.6 FFLD Language

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

4.12.10.7 IL Language

Op1: LD IN
FIND STR
ST POS

See Also

- Addition +
- delete
- INSERT
- LEFT
- MID
- MLEN
4.12.11 HTOA

**Function** - Converts integer to string using hexadecimal basis.

### 4.12.11.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Integer value.</td>
</tr>
</tbody>
</table>

### 4.12.11.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>STRING</td>
<td></td>
<td></td>
<td>String representing an integer in hexadecimal format.</td>
</tr>
</tbody>
</table>

### 4.12.11.3 Truth Table

<table>
<thead>
<tr>
<th>IN</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>'0'</td>
</tr>
<tr>
<td>18</td>
<td>'12'</td>
</tr>
<tr>
<td>160</td>
<td>'A0'</td>
</tr>
</tbody>
</table>

### 4.12.11.4 Remarks

None

### 4.12.11.5 FBD Language

```
IN--HTOA,Q
```

### 4.12.11.6 FFLD Language

- In the 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.

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

```
EN--HTOA( ),ENO
```

### 4.12.11.7 IL Language

- In the IL language, the input must be loaded in the current result before calling the function.
4.12.11.8 ST Language

Q := HTOA (IN);

See Also
ATOH

4.12.12 INSERT

*Function* - Insert characters in a string.

4.12.12.1 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).

4.12.12.2 Outputs
Q : STRING  Modified string.

4.12.12.3 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.

4.12.12.4 ST Language
Q := INSERT (IN, STR, POS);

4.12.12.5 FBD Language

4.12.12.6 FFLD Language
(* The function is executed only if EN is TRUE. *)
(* ENO keeps the same value as EN. *)
4.12.12.7 IL Language

Op1: LD IN
    INSERT STR, POS
    ST Q

See Also
- Addition +
- delete
- FIND
- LEFT
- MID
- MLEN
- REPLACE
- RIGHT

4.12.13 LEFT

Function - Extract characters of a string on the left.

4.12.13.1 Inputs
IN : STRING  Character string.
NBC : DINT   Number of characters to extract.

4.12.13.2 Outputs
Q : STRING   String containing the first NBC characters of IN.

4.12.13.3 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.

4.12.13.4 ST Language
Q := LEFT (IN, NBC);

4.12.13.5 FBD Language

4.12.13.6 FFLD Language
(* The function is executed only if EN is TRUE. *)
(* ENO keeps the same value as EN. *)
4.12.13.7 IL Language

Op1: LD IN
    LEFT NBC
    ST Q

See Also
- Addition +
- delete
- FIND
- INSERT
- MID
- MLEN
- REPLACE
- RIGHT

4.12.14 LoadString

*Function* - Load a string from the active string table.

4.12.14.1 Inputs
ID: DINT ID of the string as declared in the string table.

4.12.14.2 Outputs
Q: STRING Loaded string or empty string in case of error.

4.12.14.3 Remarks
This function loads a string from the active string table and stores it in 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" this identifier again - the system does it for you.

4.12.14.4 ST Language
Q := LoadString (ID);

4.12.14.5 FBD Language

```
LoadString
```

4.12.14.6 FFLD Language

```
<table>
<thead>
<tr>
<th>EN</th>
<th>[ ]</th>
<th>[ ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>
LoadString

4.12.14.7 IL Language
4.12.15 MID

*Function* - Extract characters of a string at any position.

### 4.12.15.1 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).

### 4.12.15.2 Outputs

**Q** : STRING  String containing the first NBC characters of IN.

### 4.12.15.3 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.

### 4.12.15.4 ST Language

\[
Q := \text{MID}(\text{IN}, \text{NBC}, \text{POS})
\]

### 4.12.15.5 FBD Language

```
IN  NBC  POS  Q
MID
```

### 4.12.15.6 FFLD Language

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

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

```
EN  IN  NBC  POS  Q  ENO
MID
```

### 4.12.15.7 IL Language

```
Op1: LD  IN  MID  NBC, POS  ST  Q
```
See Also

- Addition +
- delete
- FIND
- INSERT
- LEFT
- MLEN
- REPLACE
- RIGHT

4.12.16 MLEN

Function - Get the number of characters in a string.

4.12.16.1 Inputs
IN : STRING  Character string.

4.12.16.2 Outputs
NBC : DINT  Number of characters currently in the string. 0 if string is empty.

4.12.16.3 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 input must be loaded in the current result before calling the function.

4.12.16.4 ST Language
NBC := MLEN (IN);

4.12.16.5 FBD Language

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

4.12.16.7 IL Language
Op1: LD IN
MLEN
ST NBC

See Also
### 4.12.17 REPLACE

*Function* - Replace characters in a string.

#### 4.12.17.1 Inputs

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>STRING</td>
<td>Character string.</td>
</tr>
<tr>
<td>STR</td>
<td>STRING</td>
<td>String containing the characters to be inserted in place of NDEL removed characters.</td>
</tr>
<tr>
<td>NDEL</td>
<td>DINT</td>
<td>Number of characters to be deleted before insertion of STR.</td>
</tr>
<tr>
<td>POS</td>
<td>DINT</td>
<td>Position where characters are replaced (first character position is 1).</td>
</tr>
</tbody>
</table>

#### 4.12.17.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>STRING</td>
<td>Modified string.</td>
</tr>
</tbody>
</table>

#### 4.12.17.3 Remarks

- The first valid character position is 1.
- In the 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 the IL language, 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.

#### 4.12.17.4 ST Language

```
Q := REPLACE (IN, STR, NDEL, POS);
```
4.12.17.6 FFLD Language

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

\[
\begin{array}{c}
\text{EN} \\
\hline
\text{IN} \\
\text{STR} \\
\text{NDEL} \\
\text{POS} \\
\hline
\text{REPLACE} \\
\text{Q} \\
\hline
\end{array}
\]

4.12.17.7 IL Language

Op1: LD IN

```
REPLACE STR, NDEL, POS
ST Q
```

See Also

- "Addition +" (→ p. 89)
- "delete" (→ p. 207)
- "FIND" (→ p. 208)
- "INSERT" (→ p. 211)
- "LEFT" (→ p. 212)
- "MID" (→ p. 214)
- "MLEN" (→ p. 215)
- "RIGHT" (→ p. 218)
4.12.18 RIGHT

*Function* - Extract characters of a string on the right.

### 4.12.18.1 Inputs

IN : STRING Character string.
NBC : DINT Number of characters to extract.

### 4.12.18.2 Outputs

Q : STRING String containing the last NBC characters of IN.

### 4.12.18.3 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 the IL language, the first input (the string) must be loaded in the current result before calling the function.
  - The second input is the operand of the function.

### 4.12.18.4 ST Language

```plaintext
Q := RIGHT (IN, NBC);
```

### 4.12.18.5 FBD Language

![FBD Diagram]

### 4.12.18.6 FFLD Language

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

![FFLD Diagram]

### 4.12.18.7 IL Language

```plaintext
Op1:   LD   IN
       RIGHT  NBC
       ST     Q
```

**See Also**

- "delete" (→ p. 207)
- "FIND" (→ p. 208)
- "INSERT" (→ p. 211)
• "LEFT" (➔ p. 212)
• "MID" (➔ p. 214)
• "MLEN" (➔ p. 215)
• "REPLACE" (➔ p. 216)
4.12.19 StringTable

**Function** - Selects the active string table.

### 4.12.19.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>STRING</td>
<td></td>
<td></td>
<td></td>
<td>Name of the Sting Table resource. Must be a constant.</td>
</tr>
<tr>
<td>COL</td>
<td>STRING</td>
<td></td>
<td></td>
<td></td>
<td>Name of the column in the table. Must be a constant.</td>
</tr>
</tbody>
</table>

### 4.12.19.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE if OK.</td>
</tr>
</tbody>
</table>

### 4.12.19.3 Remarks

- This function selects a column of a valid String Table resource to become the active string table.
- The "LoadString" (p. 213) function always refers to the active string table.
- Arguments must:
  - be constant string expressions.
  - fit to a declared string table and a valid column name within this table.
- If there is only one string table with only one column defined in the project, you do not need to call this function.
  - It is the default string table.

### 4.12.19.4 FBD Language

```plaintext
TABLE = StringTable

OK = COL
```

### 4.12.19.5 FFLD Language

```plaintext
EN
[ ]
( )

StringTable OK

TABLE = COL
```

### 4.12.19.6 IL Language

```plaintext
Op1: LD

'MyTable'

StringTable 'First Column'
```
4.12.19.7 ST Language

OK := StringTable ('MyTable', 'FirstColumn');

See Also
- "LoadString" (→ p. 213)
- "String Table Resources" (→ p. 221)

4.12.19.8 String Table Resources

String tables are resources (embedded configuration data) edited with Workbench.

- A string table is a list of items identified by a name and referring to one or more character strings.
- String tables are typically used for defining static texts to be used in the application.
- These functions can be used for getting access to string tables in the programs:
  - "StringTable" (→ p. 220): selects the active string table.
  - "LoadString" (→ p. 213): Load a string from the active table.
- Each string table may contain several columns of texts for each item, and thus ease the localization of application, simply by defining a column for each language.
  - This way, the language can be selected dynamically at runtime by specifying the active language (as a column) in the StringTable() function.

The name entered in the string table as an ID is automatically declared for the compiler.

- The name:
  - Can directly be passed to the LoadString() function without re-declaring it.
  - Must conform to IEC standard naming rules.

You could do the same by declaring an array of STRING variables and enter some initial values for all items in the array.

- String tables provide significant advantages compared to arrays:
  - The editor provides a comfortable view of multiple columns at editing.
  - String tables are loaded in the application code and does not require any further RAM memory unlike declared arrays.
  - The string table editor automatically declares readable IDs for any string item to be used in programs instead of working with hard-coded index values.

⚠️ TIP ⚠️

If the text is too long for the STRING variable when used at runtime, it is truncated.
Use special $ sequences in strings to specify non printable characters, according to the IEC standard:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$</td>
<td>A &quot;$&quot; character.</td>
</tr>
<tr>
<td>$'</td>
<td>A Single quote.</td>
</tr>
<tr>
<td>$T</td>
<td>A tab stop (ASCII code 9).</td>
</tr>
<tr>
<td>$R</td>
<td>A carriage return character (ASCII code 13).</td>
</tr>
<tr>
<td>Code</td>
<td>Meaning</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>$L</td>
<td>A line feed character (ASCII code 10).</td>
</tr>
<tr>
<td>$N</td>
<td>Carriage return plus line feed characters (ASCII codes 13 and 10).</td>
</tr>
<tr>
<td>$P</td>
<td>A page break character (ASCII code 12).</td>
</tr>
<tr>
<td>$xx</td>
<td>Any character ($xx$ is the ASCII code expressed on two hexadecimal digits.)</td>
</tr>
</tbody>
</table>

4.12.20 **StringToArray / StringToArrayU**

**Function** - Copies the characters of a STRING to an array of SINT.

### 4.12.20.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DST</td>
<td>SINT</td>
<td></td>
<td></td>
<td></td>
<td>Destination array of SINT small integers. USINT for StringToArrayU.</td>
</tr>
<tr>
<td>SRC</td>
<td>STRING</td>
<td></td>
<td></td>
<td></td>
<td>Source STRING.</td>
</tr>
</tbody>
</table>

### 4.12.20.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td></td>
<td></td>
<td>Number of characters copied.</td>
</tr>
</tbody>
</table>

### 4.12.20.3 Remarks

This function:

- Copies the characters of the SRC string to the first characters of the DST array.
- Checks the maximum size destination arrays and reduces the number of copied characters if necessary.

### 4.12.20.4 FBD Language

```
Src
StringToArray
Dst
Q
```

### 4.12.20.5 FFLD Language

- In the 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.

```
EN [ ] StringToArray( ) [ ] ENO ( | )
Src
Src
Dst
Dst
Q
```

### 4.12.20.6 IL Language

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

```
Op1: LD
    SRC
    StringToArray DST
    ST Q
```

### 4.12.20.7 ST Language

```
Q := StringToArray (SRC, DST);
```

**See Also**

"ArrayToString / ArrayToStringU" (p. 201)
5 PLC Advanced Libraries

These are the standard blocks that perform advanced operations.

5.1 Analog Signal Processing

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>average / averageL</td>
<td>Calculates the average of signal samples.</td>
</tr>
<tr>
<td>CurveLin</td>
<td>Linear interpolation on a curve.</td>
</tr>
<tr>
<td>derivate</td>
<td>Computes the derivative of a signal with respect to time.</td>
</tr>
<tr>
<td>hyster</td>
<td>Hysteresis detection.</td>
</tr>
<tr>
<td>integral</td>
<td>Calculates the integral of a signal with respect to time.</td>
</tr>
<tr>
<td>lim_alarm</td>
<td>Detects high and low limits of a signal with hysteresis.</td>
</tr>
<tr>
<td>PID</td>
<td>PID loop.</td>
</tr>
<tr>
<td>RAMP</td>
<td>Ramp signal.</td>
</tr>
<tr>
<td>&quot;rand&quot; (~ p. 262)</td>
<td>Returns a pseudo-random integer value between 0 (zero) and (base - 1).</td>
</tr>
<tr>
<td>SigPlay</td>
<td>Play an analog signal from a resource.</td>
</tr>
<tr>
<td>SigScale</td>
<td>Get a point from a signal resource.</td>
</tr>
<tr>
<td>SurfLin</td>
<td>Linear interpolation on a surface.</td>
</tr>
</tbody>
</table>

5.2 Alarm Management

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Alarm_A&quot; (~ p. 225)</td>
<td>Alarm with automatic reset.</td>
</tr>
<tr>
<td>Alarm_M</td>
<td>Alarm with manual reset.</td>
</tr>
<tr>
<td>lim_alarm</td>
<td>Detects high and low limits of a signal with hysteresis.</td>
</tr>
</tbody>
</table>

5.3 Data Collections and Serialization

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO</td>
<td>Manages a first in / first out list.</td>
</tr>
<tr>
<td>LIFO</td>
<td>Last in / first out stack.</td>
</tr>
<tr>
<td>&quot;SerializeIn&quot; (~ p. 293)</td>
<td>Extract the value of a variable from a binary frame.</td>
</tr>
<tr>
<td>Block</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>&quot;SerializeOut&quot; (→ p. 296)</td>
<td>Copy the value of a variable to a binary frame.</td>
</tr>
<tr>
<td>stackint</td>
<td>Manages a stack of DINT integers.</td>
</tr>
</tbody>
</table>

### 5.4 Data Log

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;LogFileCSV&quot; (→ p. 255)</td>
<td>Create a log file in CSV format for a list of variables.</td>
</tr>
</tbody>
</table>

### 5.5 Special Operations

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyRecipeColumn</td>
<td>Apply the values of a column from a recipe file.</td>
</tr>
<tr>
<td>CycleStop</td>
<td>Sets the application in cycle stepping mode.</td>
</tr>
<tr>
<td>EnableEvents</td>
<td>Enable or disable the production of events for binding (runtime to runtime variable exchange).</td>
</tr>
<tr>
<td>FatalStop</td>
<td>Breaks the cycle and stop with fatal error.</td>
</tr>
<tr>
<td>GetSysInfo</td>
<td>Get system information.</td>
</tr>
<tr>
<td>printf</td>
<td>Trace messages.</td>
</tr>
<tr>
<td>SigID</td>
<td>Get the ID of a signal resource.</td>
</tr>
<tr>
<td>VLID</td>
<td>Get the ID of an embedded list of variables.</td>
</tr>
</tbody>
</table>

### 5.6 Communication

- AS-interface Functions
- TCP/IP Function Blocks
- UDP Functions for PxMM & Simulator

### 5.7 Others

- File Management
- File Tools Function Blocks
- Real Time Clock Management Functions

### 5.8 Alarm_A

**PLCopen**

Function Block - Alarm with automatic reset.
5.8.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Acknowledge command.</td>
</tr>
<tr>
<td>IN</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Process signal.</td>
</tr>
</tbody>
</table>

5.8.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE if alarm is active.</td>
</tr>
<tr>
<td>QACK</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE if alarm is acknowledged.</td>
</tr>
</tbody>
</table>

5.8.3 Remarks

- Combine this block with the lim_alarm block for managing analog alarms.

5.8.3.1 Sequence

5.8.4 FBD Language

```
IN
ACK
```

```
ALARM_A
Q
QACK
```

5.8.5 FFLD Language

```
IN
ACK
```

```
ALARM_A
Q
QACK
```

5.8.6 IL Language

```il
(* MyALARM is declared as an instance of ALARM_A function block *)
Op1: CAL
MyALARM (IN, ACK)
FFLD MyALARM.Q
ST Q
```
5.8.7 ST Language

(* MyALARM is declared as an instance of ALARM_A function block *)
MyALARM (IN, ACK, RST);
Q := MyALARM.Q;
QACK := MyALARM.QACK;

See Also
Alarm_M

5.9 Alarm_M

Function Block - Alarm with manual reset.

5.9.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Acknowledge command.</td>
</tr>
<tr>
<td>IN</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Process signal.</td>
</tr>
<tr>
<td>RST</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Reset command.</td>
</tr>
</tbody>
</table>

5.9.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td>TRUE</td>
<td></td>
<td>TRUE if alarm is active.</td>
</tr>
<tr>
<td>QACK</td>
<td>BOOL</td>
<td>TRUE</td>
<td></td>
<td>TRUE if alarm is acknowledged.</td>
</tr>
</tbody>
</table>

5.9.3 Remarks

- Combine this block with the lim_alarm block for managing analog alarms.

5.9.3.1 Sequence

5.9.4 FBD Language
5.9.5 FFLD 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

5.9.6 IL Language

MyALARM (IN, ACK, RST);
Q := MyALARM.Q;
QACK := MyALARM.QACK;

5.9.7 ST Language

(* MyALARM is declared as an instance of ALARM_M function block *)
MyALARM (IN, ACK, RST);
Q := MyALARM.Q;
QACK := MyALARM.QACK;

See Also
Alarm_A

5.10 ApplyRecipeColumn

**Function** - Apply the values of a column from a recipe file.

5.10.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>STRING</td>
<td></td>
<td></td>
<td></td>
<td>Path name of the recipe file (.CSV or .RCP). Must be a constant value.</td>
</tr>
<tr>
<td>COL</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Index of the column in the recipe (0 (zero) based).</td>
</tr>
</tbody>
</table>
5.10.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>BOOL</td>
<td></td>
<td>TRUE</td>
<td>TRUE if OK.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FALSE</td>
<td>FALSE if parameters are invalid.</td>
</tr>
</tbody>
</table>

5.10.3 Remarks

- The FILE input is a constant string expression specifying the path name of a valid .CSV or .RCP file.
  - If no path is specified, the file is assumed to be located in the project folder.
  - CSV files are created using Excel or Notepad.
  - RCP files are created using an external recipe editor.

- 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).

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
```

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

![Image of network diagram]

Column 3 corresponds to column E in the Excel sheet because this parameter is 0 based

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>comment lines here</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TravelSpeed</td>
<td>100</td>
<td>200</td>
<td>300</td>
<td>400</td>
<td>500</td>
</tr>
<tr>
<td>3</td>
<td>MasterAbsPos</td>
<td>0</td>
<td>45</td>
<td>90</td>
<td>135</td>
<td>180</td>
</tr>
<tr>
<td>4</td>
<td>MasterDeltaPos</td>
<td>0</td>
<td>90</td>
<td>180</td>
<td>270</td>
<td>360</td>
</tr>
<tr>
<td>5</td>
<td>MachineSpeed</td>
<td>50</td>
<td>100</td>
<td>150</td>
<td>200</td>
<td>250</td>
</tr>
<tr>
<td>6</td>
<td>MachineState</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Result displayed in the Dictionary when the application is running

Example of RCP File

```
@COLNAME=Col3 Col4
@SIZECOL1=100
@SIZECOL2=100
@SIZECOL3=100
@SIZECOL4=100
bCommand
tPerio
bFast
Blink1
test_var
bOut
@EXPANDED=Blink1
```

**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.

5.10.4 FBD Language

5.10.5 FFLD Language

- In the FFLD language, the operation is executed only if the input rung (EN) is TRUE.
  - The output rung is the result of the function.
- The function is executed only if **ApplyRecipe** is TRUE.
5.10.6 IL Language

Op1: LD
'MyFile.rcp'
ApplyRecipeColumn COL
ST
OK

5.10.7 ST Language

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

5.11 AS-interface Functions

These functions enable special operation on AS-i networks:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASiReadPI</td>
<td>Read actual parameters of an AS-i slave.</td>
</tr>
<tr>
<td>ASiReadPP</td>
<td>Read permanent parameters of an AS-i slave.</td>
</tr>
<tr>
<td>ASiSendParam</td>
<td>Send parameters to an AS-i slave.</td>
</tr>
<tr>
<td>ASiStorePI</td>
<td>Store actual parameters as permanent parameters.</td>
</tr>
<tr>
<td>ASiWritePP</td>
<td>Write permanent parameters of an AS-i slave.</td>
</tr>
</tbody>
</table>

**IMPORTANT**

AS-i networking may be not available on some targets. See the OEM instructions for more information.

5.11.1 Interface

Params := ASiReadPP (Master, Slave);
bOK := ASiWritePP (Master, Slave, Params);
bOK := ASiSendParam (Master, Slave, Params);
Params := ASiReadPI (Master, Slave);
bOK := ASiStorePI (Master);

5.11.2 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).
5.12  **average / averageL**

![PLCopen](https://kollmorgen.com)

**Function Block** - Calculates the average of signal samples.

### 5.12.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Enabling command.</td>
</tr>
<tr>
<td>XIN</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>Input signal.</td>
</tr>
<tr>
<td>N</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Number of samples stored for average calculation. Cannot exceed 128.</td>
</tr>
</tbody>
</table>

### 5.12.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOUT</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Average of the stored samples (*).</td>
</tr>
</tbody>
</table>

(*) averageL has LREAL arguments.

### 5.12.3 Remarks

- Average is calculated according to the number of stored samples.
  - This can be less than N when the block is enabled.
    - The "N" input (or the number of samples) is taken into account **only** when the RUN input is FALSE.
  - By default, the number of samples is 128.
- The "RUN" must be reset after a change in the number of samples.
  - Cycle the RUN input when you first call this function; this clears the default.

### 5.12.4 FBD Language

```
RUN XIN N
    AVERAGE
    XOUT
```

### 5.12.5 FFLD Language

- In the FFLD language, the input rung is the RUN command.
  - The output rung keeps the state of the input rung.
  - ENO has the same value as RUN.
5.12.6 IL Language

(* MyAve is a declared instance of AVERAGE function block *)
Op1: CAL MyAve (RUN, XIN, N)
     FFLD MyAve.XOUT
     ST XOUT

5.12.7 ST Language

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

See Also
- derivate
- hyster
- integral
- lim_alarm
- stackint

5.13 CurveLin

Function Block - Linear interpolation on a curve.

5.13.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>X coordinate of the point to be interpolated.</td>
</tr>
<tr>
<td>XAxis</td>
<td>REAL[]</td>
<td></td>
<td></td>
<td></td>
<td>X coordinates of the known points of the X axis.</td>
</tr>
<tr>
<td>YVal</td>
<td>REAL[]</td>
<td></td>
<td></td>
<td></td>
<td>Y coordinate of the points defined on the X axis.</td>
</tr>
</tbody>
</table>

5.13.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR</td>
<td>DINT</td>
<td></td>
<td></td>
<td>• Error code if failed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 0 (zero) if OK.</td>
</tr>
<tr>
<td>OK</td>
<td>BOOL</td>
<td>TRUE</td>
<td></td>
<td>TRUE if successful.</td>
</tr>
<tr>
<td>Y</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Interpolated Y value corresponding to the X input.</td>
</tr>
</tbody>
</table>

5.13.3 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.
- 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.
- An error is reported.

The ERR output gives the cause of the error if the function fails:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>1</td>
<td>Invalid dimension of input arrays</td>
</tr>
<tr>
<td>2</td>
<td>Invalid points for the X axis</td>
</tr>
<tr>
<td>4</td>
<td>X is out of the defined X axis</td>
</tr>
</tbody>
</table>

5.14 derivate

Function Block - Computes the derivative of a signal with respect to time.

5.14.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td></td>
<td></td>
<td>Run command:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• TRUE=derivate.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• FALSE=hold.</td>
</tr>
<tr>
<td>XIN</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>Input signal.</td>
</tr>
<tr>
<td>CYCLE</td>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
<td>Sampling period.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Must not be less than the target cycle timing.</td>
</tr>
</tbody>
</table>

5.14.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOUT</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Output signal.</td>
</tr>
</tbody>
</table>
5.14.3 Remarks
- 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.

5.14.4 FBD Language
```
RUN  DERIVATE  XOUT
XIN
CYCLE
```

5.14.5 FFLD Language
- In the FFLD language, the input rung is the RUN command.
  - The output rung keeps the state of the input rung.
  - ENO has the same state as RUN.
```
RUN       DERIVATE  Q
[ ______ ] [ ______ ]
XIN
CYCLE  XOUT
```

5.14.6 IL Language
```
(* MyDerv is a declared instance of DERIVATE function block *)
Op1: CAL MyDerv (RUN, XIN, CYCLE)
FFLD MyDerv.XOUT
ST XOUT
```

5.14.7 ST Language
```
(* MyDerv is a declared instance of DERIVATE function block. *)
MyDerv (RUN, XIN, CYCLE);
XOUT := MyDerv.XOUT;
```

See Also
- average / averageL
- hyster
- integral
- lim_alarm
- stackint

5.15 EnableEvents

PLCopen ✔
**Function** - Enable or disable the production of events for binding (runtime to runtime variable exchange).

### 5.15.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>BOOL</td>
<td>l</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE to enable events.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>l</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE to disable events.</td>
</tr>
</tbody>
</table>

### 5.15.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENO</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Echo of EN input.</td>
</tr>
</tbody>
</table>

### 5.15.3 Remarks

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

### 5.15.4 FBD Language

```
EN  EnableEvents  ENO
```

### 5.15.5 FFLD Language

- In FFLD language, the input rung (EN) enables the event production.
  - The output rung keeps the state of the input rung.
  - Events are enables if EN is TRUE.
  - ENO has the same value as EN.

```
EN  EnableEvents  ENO
```

### 5.15.6 IL Language

- In the IL language, the first input must be loaded before the function call.

```
Op1:  LD  EN
      EnableEvents
      ST  ENO
```

### 5.15.7 ST Language

```
ENO := EnableEvents (EN);
```

See Also
5.16 FIFO

Function Block - Manages a first in / first out list.

5.16.1 Inputs

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Tail</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Value of the oldest pushed value - updated after call!</td>
</tr>
<tr>
<td>Buf[]</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Array for storing values.</td>
</tr>
<tr>
<td>IN</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
<td>Value to be pushed.</td>
</tr>
<tr>
<td>POP</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Pop a new value (on rising edge).</td>
</tr>
<tr>
<td>PUSH</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Push a new value (on rising edge).</td>
</tr>
<tr>
<td>RST</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Reset the list.</td>
</tr>
</tbody>
</table>

5.16.2 Outputs

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPTY</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE if the list is empty.</td>
</tr>
<tr>
<td>OFLO</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE if overflow on a PUSH command.</td>
</tr>
<tr>
<td>Count</td>
<td>DINT</td>
<td></td>
<td></td>
<td>Number of values in the list.</td>
</tr>
<tr>
<td>pRead</td>
<td>DINT</td>
<td></td>
<td></td>
<td>Index in the buffer of the oldest pushed value.</td>
</tr>
<tr>
<td>pWrite</td>
<td>DINT</td>
<td></td>
<td></td>
<td>Index in the buffer of the next push position.</td>
</tr>
</tbody>
</table>

5.16.3 Remarks

- IN, @Tail and Buf[] must have the same data type.
  - It cannot be STRING.
- The @Tail argument specifies a variable 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 an instance of the FIFO function block is called, that instance stores which array is passed to BUF[].
- If a later call to the same instance passes a different array for the BUF[] argument, the call is considered invalid and no action is performed.
- In this instance, the EMPTY output returns TRUE.

5.16.4 FBD Language

![FBD Diagram]

5.16.5 FFLD Language

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

5.16.6 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

5.16.7 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;
See Also
LIFO

5.17  File Management

File Management functions provide the ability to:

- Read machine recipes or other machine operational data into the KAS program from the SD card, USB flash drive, or a shared directory.
- Read cam tables into the program from the SD card, USB flash drive, or a shared directory.
- Store machine operational data in internal PxMM or PCMM2G flash memory (retrievable through the web server), the SD card, USB flash drive, or a shared directory.

**NOTE**
A shared directory connection is setup through the web server.

**TIP**
- Functions to parse out information from a file using a string format can be found in "String Operations" (→ p. 200).
- If the file is in a .CSV format, these functions can be used: "LogFileCSV" (→ p. 255), "ApplyRecipeColumn" (→ p. 228).
- You can create, store, and retrieve recipes and other data using either:
  - the AKI Terminals. For more information see the KVB manual.
  - an external bus connection to the PxMM or PCMM2G with a supported fieldbus (e.g., UDP or HTTP).

These function blocks enable sequential read / write operations in disk files:

<table>
<thead>
<tr>
<th>Function Block</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileClose</td>
<td>Close an open file.</td>
</tr>
<tr>
<td>FileCopy</td>
<td>Copy a file.</td>
</tr>
<tr>
<td>FileDelete</td>
<td>Remove a file.</td>
</tr>
<tr>
<td>FileEOF</td>
<td>Test if the end of the file is reached in a file that is open for reading.</td>
</tr>
<tr>
<td>FileExists</td>
<td>Test if a file exists.</td>
</tr>
<tr>
<td>FileOpenA</td>
<td>Create or open a file in append mode.</td>
</tr>
<tr>
<td>FileOpenR</td>
<td>Open a file for reading.</td>
</tr>
<tr>
<td>FileOpenW</td>
<td>Create or reset a file and open it for writing.</td>
</tr>
<tr>
<td>FileReadBinData</td>
<td>Read binary data from a file.</td>
</tr>
<tr>
<td>FileReadLine</td>
<td>Read a string value from a text file.</td>
</tr>
<tr>
<td>FileRename</td>
<td>Rename a file.</td>
</tr>
<tr>
<td>FileSeek</td>
<td>Set the current position of a file.</td>
</tr>
<tr>
<td>FileSize</td>
<td>Get the size of a file.</td>
</tr>
<tr>
<td>FileWriteBinData</td>
<td>Write binary data to a file.</td>
</tr>
<tr>
<td>FileWriteLine</td>
<td>Write a string value to a text file.</td>
</tr>
</tbody>
</table>

These functions handle mounting of SD cards:
### Name | Use
---|---
"SD_ISREADY" (→ p. 242) | Check that the SD card is ready for read/write
"SD_MOUNT" (→ p. 241) | Mount an SD card
"SD_UNMOUNT" (→ p. 241) | Unmount an SD card

Each file is identified in the application by a unique handle manipulated as a DINT value.

- The file handles are allocated by the target system.
- Handles are returned by the Open function blocks and used by all other function blocks for identifying the file.

**Related Function Blocks**

LogFileCSV  log values of variables to a CSV file

#### IMPORTANT

- Files are opened and closed directly by the Operating System of the target.
  - Opening some files can be dangerous for system safety and integrity.
  - The number of open files (from FileOpenA, FileOpenR, and FileOpenW) is limited by the resources available on the target system.

- Ensure that each file successfully opened using FileOpenA, FileOpenR, and FileOpenW.
  - The FileOpenW has a corresponding FileClose to close the file.
  - Closing the file will release the file ID, making it available for operations on other files.

#### NOTE

- Opening a file with FileOpenA, FileOpenR, and FileOpenW can be unsuccessful (invalid path or file name, too many open files.)
  - Your application must check the file ID for a NULL value.
  - If the file ID is NULL (zero), then file read or write operations will fail.

- File management may be unavailable on some targets.
- Memory on the SD card is available in addition to the existing flash memory.
- Valid paths for storing files depend on the target implementation.
- Error messages are logged in the Controller log section of KAS Runtime where there is a failure in any related function block.
- Using the KAS Simulator, all path names 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.
- PCMM2G files are little endian.

#### TIP

Review the "File Path Conventions" (→ p. 243) to understand hardware-based functional differences.

### 5.17.1 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.

Use an SD card on the controller:

1. Verify the SD card is inserted.
2. Mount the card using "SD_MOUNT" (→ p. 241).
3. Verify the card is accessible using "SD_ISREADY" (→ p. 242) before performing a read or write action.
4. Unmount the card, using "SD_UNMOUNT" (→ p. 241) after performing read/write actions.
SD Card is not supported by PCMM2G.

Recommended: Stop all motion before using SD_MOUNT and SD_UNMOUNT.

5.17.2 SD Card Mounting Functions

Function - These functions handle mounting of an SD card:

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SD_MOUNT&quot; (➔ p. 241)</td>
<td>Mount an SD card.</td>
</tr>
<tr>
<td>&quot;SD_UNMOUNT&quot; (➔ p. 241)</td>
<td>Unmount an SD card.</td>
</tr>
<tr>
<td>&quot;SD_ISREADY&quot; (➔ p. 242)</td>
<td>Verify the SD card is ready for read/write.</td>
</tr>
</tbody>
</table>

5.17.2.1 SD_MOUNT

Mount the SD Card.

Recommended: Stop all motion before using SD_MOUNT.

<table>
<thead>
<tr>
<th>Device</th>
<th>Action</th>
<th>Return Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKD PDMM</td>
<td>Mount the SD Card.</td>
<td>If the mount is successful, the return value is TRUE.</td>
<td>OK := SD_MOUNT();</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the mount is not successful, the return value is FALSE.</td>
<td>OK : BOOL TRUE if mounting the SD Card is successful.</td>
</tr>
<tr>
<td>PCMM2G</td>
<td>SD Card is not supported by PCMM2G.</td>
<td>It always returns FALSE.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This does not perform any action.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulator</td>
<td>This does not perform any action.</td>
<td>It always returns TRUE.</td>
<td></td>
</tr>
</tbody>
</table>

5.17.2.2 SD_UNMOUNT

Unmount the SD Card.

Recommended: Stop all motion before using SD_UNMOUNT.
### 5.17.2.3 SD_ISREADY

<table>
<thead>
<tr>
<th>Device</th>
<th>Action</th>
<th>Return Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKD PDMM</td>
<td>Unmount the SD Card.</td>
<td>If the unmount is successful, the return value is TRUE.</td>
<td>OK := SD_UNMOUNT();</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the unmount is not successful, the return value is FALSE.</td>
<td>OK : BOOL TRUE if unmounting the SD Card is successful.</td>
</tr>
<tr>
<td>PCMM2G</td>
<td>SD Card is not supported by PCMM2G. This does not perform any action.</td>
<td>It always returns FALSE.</td>
<td></td>
</tr>
<tr>
<td>Simulator</td>
<td>This does not perform any action.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device</th>
<th>Action</th>
<th>Return Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKD PDM M</td>
<td>Verify the SD Card is mounted in the AKD PDMM.</td>
<td>If the SD Card is mounted, the return value is TRUE.</td>
<td>OK := SD_ISREADY();</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the SD Card is not mounted, the return value is FALSE.</td>
<td>OK : BOOL TRUE if the SD Card is mounted.</td>
</tr>
<tr>
<td>PCMM2G</td>
<td>SD Card is not supported by PCMM2G. This does not perform any action.</td>
<td>It always returns FALSE.</td>
<td></td>
</tr>
</tbody>
</table>
5.17.3 File Path Conventions

Depending on the system used, paths to file locations may be defined as either:

- **absolute** (`C://dir1/file1`)
- **relative paths** (`/dir1/file1`)

Not all systems handle all options. The paths vary depending upon the system.

<table>
<thead>
<tr>
<th>System</th>
<th>Absolute Paths</th>
<th>Relative Paths</th>
<th>Handling of Directories</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKD PDMM</td>
<td>X</td>
<td>✓</td>
<td>There is no support for creating directories on the controller. Any path provided to the function blocks (e.g., <code>file1</code>) is appended to the default user data folder. User Data Folders: PCMM &amp; AKD PDMM: <code>/mount/flash/userdata/</code> PCMM2G: <code>/home/kas/kas/userdata/</code></td>
</tr>
<tr>
<td>PCMM</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>PCMM2G</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Simulator</td>
<td>✓</td>
<td>✓</td>
<td>When a relative path is provided to the function blocks, the path is appended to the default user data folder: <code>&lt;User Directory&gt;/Kollmorgen/Kollmorgen Automation Suite/SinopeSimulator/Application/userdata/</code></td>
</tr>
</tbody>
</table>

See Also

- "File Name Warning and Limitations" (→ p. 243)
- "SD Card Path Conventions" (→ p. 245)
- "Shared Directory Path Conventions" (→ p. 244)
- "USB Flash Drive Path Conventions" (→ p. 245)

5.17.3.1 File Name Warning and Limitations
File names in the controller's flash storage are case-sensitive.
The SD card or USB flash drive (FAT16 or FAT32) are NOT case-sensitive.

<table>
<thead>
<tr>
<th>Storage</th>
<th>File System</th>
<th>Case-Sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedded flash: AKD PDMM / PCMM / PCMM2G</td>
<td>FFS3 (POSIX-like)</td>
<td>Yes</td>
</tr>
<tr>
<td>SD card / USB flash drive: AKD PDMM / PCMM</td>
<td>FAT16 or FAT32</td>
<td>No</td>
</tr>
</tbody>
</table>

**Example**

- Two files (MyFile.txt and myfile.txt) can exist in the same directory of the controller's flash.
  - They cannot exist in the same directory on the controller'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 controller's flash to the SD card or USB flash drive, one of the files is lost.

**IMPORTANT**

Use unique file names to prevent conflicts and to keep the application compatible across all platforms.
Do not rely on case-sensitive file names.

**See Also**

- "SD Card Path Conventions" (⇒ p. 245)
- "Shared Directory Path Conventions" (⇒ p. 244)
- "USB Flash Drive Path Conventions" (⇒ p. 245)

**5.17.3.2 Shared Directory Path Conventions**

The AKD PDMM, PCMM, and PCMM2G support access to a shared directory on a remote computer.

To access files in a shared directory from the AKD PDMM, PCMM, and PCMM2G use
/mount/shared at the beginning of the path, before the shared directory's relative path and file name:

```
/mount/shared/directory/filename
```

<table>
<thead>
<tr>
<th>Valid Paths</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>/mount/shared</td>
<td></td>
</tr>
<tr>
<td>mount/shared</td>
<td></td>
</tr>
<tr>
<td>/mount/shared</td>
<td></td>
</tr>
<tr>
<td>mount/shared</td>
<td></td>
</tr>
<tr>
<td>mount/shared</td>
<td></td>
</tr>
</tbody>
</table>

- The path is not case sensitive.
- The /MOUNT/SHARED, MOUNT/SHARED/, etc. are also valid.

**Example 1**

Opening the file example.txt from a shared directory on a remote computer.

```
fileID := Inst_FileOpenA(TRUE,'/mount/shared/example.txt');
```

**Example 2**

Opening the file myfiles/example.txt from a shared directory on a remote computer.

```
fileID := Inst_FileOpenA(TRUE,'/mount/shared/myfiles/example.txt');
```
5.17.3.3  SD Card Path Conventions
Access to the SD card memory requires that a valid SD card label 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 /.
- This path label is case insensitive.

The SDCard folder is created inside the userdata folder to maintain compatibility with the Simulator.

File access points to userdata/SDCard when a AKD PDMM SDCard path is used on the Simulator.

5.17.3.3.1  Valid Paths

<table>
<thead>
<tr>
<th>Valid Paths</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>//SDCard/file1</td>
<td></td>
</tr>
<tr>
<td>$dcard/dir1/file1</td>
<td>dir1 must have been already created.</td>
</tr>
<tr>
<td>/sdcard/dir1/file1</td>
<td>dir1 must have been already created.</td>
</tr>
<tr>
<td>//sdCard\file1</td>
<td></td>
</tr>
</tbody>
</table>

5.17.3.3.2  Invalid Paths

<table>
<thead>
<tr>
<th>Invalid Paths</th>
<th>Invalid Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>///SDCard/file1</td>
<td>Started with more than two forward or two backward slashes.</td>
</tr>
<tr>
<td>/\Sdcard/dir1/file1</td>
<td>Started with one forward and one backward slash.</td>
</tr>
<tr>
<td>/sdcarddir1/file1</td>
<td>No forward or backward slash.</td>
</tr>
<tr>
<td>/sdcard1/dir1/file1</td>
<td>Invalid label.</td>
</tr>
</tbody>
</table>

See Also

- "File Name Warning and Limitations" (→ p. 243)
- "SD Card Path Conventions" (→ p. 245)
- "USB Flash Drive Path Conventions" (→ p. 245)

5.17.3.4  USB Flash Drive Path Conventions
Access to the USB flash drive memory requires that a valid USB flash drive label be used at the beginning of the path, followed by the relative path to the USB flash drive.

(Valid USB Flash Drive Label)/(Relative Path)
• A valid USB flash drive relative path starts with //, /, \, or \\.
• This is immediately followed by usbf lash which is followed by \ or /.
• This path label is case insensitive.

The usbf lash folder is created inside the userdata folder to maintain compatibility with the Simulator.

File access points to userdata/usbf lash when a PCMM2G usbf lash path is used on the Simulator.

### 5.17.3.4.0.1 Valid Paths

<table>
<thead>
<tr>
<th>Valid Paths</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>/usbflash/file1</td>
<td></td>
</tr>
<tr>
<td>\usbflash/dir1/file1</td>
<td>dir1 must have been already created.</td>
</tr>
<tr>
<td>/usbflash/dir1/file1</td>
<td>dir1 must have been already created.</td>
</tr>
<tr>
<td>/usbflash/file1</td>
<td></td>
</tr>
</tbody>
</table>

### 5.17.3.4.0.2 Invalid Paths

<table>
<thead>
<tr>
<th>Invalid Paths</th>
<th>Invalid Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>//usbflash/file1</td>
<td>Started with more than two forward or two backward slashes.</td>
</tr>
<tr>
<td>/usbflash/dir1/file1</td>
<td>Started with one forward and one backward slash.</td>
</tr>
<tr>
<td>/usbflash/dir1/file1</td>
<td>No forward or backward slash.</td>
</tr>
<tr>
<td>/usbflash1/dir1/file1</td>
<td>Invalid label.</td>
</tr>
</tbody>
</table>

See Also

- "File Name Warning and Limitations" (➔ p. 243)
- "SD Card Path Conventions" (➔ p. 245)
- "Shared Directory Path Conventions" (➔ p. 244)

### 5.18 FilterOrder1

#### Function Block
- First order filter.

#### 5.18.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XIN</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>Input analog value.</td>
</tr>
<tr>
<td>GAIN</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>Transformation gain.</td>
</tr>
</tbody>
</table>

#### 5.18.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOUT</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Output signal.</td>
</tr>
</tbody>
</table>

#### 5.18.3 Remarks
The operation performed is:

\[
\text{Output} = (\text{Input} \times \text{Gain}) + (\text{OutputPrev} \times (1-\text{Gain}))
\]

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

### 5.18.4 Example

![Diagram of FilterOrder1 function block]

### 5.18.5 FBD Language

Not available.

### 5.18.6 FFLD Language

Not available.

### 5.18.7 IL Language

Not available.

### 5.18.8 ST Language

Filt1 is a declared instance of FilterOrder1 function block.

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

### 5.19 GetSysInfo

**Function** - Get system information.

#### 5.19.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO</td>
<td>DINT</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Identifier of the requested information.</td>
</tr>
</tbody>
</table>

#### 5.19.2 Outputs
Output | Data Type | Range | Unit | Description
---|---|---|---|---
Q | DINT | | | Value of the requested information or 0 (zero) if error.

### 5.19.3 Remarks
The INFO parameter can be one of these predefined values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SYSINFO_APPSTAMP</td>
<td>Compiling date stamp of the application.</td>
</tr>
<tr>
<td>_SYSINFO_BIGENDIAN</td>
<td>Non zero if the runtime processor is big endian.</td>
</tr>
<tr>
<td>_SYSINFO_CHANGE_CYCLE</td>
<td>Indicates a cycle just after an Online Change</td>
</tr>
<tr>
<td>_SYSINFO_CODECRC</td>
<td>CRC of the application code.</td>
</tr>
<tr>
<td>_SYSINFO_CYCLECOUNT</td>
<td>Counter of cycles.</td>
</tr>
<tr>
<td>_SYSINFO_CYCLEMAX_MICROS</td>
<td>Maximum detected cycle time in micro-seconds.</td>
</tr>
<tr>
<td>_SYSINFO_CYCLEMAX_MS</td>
<td>Maximum detected cycle time in milliseconds.</td>
</tr>
<tr>
<td>_SYSINFO_CYCLEOVERFLOWS</td>
<td>Number of detected cycle time overflows.</td>
</tr>
<tr>
<td>_SYSINFO_CYCLESTAMP_MS</td>
<td>Timestamp of the current cycle in milliseconds (OEM dependent).</td>
</tr>
<tr>
<td>_SYSINFO_CYCLETIME_MICROS</td>
<td>Duration of the previous cycle in micro-seconds.</td>
</tr>
<tr>
<td>_SYSINFO_CYCLETIME_MS</td>
<td>Duration of the previous cycle in milliseconds.</td>
</tr>
<tr>
<td>_SYSINFO_DATACRC</td>
<td>CRC of the application symbols.</td>
</tr>
<tr>
<td>_SYSINFO_DBSIZE</td>
<td>Space used in RAM (bytes).</td>
</tr>
<tr>
<td>_SYSINFO_DEMOAPP</td>
<td>Non zero if the application was compiled in DEMO mode.</td>
</tr>
<tr>
<td>_SYSINFO_ELAPSED</td>
<td>Seconds elapsed since startup.</td>
</tr>
<tr>
<td>_SYSINFO_FREEHEAP</td>
<td>Available space in memory heap (bytes).</td>
</tr>
<tr>
<td>_SYSINFO_NBBREAKPOINTS</td>
<td>Number of installed breakpoints.</td>
</tr>
<tr>
<td>_SYSINFO_NBLOCKED</td>
<td>Number of locked variables.</td>
</tr>
<tr>
<td>_SYSINFO_TRIGGER_MICROS</td>
<td>Programmed cycle time in micro-seconds.</td>
</tr>
<tr>
<td>_SYSINFO_TRIGGER_MS</td>
<td>Programmed cycle time in milliseconds.</td>
</tr>
<tr>
<td>_SYSINFO_WARMSTART</td>
<td>Non zero if RETAIN variables were loaded at the last start.</td>
</tr>
</tbody>
</table>

### 5.19.4 FBD Language

```
Info GetSysInfo Q
```

### 5.19.5 FFLD Language
- In the 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.
- The function is executed only if EN is TRUE.
- ENO keeps the same value as EN.
5.19.6 IL Language

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

```
Op1: LD  INFO
     GETSYSINFO
     ST  Q
```

5.19.7 ST Language

```
Q := GETSYSINFO (INFO);
```

5.20  hyster

**Function Block** - Hysteresis detection.

5.20.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XIN1</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>First input.</td>
</tr>
<tr>
<td>XIN2</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>Second input.</td>
</tr>
<tr>
<td>EPS</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>Hysteresis.</td>
</tr>
</tbody>
</table>

5.20.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Detected hysteresis: TRUE if XIN1 becomes greater than XIN2+EPS and is not yet below XIN2-EPS.</td>
</tr>
</tbody>
</table>

5.20.3 Remarks

- The hysteresis is detected on the difference of XIN1 and XIN2 signals.

5.20.4 FBD Language
5.20.5 FFLD Language

- In the FFLD language, the input rung (EN) is used for enabling the block.
  - The output rung is the Q output.
  - The block is not called if EN is FALSE.

5.20.6 IL Language

```IL
(* MyHyst is a declared instance of HYSTER function block *)
Op1: CAL MyHyst (XIN1, XIN2, EPS)
FFLD MyHyst.Q
ST Q
```

5.20.7 ST Language

```ST
(* MyHyst is a declared instance of HYSTER function block. *)
MyHyst (XIN1, XIN2, EPS);
Q := MyHyst.Q;
```

See Also

- average / averageL
- derivate
- integral
- lim_alarm
- stackint

5.21 integral

**PLCopen**

*Function Block* - Calculates the integral of a signal with respect to time.

5.21.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CYCLE</td>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
<td>Sampling period. Must not be less than the target cycle timing.</td>
</tr>
</tbody>
</table>
### Input

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Overriding reset.</td>
</tr>
<tr>
<td>RUN</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Run command:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TRUE = integrate.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>FALSE = hold.</td>
<td></td>
</tr>
<tr>
<td>X0</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>Initial value.</td>
</tr>
<tr>
<td>XIN</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>Input signal.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td></td>
<td></td>
<td>Running mode report: NOT (R1).</td>
</tr>
<tr>
<td>XOUT</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Output signal.</td>
</tr>
</tbody>
</table>

### Remarks

- 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.

### FBD Language

![FBD Diagram]

### FFLD Language

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

### IL Language

The IL language is not specified.
5.21.7 ST Language

(* MyIntg is a declared instance of INTEGRAL function block. *)
 MyIntg (RUN, R1, XIN, X0, CYCLE);
 Q := MyIntg.Q;
 XOUT := MyIntg.XOUT;

See Also
- "average / averageL" (→ p. 232)
- "derivate" (→ p. 234)
- "hyster" (→ p. 249)
- "lim_alarm" (→ p. 254)
- stackint

5.22 LIFO

*PLCopen*

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

5.22.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH</td>
<td>BOOL</td>
<td>Push a new value (on rising edge).</td>
</tr>
<tr>
<td>POP</td>
<td>BOOL</td>
<td>Pop a new value (on rising edge).</td>
</tr>
<tr>
<td>RST</td>
<td>BOOL</td>
<td>Reset the list.</td>
</tr>
<tr>
<td>NEXTIN</td>
<td>ANY</td>
<td>Value to be pushed.</td>
</tr>
<tr>
<td>NEXTOUT</td>
<td>ANY</td>
<td>Value at the top of the stack - updated after call.</td>
</tr>
<tr>
<td>BUFFER</td>
<td>ANY</td>
<td>Array for storing values.</td>
</tr>
</tbody>
</table>

5.22.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPTY</td>
<td>BOOL</td>
<td>TRUE if the stack is empty.</td>
</tr>
<tr>
<td>OFLO</td>
<td>BOOL</td>
<td>TRUE if overflow on a PUSH command.</td>
</tr>
<tr>
<td>COUNT</td>
<td>DINT</td>
<td>Number of values in the stack.</td>
</tr>
<tr>
<td>PREAD</td>
<td>DINT</td>
<td>Index in the buffer of the top of the stack.</td>
</tr>
<tr>
<td>PWRITE</td>
<td>DINT</td>
<td>Index in the buffer of the next push position.</td>
</tr>
</tbody>
</table>

5.22.3 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 an instance of the LIFO function block is called, that instance will store which array is passed to BUFFER. If a later call to the same instance passes a different array for the BUFFER argument, the call is considered invalid and no action is performed. The EMPTY output returns TRUE in this case.

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

### 5.22.4 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;
```

### 5.22.5 FBD Language

```
LIFO
PUSH
POP
RST
NEXTIN
NEXTOUT
BUFFER
```

### 5.22.6 FFLD Language

```
PUSH
[       ]
|       |
POP
RST
NEXTIN
NEXTOUT
BUFFER
```

### 5.22.7 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
```
lim_alarm

**Function Block** - Detects high and low limits of a signal with hysteresis.

### 5.23.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPS</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>Value of the hysteresis.</td>
</tr>
<tr>
<td>H</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>Value of the high limit.</td>
</tr>
<tr>
<td>L</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>Value of the low limit.</td>
</tr>
<tr>
<td>X</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>Input signal.</td>
</tr>
</tbody>
</table>

### 5.23.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE if the signal exceeds one of the limits. Equals to QH OR QL.</td>
</tr>
<tr>
<td>QH</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE if the signal exceeds the high limit.</td>
</tr>
<tr>
<td>QL</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE if the signal exceeds the low limit.</td>
</tr>
</tbody>
</table>

### 5.23.3 Remarks

### 5.23.4 FBD Language
5.23.5 FFLD Language

- In the FFLD language, the input rung (EN) is used for enabling the block.
  - The output rung is the QH output.
  - The block is not called if EN is FALSE.

5.23.6 IL Language

```plaintext
(* MyAlarm is a declared instance of LIM_ALRM function block *)
Op1: CAL MyAlarm (H, X, L, EPS)
    FFLD MyAlarm.QH
    ST QH
    FFLD MyAlarm.Q
    ST Q
    FFLD MyAlarm.QL
    ST QL
```

5.23.7 ST Language

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

See Also
- Alarm_A
- Alarm_M

5.24 LogFileCSV

**PLCopen ✔**

**Function Block** - Create a log file in CSV format for a list of variables.

5.24.1 Inputs
### 5.24.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE if the requested operation has been performed without error.</td>
</tr>
<tr>
<td>ERR</td>
<td>DINT</td>
<td></td>
<td></td>
<td>Error report for the last requested operation. 0 (zero) is OK.</td>
</tr>
</tbody>
</table>

### 5.24.3 Remarks

**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.
  - See the OEM instructions for more information about available features.
- Valid paths for storing files depend on the target implementation.
  - See the OEM instructions for more information about available paths.
- 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.

This example shows a list and a program that log values everyday at 14h23m (2:23 pm) (see call out 1).

5.24.4 FBD Language

5.24.5 FFLD Language

5.24.6 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

5.24.7 ST Language

(* MyLOG is a declared instance of LogFileCSV function block *)
MyLOG (b_LOG, RST, LIST, PATH);
:= MyLOG.Q;
R := MyLog.ERR;

See Also
VLID

5.25 PID

**Function Block** - PID loop

### 5.25.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTO</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE = normal mode&lt;br&gt;FALSE = manual mode</td>
<td></td>
</tr>
<tr>
<td>DEADB.ERR</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Hysteresis on PV. PV is considered as unchanged if it is both:&lt;br&gt;Greater than (PVprev - DEADBAND_W).&lt;br&gt;Less than (PRprev + DEADBAND_W).</td>
<td></td>
</tr>
<tr>
<td>FFD</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Disturbance value on output.</td>
<td></td>
</tr>
<tr>
<td>I_ITL_ON</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>If TRUE, the integrated value is reset to I_ITLVAL.</td>
<td></td>
</tr>
<tr>
<td>I_ITLVAL</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Reset value for integration when I_ITL_ON is TRUE.</td>
<td></td>
</tr>
<tr>
<td>I_SEL</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>If FALSE, the integrated value is ignored.</td>
<td></td>
</tr>
<tr>
<td>INT_HOLD</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>If TRUE, the integrated value is frozen.</td>
<td></td>
</tr>
<tr>
<td>KP</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Gain.</td>
<td></td>
</tr>
<tr>
<td>PV</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Process value.</td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Set point.</td>
<td></td>
</tr>
<tr>
<td>TD</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Derivation factor.</td>
<td></td>
</tr>
<tr>
<td>TI</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Integration factor.</td>
<td></td>
</tr>
<tr>
<td>TS</td>
<td>TIME</td>
<td></td>
<td></td>
<td>Sampling period.</td>
<td></td>
</tr>
<tr>
<td>XMAX</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Maximum output value.</td>
<td></td>
</tr>
<tr>
<td>XMIN</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Minimum allowed output value.</td>
<td></td>
</tr>
<tr>
<td>Xout_Manu</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Output value in manual mode.</td>
<td></td>
</tr>
</tbody>
</table>

### 5.25.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Last calculated error.</td>
</tr>
<tr>
<td>Xout</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Output command value.</td>
</tr>
</tbody>
</table>
### Output Data Type Range Unit Description

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xout_D</td>
<td>REAL</td>
<td>Last</td>
<td></td>
<td>calculated derivated value.</td>
</tr>
<tr>
<td>Xout_HLM</td>
<td>BOOL</td>
<td>TRUE</td>
<td></td>
<td>if the output value is saturated to XMAX.</td>
</tr>
<tr>
<td>Xout_I</td>
<td>REAL</td>
<td>Last</td>
<td></td>
<td>calculated integrated value.</td>
</tr>
<tr>
<td>Xout_LLM</td>
<td>BOOL</td>
<td>TRUE</td>
<td></td>
<td>if the output value is saturated to XMIN.</td>
</tr>
<tr>
<td>Xout_P</td>
<td>REAL</td>
<td>Last</td>
<td></td>
<td>calculated proportional value.</td>
</tr>
</tbody>
</table>

### 5.25.3 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 varies per the inputs provided upon further cycle executions.
- In the FFLD Language, the output rung has the same value as the AUTO input, corresponding to the input rung.

#### 5.25.3.1 Diagram

![PID Block Diagram](image)

#### 5.25.4 FBD Language
5.25.5 FFLD Language

(* ENO has the same state as the input rung. *)

5.25.6 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

5.25.7 ST Language
5.26 PWM

Function Block - Generate a PWM signal.

5.26.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XIN</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>Input analog value.</td>
</tr>
<tr>
<td>XinMin</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>Minimum input value.</td>
</tr>
<tr>
<td>XinMax</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>Maximum input value.</td>
</tr>
<tr>
<td>MinPulse</td>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
<td>Minimum pulse time on output.</td>
</tr>
<tr>
<td>Period</td>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
<td>Period of the output signal.</td>
</tr>
</tbody>
</table>

5.26.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default (.Q)</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Blinking PWM signal.</td>
</tr>
</tbody>
</table>

is this correct?
the original text here was just Q

5.26.3 Remarks

- The input value is truncated to [XinMin .. XinMax] interval.
  - XinMax must be greater than XinMin.
- The signal is TRUE during:

  \[(Xin - XinMin) \ast \text{Period} / (XinMax - XinMin)\]

5.26.4 FBD Language

```plaintext
(* 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;
```
5.26.5 FFLD Language
Not available. - IS THIS TRUE?

5.26.6 IL Language
Not available. - IS THIS TRUE?

5.26.7 ST Language
PWM1 is a declared instance of PWM function block.

```
PWM1 (rIn, rInMin, rInMax, tMinPulse, tPeriod);
Signal := PWM1.Q;
```

5.27 rand

**Function Block** - Returns a pseudo-random integer value between 0 (zero) and (base - 1).

### 5.27.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>DINT</td>
<td>1 to 2147483647</td>
<td>N/A</td>
<td>No default</td>
<td>The number of possible outcomes. Example: When base is 5, there are 5 possible outcomes: 0,1,2,3,4.</td>
</tr>
</tbody>
</table>

### 5.27.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Value</td>
<td>DINT</td>
<td>0 (zero) to (base - 1)</td>
<td>N/A</td>
<td>The generated pseudo-random number.</td>
</tr>
</tbody>
</table>
• `rand` uses a low-quality, but fast number generation algorithm.
  • It is sufficient for small bases and where security is not a concern.

• There is no way to seed the random number generator.
  • It is possible to receive the same pattern of generated numbers after the controller reboots.

### 5.27.4 FBD Language
Not available.

### 5.27.5 FFLD Language
Not available.

### 5.27.6 IL Language
Not available.

### 5.27.7 ST Language
```plaintext
dieValue := 1 + rand(6);
```

### 5.28 RAMP

**Function** - Limit the ascendance or descendance of a signal.

#### 5.28.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>REAL</td>
<td>Input signal.</td>
</tr>
<tr>
<td>ASC</td>
<td>REAL</td>
<td>Maximum ascendance during time base.</td>
</tr>
<tr>
<td>DSC</td>
<td>REAL</td>
<td>Maximum descendence during time base.</td>
</tr>
<tr>
<td>TM</td>
<td>TIME</td>
<td>Time base.</td>
</tr>
<tr>
<td>RST</td>
<td>BOOL</td>
<td>Reset.</td>
</tr>
</tbody>
</table>

#### 5.28.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT</td>
<td>REAL</td>
<td>Ramp signal.</td>
</tr>
</tbody>
</table>

#### 5.28.3 Time Diagram
5.28.4 Remarks

- Parameters are not updated constantly.
  - They are taken into account only when the:
    - first time the block is called.
    - 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 the 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.

5.28.5 ST Language

```plaintext
(* MyRamp is a declared instance of RAMP function block *)
MyRamp (IN, ASC, DSC, TM, RST);
OUT := MyBlinker.OUT;
```

5.28.6 FBD Language

```
Ramp
```

5.28.7 FFLD Language
The function is executed only if EN is TRUE
ENO keeps the same value as EN

5.28.8 IL Language

(* MyRamp is a declared instance of RAMP function block *)
Op1: CAL
MyRamp (IN, ASC, DSC, TM, RST)
FFLD MyBlinker.OUT
ST OUT

5.29 Real Time Clock Management Functions

5.29.1 Time Zone and Clock Synchronization
These function blocks configure the time zone and clock synchronization for the controller.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;DTGetNTPServer&quot; (→ p. 273)</td>
<td>Read the NTP server address.</td>
</tr>
<tr>
<td>&quot;DTGetNTPSync&quot; (→ p. 275)</td>
<td>Read the NTP synchronization enable state.</td>
</tr>
<tr>
<td>&quot;DTGetTimeZone&quot; (→ p. 277)</td>
<td>Read the Time Zone.</td>
</tr>
<tr>
<td>&quot;DTListTimeZones&quot; (→ p. 281)</td>
<td>List the time zones available on the controller.</td>
</tr>
<tr>
<td>&quot;DTSetDateTime&quot; (→ p. 285)</td>
<td>Sets the local date and time.</td>
</tr>
<tr>
<td>&quot;DTSetNTPServer&quot; (→ p. 288)</td>
<td>Set the NTP server address.</td>
</tr>
<tr>
<td>&quot;DTSetNTPSync&quot; (→ p. 289)</td>
<td>Set the NTP synchronization enable state.</td>
</tr>
<tr>
<td>&quot;DTSetTimeZone&quot; (→ p. 291)</td>
<td>Set the time zone.</td>
</tr>
</tbody>
</table>

5.29.2 Read the Real Time Clock
These functions read the real time clock of the target system:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;DTCurDate&quot; (→ p. 269)</td>
<td>Get the present date stamp.</td>
</tr>
<tr>
<td>&quot;DTCurDateTime&quot; (→ p. 270)</td>
<td>Get the present date and time stamp.</td>
</tr>
<tr>
<td>&quot;DTCurTime&quot; (→ p. 272)</td>
<td>Get the present time stamp.</td>
</tr>
</tbody>
</table>
### 5.29.3 Format the Present Date / Time
These functions format the present date/time to a string:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;day_time&quot;</td>
<td>Format the present date / time to a string.</td>
</tr>
<tr>
<td>&quot;DTFormat&quot;</td>
<td>Format the present date/time to a string with a custom format.</td>
</tr>
</tbody>
</table>

### 5.29.4 Triggering Operations
These functions are used for triggering operations:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;DTAt&quot;</td>
<td>Generate a pulse at designated time stamp (date and time).</td>
</tr>
<tr>
<td>&quot;DTEvery&quot;</td>
<td>Generate a pulse signal with long period.</td>
</tr>
</tbody>
</table>

**IMPORTANT**
- A real-time clock may not be available on all controller hardware models. See the controller hardware specifications for real-time clock availability.
- The AKD PDMM and PCMM reset the date and time when powered-on. The reset is to Jan 1, 1970 00:00:00. The elapsed time from device power-on can be determined from the Real Time Clock functions.
- PCMM2G does not reset the date and time when powered on.

### 5.29.5 day_time
- **Function** - Format the present date / time to a string.

#### 5.29.5.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEL</td>
<td>DINT</td>
<td>0 to 2</td>
<td>N/A</td>
<td>No default</td>
<td>Format string.</td>
</tr>
</tbody>
</table>

#### 5.29.5.2 Outputs
### Output | Data Type | Range | Unit | Description
--- | --- | --- | --- | ---
Q | STRING | No range | N/A | String containing formatted date or time.

**IMPORTANT**

PCMM generation 1 controllers do **not** have real-time clock hardware. PCMM2G does have re-time clock hardware. Real-time clock may not be available on all controller hardware models. See the controller hardware specifications for real-time clock availability.

#### 5.29.5.3 Remarks

Valid values of the SEL input are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (default)</td>
<td>Current date - format: YYYY/MM/DD.</td>
</tr>
<tr>
<td>1</td>
<td>Current time - format: HH:MM:SS.</td>
</tr>
<tr>
<td>2</td>
<td>Day of the week.</td>
</tr>
</tbody>
</table>

#### 5.29.5.4 FBD Language

```
  day_time
  SEL Q
```

#### 5.29.5.5 FFLD Language

- In the 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.

```
  day_time
  - En OK
  - SEL Q
```

#### 5.29.5.6 IL Language

```
Op1: LD SEL
    DAY_TIME
    ST Q
```

#### 5.29.5.7 ST Language

```
Q := DAY_TIME (SEL);
```

**See Also**

DTFormat

#### 5.29.6 DTAt
**Function Block** - Generate a pulse at designated time stamp (date and time).

### 5.29.6.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>DINT</td>
<td>1900 to 2200</td>
<td>Years</td>
<td>No default</td>
<td>Year of the time stamp (e.g., 2006).</td>
</tr>
<tr>
<td>Month</td>
<td>DINT</td>
<td>1 to 12</td>
<td>Months</td>
<td>No default</td>
<td>Month of the time stamp (1 = January).</td>
</tr>
<tr>
<td>Day</td>
<td>DINT</td>
<td>1 to 31</td>
<td>Days</td>
<td>No default</td>
<td>Day of the time stamp.</td>
</tr>
<tr>
<td>TmOfDay</td>
<td>TIME</td>
<td>0 to 86,399,999</td>
<td>Milliseconds</td>
<td>No default</td>
<td>Time of day of the time stamp.</td>
</tr>
<tr>
<td>RST</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>n/a</td>
<td>No default</td>
<td>Reset command.</td>
</tr>
</tbody>
</table>

### 5.29.6.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QAt</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>Pulse signal.</td>
</tr>
<tr>
<td>QPast</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>True if elapsed.</td>
</tr>
</tbody>
</table>

### 5.29.6.3 Remarks

**IMPORTANT**
The real-time clock may not be available on all controller hardware models. See the controller hardware specifications for real-time clock availability.

- 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.
    - The output QAT is set to TRUE for one cycle only (pulse signal).
- Highest units are ignored if set to 0.
  - Example: If arguments are year=0, month=0, day = 3, tmofday=t#10h, the block triggers on the next 3rd day of the month at 10h.

### 5.29.6.4 FBD Language

```
YEAR
MONTH
DAY
TMOFDAY
RST
```

```
DTAT
QAt
QPast
```
5.29.6.5 FFLD Language
In the FFLD language, the block is activated only if the input rung is TRUE.
(* Called only if EN is TRUE. *)

```
```

5.29.6.6 IL Language

```
(* MyDTAT is a declared instance of DTAT function block. *)
Op1: CAL
MyDTAT (YEAR, MONTH, DAY, TMOFDAY, RST)
FFLD MyDTAT.QAT
ST QAT
FFLD MyDTATA.QPAST
ST QPAST
```

5.29.6.7 ST Language

```
(* MyDTAT is a declared instance of DTAT function block. *)
MyDTAT (YEAR, MONTH, DAY, TMOFDAY, RST);
QAT := MyDTAT.QAT;
QPAST := MyDTATA.QPAST;
```

See Also
- DTEvery
- Real Time Clock Management Functions

5.29.7 DTCurDate

**Function** - Get the present date stamp.

5.29.7.1 Inputs
None

5.29.7.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td>No range</td>
<td>N/A</td>
<td>Numerical stamp representing the current date.</td>
</tr>
</tbody>
</table>

5.29.7.3 ST Language
Q := DTCurDate ();

See Also
- "DTDay" (p. 272)
- "DTMonth" (p. 284)
- "DTYear" (p. 293)

5.29.8 DTCurDateTime

Function Block - Get the present date and time stamp.

5.29.8.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| Local | BOOL      | TRUE, FALSE | N/A  | No default | • TRUE if local time is requested.  
                                         • FALSE if GMT is requested. |

5.29.8.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>DINT</td>
<td>1900 to 2200</td>
<td>Years</td>
<td>Present year.</td>
</tr>
<tr>
<td>Month</td>
<td>DINT</td>
<td>1 to 12</td>
<td>Months</td>
<td>Present month.</td>
</tr>
<tr>
<td>Day</td>
<td>DINT</td>
<td>1 to 31</td>
<td>Days</td>
<td>Present day.</td>
</tr>
<tr>
<td>Hour</td>
<td>DINT</td>
<td>0 to 23</td>
<td>Hours</td>
<td>Present time: hours.</td>
</tr>
<tr>
<td>Min</td>
<td>DINT</td>
<td>0 to 59</td>
<td>Minutes</td>
<td>Present time: minutes.</td>
</tr>
<tr>
<td>Sec</td>
<td>DINT</td>
<td>0 to 60</td>
<td>Seconds</td>
<td>Present time: seconds.</td>
</tr>
<tr>
<td>MSec</td>
<td>DINT</td>
<td>0 to 999</td>
<td>Milliseconds</td>
<td>Present time: milliseconds.</td>
</tr>
<tr>
<td>TmOfDay</td>
<td>TIME</td>
<td>0 to 86,399,999</td>
<td>Milliseconds</td>
<td>Present time of day (milliseconds since midnight).</td>
</tr>
</tbody>
</table>
| DST    | BOOL      | TRUE, FALSE | N/A       | Indicates if the time is in:  
                                         • Daylight saving time (DST = TRUE)  
                                         • Standard time (DST = FALSE) |

5.29.8.3 Remarks
None

5.29.8.4 FBD Language
5.29.8.5  FFLD Language

Not available.

5.29.8.6  IL Language

Not available.

5.29.8.7  ST Language

```
Inst_DTCurDateTime(useLocalTime);
localYear := Inst_DTCurDateTime.Year;
localMonth := Inst_DTCurDateTime.Month;
localDay := Inst_DTCurDateTime.Day;
localHour := Inst_DTCurDateTime.Hour;
localMin := Inst_DTCurDateTime.Min;
```
localSec := Inst_DTCurDateTime.Sec;
localMSec := Inst_DTCurDateTime.MSec;
localTmOfDay := Inst_DTCurDateTime.TmOfDay;
localDST := Inst_DTCurDateTime.DST;

5.29.9 DTCurTime

**Function** - Get the present time stamp.

5.29.9.1 Inputs
None

5.29.9.2 Output

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td>0 to 86,399,999</td>
<td>Milliseconds</td>
<td>Present milliseconds of the time.</td>
</tr>
</tbody>
</table>

5.29.9.3 ST Language

Q := DTCurTime ();

See Also
- "DTHour" (p. 281)
- "DTMin" (p. 283)
- "DTMs" (p. 284)
- "DTSec" (p. 285)

5.29.10 DTDay

**Function** - Get the day of the month from the date stamp.

5.29.10.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>DINT</td>
<td>No range</td>
<td>N/A</td>
<td>No default</td>
<td>Numerical stamp representing a date.</td>
</tr>
</tbody>
</table>

5.29.10.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td>1 to 31</td>
<td>N/A</td>
<td>Day of the month of the date.</td>
</tr>
</tbody>
</table>

5.29.10.3 ST Language

Q := DTDay (iDate);

See Also
5.29.11 DTGetNTPServer

Function Block - Read the NTP server address.
This function block is specific for PCMM2G only.

5.29.11.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>No default</td>
<td>If TRUE, request to read the NTP server address.</td>
</tr>
</tbody>
</table>

5.29.11.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>If TRUE, the command completed successfully.</td>
</tr>
<tr>
<td>Error</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>If TRUE, an error has occurred.</td>
</tr>
<tr>
<td>ErrorID</td>
<td>DINT</td>
<td>No range</td>
<td>N/A</td>
<td>Indicates the error if the Error output is set to TRUE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Error Codes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 23 = Internal error. See controller log for details.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 15000 = Controller type does not support this function block.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 16200 = Could not read NTP server configuration file.</td>
</tr>
<tr>
<td>NTPServer</td>
<td>STRING</td>
<td>No range</td>
<td>N/A</td>
<td>The address of the NTP server used for clock synchronization.</td>
</tr>
</tbody>
</table>

5.29.11.3 Remarks

None

5.29.11.4 FBD Language
5.29.11.5 FFLD Language

```plaintext
// read the NTP server address
Inst_DTGetNTPServer( bGetNTPServer );
if Inst_DTGetNTPServer.Done then
  bGetNTPServer := false;
  if NOT Inst_DTGetNTPServer.Error then
    NTPServer := Inst_DTGetNTPServer.NTPServer;
  else
    ErrorID := Inst_DTGetNTPServer.ErrorID;
  end_if;
end_if;
```

5.29.11.6 IL Language

Not available.

5.29.11.7 ST Language

See Also
5.29.12 DTGetNTPSync

**Function Block** - Read the NTP synchronization enable state.
This function block is specific for PCMM2G only.

### 5.29.12.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>No</td>
<td>If TRUE, request to read the synchronization enable state.</td>
</tr>
</tbody>
</table>

### 5.29.12.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>If TRUE, the command completed successfully.</td>
</tr>
<tr>
<td>Error</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>If TRUE, an error has occurred.</td>
</tr>
<tr>
<td>ErrorID</td>
<td>DINT</td>
<td>No range</td>
<td>N/A</td>
<td>Indicates the error if the Error output is set to TRUE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Error Codes</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 23 = Internal error. See controller log for details.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 15000 = Controller type does not support this function block.</td>
</tr>
<tr>
<td>SynchEn</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>The present NTP synchronization state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• TRUE = synchronization enabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• FALSE = synchronization disabled.</td>
</tr>
</tbody>
</table>

### 5.29.12.3 Remarks

None

### 5.29.12.4 FBD Language

- "DTCurDateTime" (→ p. 270)
- "DTGetNTPServer" (→ p. 273)
- "DTSetNTPServer" (→ p. 288)
- "DTSetNTPSync" (→ p. 289)
- "List of Date / Time / NTP ErrorID Codes" (→ p. 293)
5.29.12.5  FFLD Language

![Diagram of the FFLD Language](image)

5.29.12.6  IL Language

Not available.

5.29.12.7  ST Language

```plaintext
// read the NTP synchronization state
Inst_DTGetNTPSync( bGetNTPSync );
if Inst_DTGetNTPSync.Done then
  bGetNTPSync := false;

  if NOT Inst_DTGetNTPSync.Error then
    bNTPSyncEnable := Inst_DTGetNTPSync.SynchEn;
  else
    ErrorID := Inst_DTGetNTPSync.ErrorID;
  end_if;
end_if;
```

See Also
5.29.13 DTGetTimeZone

**Function Block** - Read the Time Zone. This function block is specific for PCMM2G only.

### 5.29.13.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>No default</td>
<td>If TRUE, request to read the time zone.</td>
</tr>
</tbody>
</table>

### 5.29.13.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>If TRUE, the command completed successfully.</td>
</tr>
<tr>
<td>Error</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>If TRUE, an error has occurred.</td>
</tr>
<tr>
<td>ErrorID</td>
<td>DINT</td>
<td>No range</td>
<td>N/A</td>
<td>Indicates the error if the Error output is set to TRUE.</td>
</tr>
</tbody>
</table>

**Error Codes**

- 23 = Internal error. See controller log for details.
- 15000 = Controller type does not support this function block.

| TimeZone | STRING | No range | N/A  | The time zone the controller should use.                                |

### 5.29.13.3 Remarks

None

### 5.29.13.4 FBD Language

#### Diagram

![Diagram](image)

#### FBD Code

```fbd
Inst_DTGetTimeZone
DTGetTimeZone
Execute
Done
Error
ErrorID
TimeZone
```

### 5.29.13.5 FFLD Language

None
5.29.13.6 IL Language
Not available.

5.29.13.7 ST Language

```plaintext
// read the configured time zone
Inst_DTGetTimeZone( bGetDTZone );
if Inst_DTGetTimeZone.Done then
  bGetDTZone := false;

  if NOT Inst_DTGetTimeZone.Error then
    TimeZone := Inst_DTGetTimeZone.TimeZone;
  else
    ErrorID := Inst_DTGetTimeZone.ErrorID;
end_if;
end_if;
```

See Also

- "DTCurDateTime" (p. 270)
- "DTSetTimeZone" (p. 291)
- "List of Date / Time / NTP ErrorID Codes" (p. 293)

5.29.14 DTEvery

ثلث شكل بقعة - Generate a pulse signal with long period.

5.29.14.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>No default</td>
<td>When TRUE, the signal generation is enabled.</td>
</tr>
</tbody>
</table>
### 5.29.14.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td></td>
<td>Pulse signal.</td>
</tr>
</tbody>
</table>

### 5.29.14.3 Remarks

- This function block provides a pulse signal with a period of more than 24h.
- The period is expressed as:
  \[ \text{DAYS} \times 24h + \text{TM} \]

- Example: Specifying DAYS=1 and TM=6h means a period of 30 hours.

### 5.29.14.4 FBD Language

\[
\begin{array}{c}
\text{Q} \\
\text{RUN} \\
\text{DAYS} \\
\text{TM}
\end{array}
\begin{array}{c}
\text{DTEVERY}
\end{array}
\]

### 5.29.14.5 FFLD Language

\[
\begin{array}{c}
\text{RUN} \\
\text{DAYS} \\
\text{TM}
\end{array}
\begin{array}{c}
\text{DTEVERY}
\end{array}
\begin{array}{c}
\text{Q}
\end{array}
\]

### 5.29.14.6 IL Language

Not available.

### 5.29.14.7 ST Language

\[
(* \text{MyDTEVERY is a declared instance of DTEVERY function block.} *)
\text{MyDTEVERY (RUN, DAYS, TM);}
\text{Q := MyDTEVERY.Q;}
\]

#### See Also

- DTAt
- Real Time Clock Management Functions

### 5.29.15 DTFormat
**Function** - Format the present date/time to a string with a custom format.

### 5.29.15.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMT</td>
<td>STRING</td>
<td>No range</td>
<td>n/a</td>
<td>'%Y/%m/%d-%H:%M:%S'</td>
<td>Format string</td>
</tr>
</tbody>
</table>

### 5.29.15.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>STRING</td>
<td>No range</td>
<td>N/A</td>
<td>String containing formatted date or time.</td>
</tr>
</tbody>
</table>

### 5.29.15.3 Remarks

**IMPORTANT**
The real-time clock may not be available on all controller hardware models. See the controller hardware specifications for real-time clock availability.

- The format string may contain any character.
- Special markers beginning with the `%` character indicates a date/time information:

<table>
<thead>
<tr>
<th>Marker</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%Y</td>
<td>Year including century (e.g., 2006)</td>
</tr>
<tr>
<td>%y</td>
<td>Year without century (e.g., 06)</td>
</tr>
<tr>
<td>%m</td>
<td>Month (1..12)</td>
</tr>
<tr>
<td>%d</td>
<td>Day of the month (1..31)</td>
</tr>
<tr>
<td>%H</td>
<td>Hours (0..23)</td>
</tr>
<tr>
<td>%M</td>
<td>Minutes (0..59)</td>
</tr>
<tr>
<td>%S</td>
<td>Seconds (0..59)</td>
</tr>
<tr>
<td>%T</td>
<td>Milliseconds (0..999)</td>
</tr>
</tbody>
</table>

**Example**

(*) 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 *)

### 5.29.15.4 FBD Language

```
FMT DTFORMAT Q
```

### 5.29.15.5 FFLD Language

* The function is executed only if EN is TRUE. *
* ENO keeps the same value as EN. *
5.29.15.6 IL Language

```
Op1: LD FMT
     DTFORMAT
     ST Q
```

5.29.15.7 ST Language

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

See Also
day_time

5.29.16 DTHour

**Function** - Get the hours from the time stamp.

### 5.29.16.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>DINT</td>
<td>0 to 86,399,999</td>
<td>Milliseconds</td>
<td>No default</td>
<td>The number of milliseconds that have passed since midnight. This value is typically retrieved from DTCurTime.</td>
</tr>
</tbody>
</table>

### 5.29.16.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td>0 to 23</td>
<td>Hours</td>
<td>Hours of the time.</td>
</tr>
</tbody>
</table>

### 5.29.16.3 ST Language

```
Q := DTHour (iTime);
```

See Also
- "DTCurTime" (→ p. 272)
- "DTMin" (→ p. 283)
- "DTMs" (→ p. 284)
- "DTSec" (→ p. 285)

5.29.17 DTListTimeZones

**Function Block** - List the time zones available on the controller.
This function block is specific for PCMM2G only.

### 5.29.17.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>No default</td>
<td>If TRUE, request to read the available time zones.</td>
</tr>
<tr>
<td>TimeZones</td>
<td>STRING[ ]</td>
<td>No range</td>
<td>N/A</td>
<td>No default</td>
<td>An array where the list of time zones available on the system are copied.  This is effectively an output parameter, but because it is an array, it must be an input.</td>
</tr>
</tbody>
</table>

### 5.29.17.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>If TRUE, the command completed successfully.</td>
</tr>
<tr>
<td>Error</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>If TRUE, an error has occurred.</td>
</tr>
<tr>
<td>ErrorID</td>
<td>DINT</td>
<td>No range</td>
<td>N/A</td>
<td>Indicates the error if Error output is TRUE.</td>
</tr>
<tr>
<td>TZCount</td>
<td>DINT</td>
<td>No range</td>
<td>N/A</td>
<td>The number of time zones on the system.</td>
</tr>
</tbody>
</table>

#### Error Codes
- 23 = Internal error. See controller log for details.
- 15000 = Controller type does not support this function block.

### 5.29.17.3 Remarks

None

### 5.29.17.4 FBD Language

![FBD Diagram]

### 5.29.17.5 FFLD Language
5.29.17.6  IL Language
Not available.

5.29.17.7  ST Language

```st
// read the list of supported time zones
Inst_DTListTimeZones( bListDTZones, TimeZones );
if NOT Inst_DTListTimeZones.Error then
    TZCount := Inst_DTListTimeZones.TZCount;
else
    ErrorID := Inst_DTListTimeZones.ErrorID;
end_if;
if Inst_DTListTimeZones.Done then
    bListDTZones := false;
end_if;
```

See Also
- "DTGetTimeZone" (→ p. 277)
- "DTSetTimeZone" (→ p. 291)

5.29.18  DTMin

**Function** - Get the minutes from the time stamp.

5.29.18.1  Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>DINT</td>
<td>0 to 86,399,999</td>
<td>Milliseconds</td>
<td>No default</td>
<td>The number of milliseconds that have passed since midnight. This value is typically retrieved from DTCurTime.</td>
</tr>
</tbody>
</table>

5.29.18.2  Outputs
<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td>0 to 59</td>
<td>Minutes</td>
<td>Minutes of the time.</td>
</tr>
</tbody>
</table>

### 5.29.18.3 ST Language

```
Q := DTMin (iTime);
```

**See Also**

- "DTCurTime" (→ p. 272)
- "DTHour" (→ p. 281)
- "DTMs" (→ p. 284)
- "DTSec" (→ p. 285)

#### 5.29.19 DTMonth

**Function** - Get the month from the date stamp.

### 5.29.19.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>DINT</td>
<td>No range</td>
<td>N/A</td>
<td>No default</td>
<td>Numerical stamp representing a date.</td>
</tr>
</tbody>
</table>

### 5.29.19.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td>1 to 12</td>
<td>N/A</td>
<td>Month of the date.</td>
</tr>
</tbody>
</table>

### 5.29.19.3 ST Language

```
Q := DTMonth (iDate);
```

**See Also**

- "DTCurDate" (→ p. 269)
- "DTDay" (→ p. 272)
- "DTYear" (→ p. 293)

#### 5.29.20 DTMs

**Function** - Get the milliseconds from the time stamp.

### 5.29.20.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>DINT</td>
<td>0 to 86,399,999</td>
<td>Milliseconds</td>
<td>No default</td>
<td>The number of milliseconds that have passed since midnight. This value is typically retrieved from DTCurTime.</td>
</tr>
</tbody>
</table>
5.29.20.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td>0 to 999</td>
<td>Milliseconds</td>
<td>Present milliseconds of the time.</td>
</tr>
</tbody>
</table>

5.29.20.3 ST Language

```
Q := DTM (iTime);
```

See Also

- "DTCurTime" (→ p. 272)
- "DTHour" (→ p. 281)
- "DTMin" (→ p. 283)
- "DTSec" (→ p. 285)

5.29.21 DTSec

Function - Get the seconds from the time stamp.

5.29.21.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>DINT</td>
<td>0 to 86,399,999</td>
<td>Milliseconds</td>
<td>No default</td>
<td>The number of milliseconds that have passed since midnight. This value is typically retrieved from DTCurTime.</td>
</tr>
</tbody>
</table>

5.29.21.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td>0 to 59</td>
<td>Seconds</td>
<td>Seconds of the time.</td>
</tr>
</tbody>
</table>

5.29.21.3 ST Language

```
Q := DTSec (iTime);
```

See Also

- "DTCurTime" (→ p. 272)
- "DTHour" (→ p. 281)
- "DTMin" (→ p. 283)
- "DTMs" (→ p. 284)

5.29.22 DTSetDateTime

Function Block - Sets the local date and time.

This function block is specific for PCMM2G only.

5.29.22.1 Inputs
<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>No default</td>
<td>If TRUE, request to set the local date and time.</td>
</tr>
<tr>
<td>Year</td>
<td>DINT</td>
<td>1900 to 2200</td>
<td>Year</td>
<td>No default</td>
<td>The local date’s new value of the year.</td>
</tr>
<tr>
<td>Month</td>
<td>DINT</td>
<td>1 to 12</td>
<td>Month</td>
<td>No default</td>
<td>The local date’s new value of the month.</td>
</tr>
<tr>
<td>Day</td>
<td>DINT</td>
<td>1 to 31</td>
<td>Day</td>
<td>No default</td>
<td>The local date’s new value of the day.</td>
</tr>
<tr>
<td>Hour</td>
<td>DINT</td>
<td>0 to 23</td>
<td>Hour</td>
<td>No default</td>
<td>The local date’s new value of the hour.</td>
</tr>
<tr>
<td>Min</td>
<td>DINT</td>
<td>0 to 59</td>
<td>Minute</td>
<td>No default</td>
<td>The local date’s new value of the minute.</td>
</tr>
<tr>
<td>Sec</td>
<td>DINT</td>
<td>0 to 60</td>
<td>Second</td>
<td>No default</td>
<td>The local date’s new value of the second.</td>
</tr>
</tbody>
</table>

**NOTE**

60 is valid because leap seconds may have a value of 60.

### TIP

If the UTC time needs to be set, change the time zone to UTC using "DTSetTimeZone" (→ p. 291), set the time, then restore the time zone.

#### 5.29.22.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>If TRUE, the command completed successfully.</td>
</tr>
<tr>
<td>Error</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>If TRUE, an error has occurred.</td>
</tr>
<tr>
<td>ErrorID</td>
<td>DINT</td>
<td>No range</td>
<td>N/A</td>
<td>Indicates the error if the Error output is set to TRUE.</td>
</tr>
</tbody>
</table>

**Error Codes**

- 23 = Internal error. See controller log for details.
- 15000 = Controller type does not support this function block.
- 16202 = Cannot set date / time when NTP synchronization is active.
- 16203 = Invalid date / time value specified.

### 5.29.22.3 Remarks

None

### 5.29.22.4 FBD Language
5.29.22.5 FFLD Language

5.29.22.6 IL Language
Not available.

5.29.22.7 ST Language

```c
// write the date and time
Inst_DTSetDateTime( bSetDateTime, Year, Month, Day, Hour, Min, Sec
```
See Also

- "DTCurDateTime" (→ p. 270)
- "DTSetTimeZone" (→ p. 291)
- "List of Date / Time / NTP ErrorID Codes" (→ p. 293)

5.29.23 DTSetNTPServer

Function Block - Set the NTP server address.
This function block is specific for PCMM2G only.

5.29.23.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>No</td>
<td>If TRUE, request to set the NTP server address.</td>
</tr>
<tr>
<td>NTPServer</td>
<td>STRING</td>
<td>No range</td>
<td>N/A</td>
<td>No</td>
<td>The address of the NTP server used for clock synchronization.</td>
</tr>
</tbody>
</table>

5.29.23.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>If TRUE, the command completed successfully.</td>
</tr>
<tr>
<td>Error</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>If TRUE, an error has occurred.</td>
</tr>
<tr>
<td>ErrorID</td>
<td>DINT</td>
<td>No range</td>
<td>N/A</td>
<td>Indicates the error if the Error output is set to TRUE.</td>
</tr>
</tbody>
</table>

Error Codes

- 23 = Internal error. See controller log for details.
- 15000 = Controller type does not support this function block.

5.29.23.3 Remarks

None

5.29.23.4 FBD Language

```plaintext
); if Inst_DTSetDateTime.Done then
    bSetDateTime := false;
    bError := Inst_DTSetDateTime.Error;
    ErrorID := Inst_DTSetDateTime.ErrorID;
end_if;
```
5.29.23.5 FFLD Language

```
// configure the NTP server address
Inst_DTSetNTPServer( bSetNTPServer, NTPServer );
if Inst_DTSetNTPServer.Done then
    bSetNTPServer := false;
    bError := Inst_DTSetNTPServer.Error;
    ErrorID := Inst_DTSetNTPServer.ErrorID;
end_if;
```

See Also
- "DTCurDateTime" (→ p. 270)
- "DTGetNTPServer" (→ p. 273)
- "DTGetNTPSync" (→ p. 275)
- "DTSetNTPSync" (→ p. 289)
- "List of Date / Time / NTP ErrorID Codes" (→ p. 293)

5.29.24 DTSetNTPSync
**Function Block** - Set the NTP synchronization enable state.
This function block is specific for PCMM2G only.

### 5.29.24.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>No default</td>
<td>If TRUE, request to set the synchronization enable state.</td>
</tr>
</tbody>
</table>
| SynchEn| BOOL      | TRUE, FALSE | N/A  | No default | • TRUE = enable NTP synchronization.  
• FALSE = disable NTP synchronization. |

### 5.29.24.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>If TRUE, the command completed successfully.</td>
</tr>
<tr>
<td>Error</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>If TRUE, an error has occurred.</td>
</tr>
<tr>
<td>ErrorID</td>
<td>DINT</td>
<td>No range</td>
<td>N/A</td>
<td>Indicates the error if the Error output is set to TRUE.</td>
</tr>
</tbody>
</table>

**Error Codes**
- 23 = Internal error. See controller log for details.
- 15000 = Controller type does not support this function block.

### 5.29.24.3 Remarks
None

### 5.29.24.4 FBD Language

![FBD Diagram](image)

### 5.29.24.5 FFLD Language
5.29.24.6 IL Language

Not available.

5.29.24.7 ST Language

```plaintext
// enable NTP server synchronization
Inst_DTSetNTPSync( bSetNTPSync, bNTPSyncEnable );
if Inst_DTSetNTPSync.Done then
    bSetNTPSync := false;
    bError := Inst_DTSetNTPSync.Error;
    ErrorID := Inst_DTSetNTPSync.ErrorID;
end_if;
```

See Also
- "DTCurDateTime" (➡ p. 270)
- "DTGetNTPServer" (➡ p. 273)
- "DTGetNTPSync" (➡ p. 275)
- "DTSetNTPServer" (➡ p. 288)
- "List of Date / Time / NTP ErrorID Codes" (➡ p. 293)

5.29.25 DTSetTimeZone

Function Block - Set the time zone.

This function block is specific for PCMM2G only.

5.29.25.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>No default</td>
<td>If TRUE, request to set the time zone.</td>
</tr>
<tr>
<td>TimeZone</td>
<td>STRING</td>
<td>No range</td>
<td>N/A</td>
<td>No default</td>
<td>The time zone the controller should use.</td>
</tr>
</tbody>
</table>

5.29.25.2 Outputs
### Output Data Type

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>If TRUE, the command completed successfully.</td>
</tr>
<tr>
<td>Error</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>If TRUE, an error has occurred.</td>
</tr>
<tr>
<td>ErrorID</td>
<td>DINT</td>
<td>No range</td>
<td>N/A</td>
<td>Indicates the error if the Error output is set to TRUE.</td>
</tr>
</tbody>
</table>

#### Error Codes
- 23 = Internal error. See controller log for details.
- 15000 = Controller type does not support this function block.
- 16201 = Invalid time zone.

### 5.29.25.3 Remarks
None

### 5.29.25.4 FBD Language

![FBD Diagram]

### 5.29.25.5 FFLD Language

![FFLD Diagram]

### 5.29.25.6 IL Language

Not available.
5.29.25.7  ST Language

```plaintext
// configure the time zone
Inst_DTSetTimeZone( bSetDTZone, TimeZone );
if Inst_DTSetTimeZone.Done then
  bSetDTZone := false;
  bError := Inst_DTSetTimeZone.Error;
  ErrorID := Inst_DTSetTimeZone.ErrorID;
end_if;
```

See Also

- "DTCurDateTime" (☞ p. 270)
- "DTGetTimeZone" (☞ p. 277)
- "DTListTimeZones" (☞ p. 281)
- "List of Date / Time / NTP ErrorID Codes" (☞ p. 293)

5.29.26 DTYear

Function - Get the year from the date stamp.

5.29.26.1  Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>DINT</td>
<td>No range</td>
<td>N/A</td>
<td>No default</td>
<td>Numerical stamp representing a date.</td>
</tr>
</tbody>
</table>

5.29.26.2  Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td>No range</td>
<td>N/A</td>
<td>Year of the date.</td>
</tr>
</tbody>
</table>

5.29.26.3  ST Language

```plaintext
Q := DTYear (iDate);
```

See Also

- "DTCurDateTime" (☞ p. 269)
- "DTDay" (☞ p. 272)
- "DTMonth" (☞ p. 284)

5.29.27 List of Date / Time / NTP ErrorID Codes

- 23 = Internal error. See controller log for details.
- 15000 = Controller type does not support this function block.
- 16200 = Could not read NTP server configuration file.
- 16201 = Invalid time zone.
- 16202 = Cannot set date / time when NTP synchronization is active.
- 16203 = Invalid date / time value specified.

5.30  SerializeIn
Function - Extract the value of a variable from a binary frame.

### 5.30.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGENDIAN</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td>N/A</td>
<td>TRUE</td>
<td>TRUE if the frame is encoded with BigEndian format.</td>
</tr>
<tr>
<td>DATA</td>
<td>ANY(*)</td>
<td>N/A</td>
<td>N/A</td>
<td>No default</td>
<td>Destination variable to be copied.</td>
</tr>
<tr>
<td>EN</td>
<td>BOOL</td>
<td>0, 1</td>
<td>N/A</td>
<td>No default</td>
<td>Execute the function.</td>
</tr>
<tr>
<td>FRAME</td>
<td>USINT</td>
<td>0,+65535</td>
<td>N/A</td>
<td>N/A</td>
<td>Source buffer - must be an array.</td>
</tr>
<tr>
<td>POS</td>
<td>DINT</td>
<td>0,+65535</td>
<td>N/A</td>
<td>N/A</td>
<td>Position in the source buffer.</td>
</tr>
</tbody>
</table>

(*): DATA cannot be a STRING.

### 5.30.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEXTPOS</td>
<td>DINT</td>
<td>N/A</td>
<td></td>
<td>Position in the source buffer after the copied data.</td>
</tr>
<tr>
<td>OK</td>
<td>BOOL</td>
<td>N/A</td>
<td></td>
<td>Returns true when the function successfully executes. See Function - General Rules.</td>
</tr>
</tbody>
</table>

### 5.30.3 Remarks

This function is used for extracting data from a communication frame in binary format.

- The DATA input must be directly connected to a variable. It cannot be a constant or complex expression. This variable is forced with the extracted value.

- The FRAME input must fit the input position and data size. If the value cannot be safely extracted, the function returns 0 (zero).

- This function cannot be used to serialize STRING variables.
- The function returns the position in the source frame, after the extracted data. The return value can be used as a position for the next serialization.

This function extracts these number of bytes from the source frame:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td>BOOL, BYTE, SINT, and USINT variables.</td>
</tr>
<tr>
<td>2 bytes</td>
<td>INT, UINT, and WORD variables.</td>
</tr>
<tr>
<td>4 bytes</td>
<td>DINT, DWORD, REAL, and UDINT variables.</td>
</tr>
<tr>
<td>8 bytes</td>
<td>LINT and LREAL variables.</td>
</tr>
</tbody>
</table>
5.30.4 FBD Language

```
FRAME
DATA
POS
BIGENDIAN
SERIALIZEIN
```

5.30.5 FFLD Language

- In the 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.

```
EN
FRAME
DATA
POS
BIGENDIAN
SERIALIZEIN
ENO
```

5.30.6 IL Language

Not available.

5.30.7 ST Language

```
Q := SERIALIZEIN (FRAME, DATA, POS, BIGENDIAN);
```

See Also

"SerializeOut" (→ p. 296)

5.30.8 Arguments

5.30.8.1 Input

<table>
<thead>
<tr>
<th>Description</th>
<th>Execute the function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>En</strong></td>
<td></td>
</tr>
<tr>
<td>Data type</td>
<td>BOOL</td>
</tr>
<tr>
<td>Range</td>
<td>[0,1]</td>
</tr>
<tr>
<td>Unit</td>
<td>N/A</td>
</tr>
<tr>
<td>Default</td>
<td>_</td>
</tr>
<tr>
<td><strong>Frame[]</strong></td>
<td>Source buffer - must be an array.</td>
</tr>
<tr>
<td>Data type</td>
<td>USINT</td>
</tr>
<tr>
<td>Range</td>
<td>[0,+65535]</td>
</tr>
<tr>
<td>Unit</td>
<td>N/A</td>
</tr>
<tr>
<td>Default</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Data

<table>
<thead>
<tr>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination variable to be copied</td>
<td>any except STRING</td>
<td>—</td>
<td>N/A</td>
<td>—</td>
</tr>
</tbody>
</table>

### Pos

<table>
<thead>
<tr>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position in the source buffer</td>
<td>DINT</td>
<td>[0,+65535]</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### BigEndian

<table>
<thead>
<tr>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE if the frame is encoded with Big Endian format.</td>
<td>BOOL</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

### 5.30.8.2 Output

<table>
<thead>
<tr>
<th>Description</th>
<th>Data type</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns true when the function successfully executes. See Function - General Rules.</td>
<td>BOOL</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### NextPos

<table>
<thead>
<tr>
<th>Description</th>
<th>Data type</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position in the source buffer after the extracted data. 0 in case or error (invalid position / buffer size).</td>
<td>DINT</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### 5.31 SerializeOut

- **PLCopen**: ✔
- **Function**: Copy the value of a variable to a binary frame.

### 5.31.1 Inputs
<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGENDIAN</td>
<td>BOOL</td>
<td>TRUE, FALSE</td>
<td></td>
<td>TRUE</td>
<td>TRUE if the frame is encoded with Big Endian format.</td>
</tr>
<tr>
<td>DATA</td>
<td>ANY(*)</td>
<td>N/A</td>
<td>N/A</td>
<td>No default</td>
<td>Source variable to be copied.</td>
</tr>
<tr>
<td>EN</td>
<td>BOOL</td>
<td>0, 1</td>
<td>N/A</td>
<td>No default</td>
<td>Execute the function.</td>
</tr>
<tr>
<td>FRAME</td>
<td>USINT</td>
<td>0,+65535</td>
<td>N/A</td>
<td>N/A</td>
<td>Destination buffer - must be an array.</td>
</tr>
<tr>
<td>POS</td>
<td>DINT</td>
<td>0,+65535</td>
<td>N/A</td>
<td>N/A</td>
<td>Position in the destination buffer.</td>
</tr>
</tbody>
</table>

(*) DATA cannot be a STRING.

### 5.31.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEXTPOS</td>
<td>DINT</td>
<td></td>
<td>N/A</td>
<td>• Position in the destination buffer after the copied data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 0 (zero) in case or error (e.g., invalid position or buffer size).</td>
</tr>
<tr>
<td>OK</td>
<td>BOOL</td>
<td></td>
<td>N/A</td>
<td>Returns true when the function successfully executes. See Function - General Rules.</td>
</tr>
</tbody>
</table>

### 5.31.3 Remarks

This function is used for building a communication frame in binary format.

- 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 (zero).
- This function cannot be used to serialize STRING variables.
- The function returns the position in the destination frame, after the copied data.
  - The return value can be used as a position for the next serialization.

This function copies these number of bytes to the destination frame:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td>BOOL, BYTE, SINT, and USINT variables.</td>
</tr>
<tr>
<td>2 bytes</td>
<td>INT, UINT, and WORD variables.</td>
</tr>
<tr>
<td>4 bytes</td>
<td>DINT, DWORD, REAL, and UDINT variables.</td>
</tr>
<tr>
<td>8 bytes</td>
<td>LINT and LREAL variables.</td>
</tr>
</tbody>
</table>

### 5.31.4 FBD Language

[Diagram of FBD language]
5.31.5 FFLD Language

- In the 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.

```
<table>
<thead>
<tr>
<th>EN</th>
<th>SERIALIZEOUT</th>
<th>ENO</th>
</tr>
</thead>
</table>
```

5.31.6 IL Language

Not available.

5.31.7 ST Language

```q := SERIALIZEOUT (FRAME, DATA, POS, BIGENDIAN);
```

See Also

"SerializeIn" (p. 293)

5.31.8 Arguments

5.31.8.1 Input

<table>
<thead>
<tr>
<th>En</th>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Execute the function</td>
<td>BOOL</td>
<td>[0,1]</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frame[]</th>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Destination buffer - must be an array.</td>
<td>USINT</td>
<td>[0,+65535]</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data</th>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Source variable to be copied</td>
<td>any except STRING</td>
<td></td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>


### 5.31.8.2 Output

<table>
<thead>
<tr>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>BOOL</td>
<td>[0,1]</td>
<td>N/A</td>
<td>—</td>
</tr>
</tbody>
</table>

**Description**
Returns true when the function successfully executes. See [Function - General rules](#).

<table>
<thead>
<tr>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>NextPos</td>
<td>DINT</td>
<td>[0,+65535]</td>
<td>N/A</td>
<td>—</td>
</tr>
</tbody>
</table>

**Description**
Position in the destination buffer after the copied data. 0 in case of an error (invalid position / buffer size).

### 5.32 SigID

**PLCopen**

*Function* - Get the identifier of a Signal resource.

#### 5.32.1 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.

#### 5.32.2 Outputs

**ID** : DINT ID of the signal - to be passed to other blocks.

#### 5.32.3 Remarks

- This function enables you to get the identifier of a signal defined as a resource.
- Some blocks have arguments that refer to a signal "signal" resource.
  - For all these blocks, the signal argument is materialized by a numerical identifier.

#### 5.32.4 ST Language
ID := SigID ('MySignal', 'FirstColumn');

5.32.5 FBD Language

```
SIGNAL     | SigID     | ID
COL
```

5.32.6 FFLD Language

```
EN [ ] [ ] [ ]
SIGNAL     | SigID     | ENO
COL
```

5.32.7 IL Language

```
Op1: LD    'MySignal'
       SigID 'FirstColumn'
       ST ID
```

See Also

- "SigPlay" (→ p. 301)
- "SigScale" (→ p. 303)
- there is supposed to be a link to a Copa-Data Analog Signals Resources topic that we don't have
  - verify this CD topic is needed and link to it if so.
5.33 SigPlay

*Function block* - Generate a signal defined in a resource.

### 5.33.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>BOOL</td>
<td>Triggering command.</td>
</tr>
<tr>
<td>ID</td>
<td>DINT</td>
<td>ID of the signal resource, provided by the &quot;SigID&quot; (→ p. 299) function.</td>
</tr>
<tr>
<td>RST</td>
<td>BOOL</td>
<td>Reset command.</td>
</tr>
<tr>
<td>TM</td>
<td>TIME</td>
<td>Minimum duration between two changes of the output.</td>
</tr>
</tbody>
</table>

### 5.33.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
<td>TRUE when the signal is finished.</td>
</tr>
<tr>
<td>OUT</td>
<td>REAL</td>
<td>Generated signal.</td>
</tr>
<tr>
<td>ET</td>
<td>TIME</td>
<td>Elapsed time.</td>
</tr>
</tbody>
</table>

### 5.33.3 Remarks

- The ID argument is the identifier of the signal"signal" resource.
  - Use the "SigID" (→ p. 299) 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 (zero).

- 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, use the "SigScale" (→ p. 303) function if you want to trigger the signal using a specific timer.

### 5.33.4 ST Language

```plaintext
MySig (II, ID, RST, TM);
Q := MySig.Q;
OUT := MySig.OUT;
ET := MySig.ET;
```

### 5.33.5 FBD Language
5.33.6 FFLD Language

5.33.7 IL Language

Op1: FFLD  IN
    SigScale ID
    ST    Q

See Also

- "SigID" (→ p. 299)
- "SigScale" (→ p. 303)
- there is supposed to be a link to a Copa-Data Analog Signals Resources topic that we don’t have - verify this CD topic is needed and link to it if so.
5.34 SigScale

*Function* - Get a point from a Signal resource.

5.34.1 Inputs

| ID | DINT | ID of the signal resource, provided by the "SigID" (p. 299) function. |
| IN | TIME | Time (X) coordinate of the wished point within the signal resource. |

5.34.2 Outputs

| Q | REAL | Value (Y) coordinate of the point in the signal. |

5.34.3 Remarks

- The ID argument is the identifier of the signal "signal" resource.
  - Use the "SigID" (p. 299) function to get this value.

- This function:
  - Converts a time value to an analog value such as defined in the signal resource.
  - Can be used instead of the "SigPlay" (p. 301) function block to trigger the signal using a specific timer.

5.34.4 FBD Language

```
SigScale
```

5.34.5 FFLD Language

```
EN [ ] [ ] [ ] [ ]
SigScale ID ENO ( ) [ ]
ID IN Q
```

5.34.6 IL Language

```
Op1: LD IN
SigScale ID
ST Q
```

5.34.7 ST Language

```q := SigScale (ID, IN);```
See Also

- "SigID" ( \( \rightarrow \) p. 299)
- "SigPlay" ( \( \rightarrow \) p. 301)
- there is supposed to be a link to a Copa-Data Analog Signals Resources topic that we don't have - verify this CD topic is needed and link to it if so.

5.35 stackint

<table>
<thead>
<tr>
<th>PLCopen ✓</th>
</tr>
</thead>
</table>

Function Block - Manages a stack of DINT integers.

5.35.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Command: When changing from FALSE to TRUE, the value of IN is pushed on the stack.</td>
</tr>
<tr>
<td>POP</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Pop command: When changing from FALSE to TRUE, deletes the top of the stack.</td>
</tr>
<tr>
<td>R1</td>
<td>BOOL</td>
<td></td>
<td></td>
<td></td>
<td>Reset command: If TRUE, the stack is emptied and its size is set to N.</td>
</tr>
<tr>
<td>IN</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Value to be pushed on a rising pulse of PUSH.</td>
</tr>
<tr>
<td>N</td>
<td>DINT</td>
<td></td>
<td></td>
<td></td>
<td>Maximum stack size - cannot exceed 128.</td>
</tr>
</tbody>
</table>

5.35.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPTY</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE if the stack is empty.</td>
</tr>
<tr>
<td>OFLO</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE if the stack is full.</td>
</tr>
<tr>
<td>OUT</td>
<td>DINT</td>
<td></td>
<td></td>
<td>Value at the top of the stack.</td>
</tr>
</tbody>
</table>

5.35.3 Remarks

- Push and pop operations are performed on rising pulse of PUSH and POP inputs.
- The specified size (N) is taken into account only when the R1 (reset) input is TRUE.

5.35.4 FBD Language
5.35.5 FFLD Language

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

5.35.6 IL Language

```plaintext
(* 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
```

5.35.7 ST Language

```plaintext
(* MyStack is a declared instance of STACKINT function block *)
MyStack (PUSH, POP, R1, IN, N);
EMPTY := MyStack.EMPTY;
OFLO := MyStack.OFLO;
OUT := MyStack.OUT;
```

See Also

- "average / averageL" (→ p. 232)
- "derivate" (→ p. 234)
- "hyster" (→ p. 249)
- "integral" (→ p. 250)
- "lim_alarm" (→ p. 254)

5.36 SurfLin

Function Block - Linear interpolation on a surface.
5.36.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>REAL</td>
<td></td>
<td></td>
<td>X</td>
<td>X coordinate of the point to be interpolated.</td>
</tr>
<tr>
<td>XAxis</td>
<td>REAL[]</td>
<td></td>
<td></td>
<td></td>
<td>X coordinates of the known points of the X axis.</td>
</tr>
<tr>
<td>Y</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Y</td>
<td>Y coordinate of the point to be interpolated.</td>
</tr>
<tr>
<td>YAxis</td>
<td>REAL[]</td>
<td></td>
<td></td>
<td></td>
<td>Y coordinates of the known points of the Y axis.</td>
</tr>
<tr>
<td>ZVal</td>
<td>REAL[,]</td>
<td></td>
<td></td>
<td></td>
<td>Z coordinate of the points defined by the axis.</td>
</tr>
</tbody>
</table>

5.36.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR</td>
<td>DINT</td>
<td></td>
<td></td>
<td>• Error code if failed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 0 (zero) if OK.</td>
</tr>
<tr>
<td>OK</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>TRUE if successful.</td>
</tr>
<tr>
<td>Z</td>
<td>REAL</td>
<td></td>
<td></td>
<td>Interpolated Z value corresponding to the X,Y input point.</td>
</tr>
</tbody>
</table>

5.36.3 Remarks

is this a function or function block?

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;
- If 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:
### 5.37 VLID

**Function** - Get the identifier (ID) of an embedded list of variables.

#### 5.37.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>STRING</td>
<td></td>
<td></td>
<td></td>
<td>Pathname of the list file (.SPL or .TXT) - must be a constant value.</td>
</tr>
</tbody>
</table>

#### 5.37.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Data Type</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>DINT</td>
<td></td>
<td></td>
<td>ID of the list - to be passed to other blocks.</td>
</tr>
</tbody>
</table>

#### 5.37.3 Remarks

- This function is used to create an Identifier (ID) or ListID for a list of application variables that are typically stored on the development PC.
- The list of application variables:
  - is a simple .TXT file.
  - can contain only one variable name per line
  - can be only global variables
- This function’s ID output can be used as an input to "LogFileCSV (→ p. 255)."
  - It defines the application variables whose present value is recorded each time LogFileCSV is executed.

**IMPORTANT**

List files are read at compiling time and are embedded into the downloaded application code. This implies that a modification performed in the list file after downloading is not taken into account by the application.

#### 5.37.4 FBD Language

```
LIST VLID ID
```

#### 5.37.5 FFLD Language
The function is executed only if EN is TRUE.

5.37.6 IL Language

Op1: LD  'MyFile.txt'
      VLID COL
      ST   ListID

5.37.7 ST Language

ID := VLID ('MyFile.spl');
6 Support and Services

About KOLLMORGEN

Kollmorgen is a leading provider of motion systems and components for machine builders. Through world-class knowledge in motion, industry-leading quality and deep expertise in linking and integrating standard and custom products, Kollmorgen delivers breakthrough solutions that are unmatched in performance, reliability and ease-of-use, giving machine builders an irrefutable marketplace advantage.

Join the Kollmorgen Developer Network for product support. Ask the community questions, search the knowledge base for answers, get downloads, and suggest improvements.

North America
KOLLMORGEN
201 West Rock Road
Radford, VA 24141, USA
Web: www.kollmorgen.com
Mail: support@kollmorgen.com
Tel.: +1 - 540 - 633 - 3545
Fax: +1 - 540 - 639 - 4162

Europe
KOLLMORGEN Europe GmbH
Pempelfurtstr. 1
40880 Ratingen, Germany
Web: www.kollmorgen.com
Mail: technik@kollmorgen.com
Tel.: +49 - 2102 - 9394 - 0
Fax: +49 - 2102 - 9394 - 3155

South America
KOLLMORGEN
Avenida João Paulo Ablas, 2970
Jardim da Glória, Cotia - SP
CEP 06711-250, Brazil
Web: www.kollmorgen.com
Mail: contato@kollmorgen.com
Tel.: +55 11 4615-6300

China and SEA
KOLLMORGEN
Room 302, Building 5, Li hpao Plaza,
88 Shenbin Road, Minhang District,
Shanghai, China.
Web: www.kollmorgen.cn
Mail: sales.china@kollmorgen.com
Tel.: +86 - 400 668 2802
Fax: +86 - 21 6248 5367