Trademarks and Copyrights

Copyrights
Copyright © 2009-2019 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:
- 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 GNU Lesser General Public License - see alsoGPL terms)
- Zlib software library
- curl software library
- Mongoose software (distributed under the MIT License - see terms)
- JsonCpp software (distributed under the MIT License - see terms)
- U-Boot, a universal boot loader is used by the AKD PDMM and PCMM (distributed under the terms of the GNU General Public License). The U-Boot source files, copyright notice, and readme are available on the distribution disk that is included with the AKD PDMM and PCMM.

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

Disclaimer
The information in this document (Version M published on 6/28/2019) 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 Table of Contents .................................................................................................................. 3
2 Programming Languages ....................................................................................................... 33
  2.1 Sequential Function Chart (SFC) ...................................................................................... 33
      2.1.1 SFC Execution at Runtime .................................................................................... 33
      2.1.2 Hierarchy of SFC programs .................................................................................... 34
  2.2 Free Form Ladder Diagram (FFLD) .................................................................................. 35
      2.2.1 Use of the "EN" input and the "ENO" output for blocks .......................................... 35
      2.2.2 Contacts and coils ................................................................................................ 36
      2.2.2.1 FFLD Contacts ................................................................................................. 36
      2.2.2.2 FFLD Coils ....................................................................................................... 38
3 PLC Standard Libraries .......................................................................................................... 40
  3.1 Basic Operations ............................................................................................................... 41
      3.1.1 := .......................................................................................................................... 42
      3.1.1.1 Inputs ............................................................................................................... 42
      3.1.1.2 Outputs ............................................................................................................ 42
      3.1.1.3 Remarks ......................................................................................................... 42
      3.1.1.4 ST Language .................................................................................................... 42
      3.1.1.5 FBD Language ................................................................................................ 42
      3.1.1.6 FFLD Language ............................................................................................... 42
      3.1.1.7 IL Language: .................................................................................................... 42
      3.1.2 Access to bits of an integer ...................................................................................... 43
      3.1.3 Differences Between Functions and Function Blocks .......................................... 44
      3.1.4 Calling a sub-program ............................................................................................. 45
      3.1.4.1 ST Language .................................................................................................... 45
      3.1.4.2 FBD and FFLD Languages .............................................................................. 45
      3.1.4.3 IL Language .................................................................................................... 45
      3.1.5 CASE OF ELSE END_CASE ............................................................................... 46
      3.1.5.1 Syntax ............................................................................................................. 46
      3.1.5.2 Remarks ......................................................................................................... 46
      3.1.5.3 ST Language .................................................................................................... 46
      3.1.5.4 FBD Language ................................................................................................ 46
      3.1.5.5 FFLD Language ............................................................................................... 46
      3.1.5.6 IL Language .................................................................................................... 46
      3.1.6 COUNTOF .............................................................................................................. 47
      3.1.6.1 Inputs ............................................................................................................. 47
      3.1.6.2 Outputs .......................................................................................................... 47
      3.1.6.3 Remarks ....................................................................................................... 47
      3.1.6.4 ST Language .................................................................................................... 47
      3.1.6.5 FBD Language ................................................................................................ 47
      3.1.6.6 FFLD Language ............................................................................................... 47
      3.1.6.7 IL Language .................................................................................................... 47
      3.1.7 DEC ....................................................................................................................... 48
      3.1.7.1 Inputs ............................................................................................................. 48
      3.1.7.2 Outputs .......................................................................................................... 48
3.1.13.6 FBD Language ................................................................. 56
3.1.13.7 FFLD Language ............................................................... 56
3.1.14 ON ................................................................. 57
3.1.14.1 Syntax ................................................................. 57
3.1.14.2 Remarks ................................................................. 57
3.1.14.3 ST Language ............................................................... 57
3.1.15 ( ) ................................................................. 58
3.1.15.1 Remarks ................................................................. 58
3.1.15.2 ST Language ............................................................... 58
3.1.15.3 FBD Language ............................................................... 58
3.1.15.4 FFLD Language ............................................................... 58
3.1.15.5 IL Language ................................................................. 58
3.1.16 REPEAT UNTIL END_REPEAT ................................................................. 59
3.1.16.1 Syntax ................................................................. 59
3.1.16.2 Remarks ................................................................. 59
3.1.16.3 ST Language ............................................................... 59
3.1.16.4 FBD Language ............................................................... 59
3.1.16.5 FFLD Language ............................................................... 59
3.1.16.6 IL Language ................................................................. 59
3.1.17 RETURN RET RETC RETNC RETCN ................................................................. 60
3.1.17.1 Remarks ................................................................. 60
3.1.17.2 ST Language ............................................................... 60
3.1.17.3 FBD Language ............................................................... 60
3.1.17.4 FFLD Language ............................................................... 60
3.1.17.5 IL Language ................................................................. 61
3.1.18 WAIT / WAIT_TIME ................................................................. 62
3.1.18.1 Syntax ................................................................. 62
3.1.18.2 Remarks ................................................................. 62
3.1.18.3 ST Language ............................................................... 62
3.1.19 WHILE DO END_WHILE ................................................................. 63
3.1.19.1 Syntax ................................................................. 63
3.1.19.2 Remarks ................................................................. 63
3.1.19.3 ST Language ............................................................... 63
3.1.19.4 FBD Language ............................................................... 63
3.1.19.5 FFLD Language ............................................................... 63
3.1.19.6 IL Language ................................................................. 63
3.2 Boolean operations ................................................................. 64
3.2.1 FLIPFLOP ................................................................. 65
3.2.1.1 Inputs ................................................................. 65
3.2.1.2 Outputs ................................................................. 65
3.2.1.3 Remarks ................................................................. 65
3.2.1.4 ST Language ............................................................... 65
3.2.1.5 FBD Language ............................................................... 65
3.2.1.6 FFLD Language ............................................................... 65
3.2.1.7 IL Language ................................................................. 65
3.2.2 F TRIG ................................................................. 66
3.2.2.1 Inputs ................................................................. 66
3.2.2.2 Outputs ................................................................. 66
3.2.2.3 Truth table ......................................................... 66
3.2.2.4 Remarks ............................................................... 66
3.2.2.5 ST Language ....................................................... 66
3.2.2.6 FBD Language ..................................................... 66
3.2.2.7 FFLD Language .................................................... 66
3.2.2.8 IL Language: ....................................................... 67
3.2.3 NOT ................................................................. 68
3.2.3.1 Inputs ............................................................... 68
3.2.3.2 Outputs ............................................................. 68
3.2.3.3 Truth table .......................................................... 68
3.2.3.4 Remarks ............................................................. 68
3.2.3.5 ST Language ....................................................... 68
3.2.3.6 FBD Language ..................................................... 68
3.2.3.7 FFLD Language .................................................... 68
3.2.3.8 IL Language: ....................................................... 69
3.2.4 QOR ................................................................. 70
3.2.4.1 Inputs ............................................................... 70
3.2.4.2 Outputs ............................................................. 70
3.2.4.3 Remarks ............................................................. 70
3.2.4.4 ST Language ....................................................... 70
3.2.4.5 FBD Language ..................................................... 70
3.2.4.6 FFLD Language .................................................... 70
3.2.4.7 IL Language ........................................................ 70
3.2.5 R ................................................................. 71
3.2.5.1 Inputs ............................................................... 71
3.2.5.2 Outputs ............................................................. 71
3.2.5.3 Truth table .......................................................... 71
3.2.5.4 Remarks ............................................................. 71
3.2.5.5 ST Language ....................................................... 71
3.2.5.6 FBD Language ..................................................... 71
3.2.5.7 FFLD Language .................................................... 71
3.2.5.8 IL Language: ....................................................... 71
3.2.6 RS ................................................................. 72
3.2.6.1 Inputs ............................................................... 72
3.2.6.2 Outputs ............................................................. 72
3.2.6.3 Truth table .......................................................... 72
3.2.6.4 Remarks ............................................................. 72
3.2.6.5 ST Language ....................................................... 72
3.2.6.6 FBD Language ..................................................... 72
3.2.6.7 FFLD Language .................................................... 73
3.2.6.8 IL Language: ....................................................... 73
3.2.7 R_TRIG ............................................................. 74
3.2.7.1 Inputs ............................................................... 74
3.2.7.2 Outputs ............................................................. 74
3.2.7.3 Truth table .......................................................... 74
3.2.7.4 Remarks ............................................................. 74
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.1.7 IL Language</td>
<td>84</td>
</tr>
<tr>
<td>3.3.2 Divide</td>
<td>85</td>
</tr>
<tr>
<td>3.3.2.1 Inputs</td>
<td>85</td>
</tr>
<tr>
<td>3.3.2.2 Outputs</td>
<td>85</td>
</tr>
<tr>
<td>3.3.2.3 Remarks</td>
<td>85</td>
</tr>
<tr>
<td>3.3.2.4 ST Language</td>
<td>85</td>
</tr>
<tr>
<td>3.3.2.5 FBD Language</td>
<td>85</td>
</tr>
<tr>
<td>3.3.2.6 FFLD Language</td>
<td>85</td>
</tr>
<tr>
<td>3.3.2.7 IL Language:</td>
<td>85</td>
</tr>
<tr>
<td>3.3.3 NEG</td>
<td>86</td>
</tr>
<tr>
<td>3.3.3.1 Inputs</td>
<td>86</td>
</tr>
<tr>
<td>3.3.3.2 Outputs</td>
<td>86</td>
</tr>
<tr>
<td>3.3.3.3 Truth table (examples)</td>
<td>86</td>
</tr>
<tr>
<td>3.3.3.4 Remarks</td>
<td>86</td>
</tr>
<tr>
<td>3.3.3.5 ST Language</td>
<td>86</td>
</tr>
<tr>
<td>3.3.3.6 FBD Language</td>
<td>86</td>
</tr>
<tr>
<td>3.3.3.7 FFLD Language</td>
<td>86</td>
</tr>
<tr>
<td>3.3.4 LIMIT</td>
<td>87</td>
</tr>
<tr>
<td>3.3.4.1 Inputs</td>
<td>87</td>
</tr>
<tr>
<td>3.3.4.2 Outputs</td>
<td>87</td>
</tr>
<tr>
<td>3.3.4.3 Function diagram</td>
<td>87</td>
</tr>
<tr>
<td>3.3.4.4 Remarks</td>
<td>87</td>
</tr>
<tr>
<td>3.3.4.5 ST Language</td>
<td>87</td>
</tr>
<tr>
<td>3.3.4.6 FBD Language</td>
<td>87</td>
</tr>
<tr>
<td>3.3.4.7 FFLD Language</td>
<td>87</td>
</tr>
<tr>
<td>3.3.4.8 IL Language:</td>
<td>88</td>
</tr>
<tr>
<td>3.3.5 MAX</td>
<td>89</td>
</tr>
<tr>
<td>3.3.5.1 Inputs</td>
<td>89</td>
</tr>
<tr>
<td>3.3.5.2 Outputs</td>
<td>89</td>
</tr>
<tr>
<td>3.3.5.3 Remarks</td>
<td>89</td>
</tr>
<tr>
<td>3.3.5.4 ST Language</td>
<td>89</td>
</tr>
<tr>
<td>3.3.5.5 FBD Language</td>
<td>89</td>
</tr>
<tr>
<td>3.3.5.6 FFLD Language</td>
<td>89</td>
</tr>
<tr>
<td>3.3.5.7 IL Language:</td>
<td>89</td>
</tr>
<tr>
<td>3.3.6 MIN</td>
<td>90</td>
</tr>
<tr>
<td>3.3.6.1 Inputs</td>
<td>90</td>
</tr>
<tr>
<td>3.3.6.2 Outputs</td>
<td>90</td>
</tr>
<tr>
<td>3.3.6.3 Remarks</td>
<td>90</td>
</tr>
<tr>
<td>3.3.6.4 ST Language</td>
<td>90</td>
</tr>
<tr>
<td>3.3.6.5 FBD Language</td>
<td>90</td>
</tr>
<tr>
<td>3.3.6.6 FFLD Language</td>
<td>90</td>
</tr>
<tr>
<td>3.3.6.7 IL Language:</td>
<td>90</td>
</tr>
<tr>
<td>3.3.7 MOD / MODR / MODLR</td>
<td>91</td>
</tr>
<tr>
<td>3.3.7.1 Remarks</td>
<td>91</td>
</tr>
<tr>
<td>3.3.7.2 ST Language</td>
<td>91</td>
</tr>
<tr>
<td>3.3.7.3 FBD Language</td>
<td>91</td>
</tr>
<tr>
<td>3.3.7.4 FFLD Language</td>
<td>91</td>
</tr>
</tbody>
</table>
3.3.7.5 IL Language ................................................................. 91
3.3.8* Multiply ................................................................. 92
  3.3.8.1 Inputs ................................................................. 92
  3.3.8.2 Outputs ............................................................... 92
  3.3.8.3 Remarks ............................................................... 92
  3.3.8.4 ST Language ............................................................ 92
  3.3.8.5 FBD Language ....................................................... 92
  3.3.8.6 FFLD Language ...................................................... 92
  3.3.8.7 IL Language: .......................................................... 92
3.3.9 ODD .............................................................. 93
  3.3.9.1 Inputs ................................................................. 93
  3.3.9.2 Outputs ............................................................... 93
  3.3.9.3 Remarks ............................................................... 93
  3.3.9.4 ST Language ............................................................ 93
  3.3.9.5 FBD Language ....................................................... 93
  3.3.9.6 FFLD Language ...................................................... 93
  3.3.9.7 IL Language: .......................................................... 93
3.3.10 SetWithin ............................................................. 94
  3.3.10.1 Inputs ................................................................. 94
  3.3.10.2 Outputs ............................................................... 94
  3.3.10.3 Truth Table .......................................................... 94
  3.3.10.4 Remarks ............................................................... 94
3.3.11 Subtraction ............................................................ 95
  3.3.11.1 Inputs ................................................................. 95
  3.3.11.2 Outputs ............................................................... 95
  3.3.11.3 Remarks ............................................................... 95
  3.3.11.4 ST Language ............................................................ 95
  3.3.11.5 FBD Language ....................................................... 95
  3.3.11.6 FFLD Language ...................................................... 95
  3.3.11.7 IL Language: .......................................................... 95
3.4 Comparison Operations .................................................. 96
  3.4.1 CMP ................................................................. 97
    3.4.1.1 Inputs ................................................................. 97
    3.4.1.2 Outputs ............................................................... 97
    3.4.1.3 Remarks ............................................................... 97
    3.4.1.4 ST Language ............................................................ 97
    3.4.1.5 FBD Language ....................................................... 97
    3.4.1.6 FFLD Language ...................................................... 97
    3.4.1.7 IL Language: .......................................................... 97
  3.4.2 >= GE ................................................................. 98
    3.4.2.1 Inputs ................................................................. 98
    3.4.2.2 Outputs ............................................................... 98
    3.4.2.3 Remarks ............................................................... 98
    3.4.2.4 ST Language ............................................................ 98
    3.4.2.5 FBD Language ....................................................... 98
    3.4.2.6 FFLD Language ...................................................... 98
    3.4.2.7 IL Language: .......................................................... 98
### 3.5.1.6 FFLD Language

*Language*: 105

### 3.5.1.7 IL Language:

*Language*: 105

### 3.5.1.8 See also

See also 105

### 3.5.2 ANY_TO_DINT / ANY_TO_UDINT

#### 3.5.2.1 Inputs

*Inputs*: 106

#### 3.5.2.2 Outputs

*Outputs*: 106

#### 3.5.2.3 Remarks

*Remarks*: 106

#### 3.5.2.4 ST Language

*ST Language*: 106

#### 3.5.2.5 FBD Language

*FBD Language*: 106

#### 3.5.2.6 FFLD Language

*FFLD Language*: 106

#### 3.5.2.7 IL Language:

*IL Language*: 106

#### 3.5.2.8 See also

See also 106

### 3.5.3 ANY_TO_INT / ANY_TO_UINT

#### 3.5.3.1 Inputs

*Inputs*: 107

#### 3.5.3.2 Outputs

*Outputs*: 107

#### 3.5.3.3 Remarks

*Remarks*: 107

#### 3.5.3.4 ST Language

*ST Language*: 107

#### 3.5.3.5 FBD Language

*FBD Language*: 107

#### 3.5.3.6 FFLD Language

*FFLD Language*: 107

#### 3.5.3.7 IL Language:

*IL Language*: 107

#### 3.5.3.8 See also

See also 107

### 3.5.4 ANY_TO_LINT / ANY_TO_ULINT

#### 3.5.4.1 Inputs

*Inputs*: 108

#### 3.5.4.2 Outputs

*Outputs*: 108

#### 3.5.4.3 Remarks

*Remarks*: 108

#### 3.5.4.4 ST Language

*ST Language*: 108

#### 3.5.4.5 FBD Language

*FBD Language*: 108

#### 3.5.4.6 FFLD Language

*FFLD Language*: 108

#### 3.5.4.7 IL Language:

*IL Language*: 108

#### 3.5.4.8 See also

See also 108

### 3.5.5 ANY_TO_LREAL

#### 3.5.5.1 Inputs

*Inputs*: 109

#### 3.5.5.2 Outputs

*Outputs*: 109

#### 3.5.5.3 Remarks

*Remarks*: 109

#### 3.5.5.4 ST Language

*ST Language*: 109

#### 3.5.5.5 FBD Language

*FBD Language*: 109

#### 3.5.5.6 FFLD Language

*FFLD Language*: 109

#### 3.5.5.7 IL Language:

*IL Language*: 109

#### 3.5.5.8 See also

See also 109

### 3.5.6 ANY_TO_REAL

#### 3.5.6.1 Inputs

*Inputs*: 110

#### 3.5.6.2 Outputs

*Outputs*: 110

#### 3.5.6.3 Remarks

*Remarks*: 110

#### 3.5.6.4 ST Language

*ST Language*: 110

#### 3.5.6.5 FBD Language

*FBD Language*: 110

#### 3.5.6.6 FFLD Language

*FFLD Language*: 110

#### 3.5.6.7 IL Language:

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

3.6 Selectors ................................................................. 117
3.6.1 MUX4 ................................................................. 118
  3.6.1.1 Inputs ............................................................ 118
  3.6.1.2 Outputs .......................................................... 118
  3.6.1.3 Truth table ...................................................... 118
  3.6.1.4 Remarks ........................................................ 118
  3.6.1.5 ST Language .................................................... 118
  3.6.1.6 FBD Language .................................................. 118
  3.6.1.7 FFLD Language ................................................ 118
  3.6.1.8 IL Language .................................................... 119
3.6.2 MUX8 ................................................................. 120
  3.6.2.1 Inputs ............................................................ 120
  3.6.2.2 Outputs .......................................................... 120
  3.6.2.3 Truth table ...................................................... 120
  3.6.2.4 Remarks ........................................................ 120
  3.6.2.5 ST Language .................................................... 120
  3.6.2.6 FBD Language .................................................. 120
  3.6.2.7 FFLD Language ................................................ 120
  3.6.2.8 IL Language .................................................... 121
3.6.3 SEL ................................................................. 122
  3.6.3.1 Inputs ............................................................ 122
  3.6.3.2 Outputs .......................................................... 122
  3.6.3.3 Truth table ...................................................... 122
  3.6.3.4 Remarks ........................................................ 122
  3.6.3.5 ST Language .................................................... 122
  3.6.3.6 FBD Language .................................................. 122
  3.6.3.7 FFLD Language ................................................ 122
  3.6.3.8 IL Language .................................................... 122

3.7 Registers .............................................................. 123
3.7.1 AND_MASK .......................................................... 124
  3.7.1.1 Inputs ............................................................ 124
  3.7.1.2 Outputs .......................................................... 124
  3.7.1.3 Remarks ........................................................ 124
  3.7.1.4 ST Language .................................................... 124
  3.7.1.5 FBD Language .................................................. 124
  3.7.1.6 FFLD Language ................................................ 124
  3.7.1.7 IL Language: ................................................... 124
3.7.2 HIBYTE ............................................................. 125
  3.7.2.1 Inputs ............................................................ 125
  3.7.2.2 Outputs .......................................................... 125
  3.7.2.3 Remarks ........................................................ 125
  3.7.2.4 ST Language .................................................... 125
  3.7.2.5 FBD Language .................................................. 125
  3.7.2.6 FFLD Language ................................................ 125
3.7.2.7  IL Language: ............................................................... 125
3.7.3 LOBYTE ................................................................. 126
  3.7.3.1 Inputs ............................................................... 126
  3.7.3.2 Outputs ............................................................. 126
  3.7.3.3 Remarks ............................................................ 126
  3.7.3.4 ST Language ...................................................... 126
  3.7.3.5 FBD Language ................................................... 126
  3.7.3.6 FFLD Language .................................................. 126
  3.7.3.7 IL Language: ..................................................... 126
3.7.4 HIWORD ................................................................. 127
  3.7.4.1 Inputs ............................................................... 127
  3.7.4.2 Outputs ............................................................. 127
  3.7.4.3 Remarks ............................................................ 127
  3.7.4.4 ST Language ...................................................... 127
  3.7.4.5 FBD Language ................................................... 127
  3.7.4.6 FFLD Language .................................................. 127
  3.7.4.7 IL Language: ..................................................... 127
3.7.5 LOWORD ................................................................. 128
  3.7.5.1 Inputs ............................................................... 128
  3.7.5.2 Outputs ............................................................. 128
  3.7.5.3 Remarks ............................................................ 128
  3.7.5.4 ST Language ...................................................... 128
  3.7.5.5 FBD Language ................................................... 128
  3.7.5.6 FFLD Language .................................................. 128
  3.7.5.7 IL Language: ..................................................... 128
3.7.6 MAKEDWORD ............................................................ 129
  3.7.6.1 Inputs ............................................................... 129
  3.7.6.2 Outputs ............................................................. 129
  3.7.6.3 Remarks ............................................................ 129
  3.7.6.4 ST Language ...................................................... 129
  3.7.6.5 FBD Language ................................................... 129
  3.7.6.6 FFLD Language .................................................. 129
  3.7.6.7 IL Language: ..................................................... 129
3.7.7 MAKEWORD ............................................................. 130
  3.7.7.1 Inputs ............................................................... 130
  3.7.7.2 Outputs ............................................................. 130
  3.7.7.3 Remarks ............................................................ 130
  3.7.7.4 ST Language ...................................................... 130
  3.7.7.5 FBD Language ................................................... 130
  3.7.7.6 FFLD Language .................................................. 130
  3.7.7.7 IL Language: ..................................................... 130
3.7.8 MBSHIFT ................................................................. 131
  3.7.8.1 Inputs ............................................................... 131
  3.7.8.2 Outputs ............................................................. 131
  3.7.8.3 Remarks ............................................................ 131
  3.7.8.4 ST Language ...................................................... 131
  3.7.8.5 FBD Language ................................................... 131
  3.7.8.6 FFLD Language .................................................. 131
  3.7.8.7 IL Language: ..................................................... 131
3.7.8.6  FFLD Language ............................................................... 131
3.7.8.7  IL Language: ............................................................... 131
3.7.9  NOT_MASK ................................................................. 132
  3.7.9.1  Inputs ................................................................. 132
  3.7.9.2  Outputs ............................................................... 132
  3.7.9.3  Remarks .............................................................. 132
  3.7.9.4  ST Language .......................................................... 132
  3.7.9.5  FBD Language ....................................................... 132
  3.7.9.6  FFLD Language ...................................................... 132
  3.7.9.7  IL Language: ........................................................ 132
3.7.10  OR_MASK ............................................................... 133
  3.7.10.1  Inputs ............................................................. 133
  3.7.10.2  Outputs ........................................................... 133
  3.7.10.3  Remarks ........................................................... 133
  3.7.10.4  ST Language ........................................................ 133
  3.7.10.5  FBD Language .................................................... 133
  3.7.10.6  FFLD Language .................................................. 133
  3.7.10.7  IL Language: ...................................................... 133
3.7.11  PACK8 ................................................................. 134
  3.7.11.1  Inputs ............................................................ 134
  3.7.11.2  Outputs .......................................................... 134
  3.7.11.3  Remarks .......................................................... 134
  3.7.11.4  ST Language ....................................................... 134
  3.7.11.5  FBD Language ................................................... 134
  3.7.11.6  FFLD Language .................................................. 135
  3.7.11.7  IL Language ....................................................... 135
3.7.12  ROL ................................................................. 136
  3.7.12.1  Inputs ............................................................ 136
  3.7.12.2  Outputs .......................................................... 136
  3.7.12.3  Diagram .......................................................... 136
  3.7.12.4  Remarks .......................................................... 136
  3.7.12.5  ST Language ....................................................... 136
  3.7.12.6  FBD Language ................................................... 136
  3.7.12.7  FFLD Language .................................................. 136
  3.7.12.8  IL Language: ...................................................... 136
3.7.13  ROR ................................................................. 137
  3.7.13.1  Inputs ............................................................ 137
  3.7.13.2  Outputs .......................................................... 137
  3.7.13.3  Diagram .......................................................... 137
  3.7.13.4  Remarks .......................................................... 137
  3.7.13.5  ST Language ....................................................... 137
  3.7.13.6  FBD Language ................................................... 137
  3.7.13.7  FFLD Language .................................................. 137
  3.7.13.8  IL Language: ...................................................... 137
3.7.14  RORb / ROR_SINT / ROR_USINT / ROR_BYTE ..................... 138
  3.7.14.1  Inputs ............................................................ 138
  3.7.14.2  Outputs .......................................................... 138
3.7.14.3 Diagram ................................................................. 138
3.7.14.4 Remarks .............................................................. 138
3.7.14.5 ST Language ......................................................... 138
3.7.14.6 FBD Language ...................................................... 138
3.7.14.7 FFLD Language .................................................... 138
3.7.14.8 IL Language: ....................................................... 138
3.7.14.9 See also ............................................................. 138
3.7.15 RORw / ROR_INT / ROR_UINT / ROR_WORD .............. 139
  3.7.15.1 Inputs .............................................................. 139
  3.7.15.2 Outputs ........................................................... 139
  3.7.15.3 Diagram ........................................................... 139
  3.7.15.4 Remarks .......................................................... 139
  3.7.15.5 ST Language ...................................................... 139
  3.7.15.6 FBD Language .................................................... 139
  3.7.15.7 FFLD Language ................................................ 139
  3.7.15.8 IL Language: .................................................... 139
  3.7.15.9 See also .......................................................... 139
3.7.16 SETBIT ............................................................... 140
  3.7.16.1 Inputs .............................................................. 140
  3.7.16.2 Outputs ........................................................... 140
  3.7.16.3 Remarks .......................................................... 140
  3.7.16.4 ST Language ...................................................... 140
  3.7.16.5 FBD Language .................................................... 140
  3.7.16.6 FFLD Language ................................................ 140
  3.7.16.7 IL Language: .................................................... 140
3.7.17 SHL ................................................................. 141
  3.7.17.1 Inputs .............................................................. 141
  3.7.17.2 Outputs ........................................................... 141
  3.7.17.3 Diagram ........................................................... 141
  3.7.17.4 Remarks .......................................................... 141
  3.7.17.5 ST Language ...................................................... 141
  3.7.17.6 FBD Language .................................................... 141
  3.7.17.7 FFLD Language ................................................ 141
  3.7.17.8 IL Language: .................................................... 141
3.7.18 SHR ................................................................. 142
  3.7.18.1 Inputs .............................................................. 142
  3.7.18.2 Outputs ........................................................... 142
  3.7.18.3 Diagram ........................................................... 142
  3.7.18.4 Remarks .......................................................... 142
  3.7.18.5 ST Language ...................................................... 142
  3.7.18.6 FBD Language .................................................... 142
  3.7.18.7 FFLD Language ................................................ 142
  3.7.18.8 IL Language: .................................................... 142
3.7.19 TESTBIT .......................................................... 143
  3.7.19.1 Inputs .............................................................. 143
  3.7.19.2 Outputs ........................................................... 143
  3.7.19.3 Remarks .......................................................... 143
3.7.19.4 ST Language ................................................................. 143
3.7.19.5 FBD Language ............................................................. 143
3.7.19.6 FFLD Language ............................................................ 143
3.7.19.7 IL Language ............................................................... 143
3.7.20 UNPACK8 ................................................................. 144
3.7.20.1 Inputs ................................................................. 144
3.7.20.2 Outputs ............................................................... 144
3.7.20.3 Remarks ............................................................... 144
3.7.20.4 ST Language ............................................................. 144
3.7.20.5 FBD Language .......................................................... 144
3.7.20.6 FFLD Language ........................................................ 144
3.7.20.7 IL Language: ........................................................ 145
3.7.21 XOR_MASK .............................................................. 146
3.7.21.1 Inputs ................................................................. 146
3.7.21.2 Outputs ............................................................... 146
3.7.21.3 Remarks ............................................................... 146
3.7.21.4 ST Language ............................................................. 146
3.7.21.5 FBD Language .......................................................... 146
3.7.21.6 FFLD Language ........................................................ 146
3.7.21.7 IL Language: ........................................................ 146
3.8 Counters ............................................................................... 147
3.8.1 CTD / CTDr ................................................................. 148
3.8.1.1 Inputs ................................................................. 148
3.8.1.2 Outputs ............................................................... 148
3.8.1.3 Remarks ............................................................... 148
3.8.1.4 ST Language ............................................................. 148
3.8.1.5 FBD Language .......................................................... 148
3.8.1.6 FFLD Language ........................................................ 148
3.8.1.7 IL Language: ........................................................ 148
3.8.2 CTU / CTUr ................................................................. 149
3.8.2.1 Inputs ................................................................. 149
3.8.2.2 Outputs ............................................................... 149
3.8.2.3 Remarks ............................................................... 149
3.8.2.4 ST Language ............................................................. 149
3.8.2.5 FBD Language .......................................................... 149
3.8.2.6 FFLD Language ........................................................ 149
3.8.2.7 IL Language: ........................................................ 149
3.8.3 CTUD / CTUDr .......................................................... 150
3.8.3.1 Inputs ................................................................. 150
3.8.3.2 Outputs ............................................................... 150
3.8.3.3 Remarks ............................................................... 150
3.8.3.4 ST Language ............................................................. 150
3.8.3.5 FBD Language .......................................................... 150
3.8.3.6 FFLD Language ........................................................ 151
3.8.3.7 IL Language: ........................................................ 151
3.9 Timers ............................................................................... 152
3.9.1 BLINK ........................................................................... 153
# Table of Contents

3.9.1.1 Inputs ................................................................. 153
3.9.1.2 Outputs ................................................................. 153
3.9.1.3 Time diagram .......................................................... 153
3.9.1.4 Remarks ................................................................. 153
3.9.1.5 ST Language ............................................................ 153
3.9.1.6 FBD Language ........................................................ 153
3.9.1.7 FFLD Language ....................................................... 153
3.9.1.8 IL Language ............................................................ 153
3.9.2 BLINKA ................................................................. 154
  3.9.2.1 Inputs .................................................................. 154
  3.9.2.2 Outputs ................................................................. 154
  3.9.2.3 Time diagram .......................................................... 154
  3.9.2.4 Remarks ................................................................. 154
  3.9.2.5 ST Language ............................................................ 154
  3.9.2.6 FBD Language ........................................................ 154
  3.9.2.7 FFLD Language ....................................................... 154
  3.9.2.8 IL Language ............................................................ 155
3.9.3 PLS ................................................................. 156
  3.9.3.1 Inputs .................................................................. 156
  3.9.3.2 Outputs ................................................................. 156
  3.9.3.3 Time diagram .......................................................... 156
  3.9.3.4 Remarks ................................................................. 156
  3.9.3.5 ST Language ............................................................ 156
  3.9.3.6 FBD Language ........................................................ 156
  3.9.3.7 FFLD Language ....................................................... 156
  3.9.3.8 IL Language ............................................................ 157
3.9.4 Sig_Gen ............................................................... 158
  3.9.4.1 Inputs .................................................................. 158
  3.9.4.2 Outputs ................................................................. 158
  3.9.4.3 FFLD Language ....................................................... 158
3.9.5 TMD ................................................................. 159
  3.9.5.1 Inputs .................................................................. 159
  3.9.5.2 Outputs ................................................................. 159
  3.9.5.3 Time diagram .......................................................... 159
  3.9.5.4 Remarks ................................................................. 159
  3.9.5.5 ST Language ............................................................ 159
  3.9.5.6 FBD Language ........................................................ 159
  3.9.5.7 FFLD Language ....................................................... 159
  3.9.5.8 IL Language ............................................................ 160
3.9.6 TMU / TMUsec .................................................. 161
  3.9.6.1 Inputs .................................................................. 161
  3.9.6.2 Outputs ................................................................. 161
  3.9.6.3 Time diagram .......................................................... 161
  3.9.6.4 Remarks ................................................................. 161
  3.9.6.5 ST Language ............................................................ 161
  3.9.6.6 FBD Language ........................................................ 161
  3.9.6.7 FFLD Language ....................................................... 162
3.9.6.8 IL Language: .......................................................... 162
3.9.7 TOF / TOFR .......................................................... 163
  3.9.7.1 Inputs .......................................................... 163
  3.9.7.2 Outputs .......................................................... 163
  3.9.7.3 Time diagram .................................................. 163
  3.9.7.4 Remarks .......................................................... 163
  3.9.7.5 ST Language .................................................... 163
  3.9.7.6 FBD Language .................................................. 163
  3.9.7.7 FFLD Language .................................................. 163
  3.9.7.8 IL Language: .................................................... 164
3.9.8 TON ...................................................................... 165
  3.9.8.1 Inputs .......................................................... 165
  3.9.8.2 Outputs .......................................................... 165
  3.9.8.3 Time diagram .................................................. 165
  3.9.8.4 Remarks .......................................................... 165
  3.9.8.5 ST Language .................................................... 165
  3.9.8.6 FBD Language .................................................. 165
  3.9.8.7 FFLD Language .................................................. 166
  3.9.8.8 IL Language: .................................................... 166
3.9.9 TP / TPR .............................................................. 167
  3.9.9.1 Inputs .......................................................... 167
  3.9.9.2 Outputs .......................................................... 167
  3.9.9.3 Time diagram .................................................. 167
  3.9.9.4 Remarks .......................................................... 167
  3.9.9.5 ST Language .................................................... 167
  3.9.9.6 FBD Language .................................................. 167
  3.9.9.7 FFLD Language .................................................. 167
  3.9.9.8 IL Language: .................................................... 168
3.10 Mathematic operations .............................................. 169
  3.10.1 ABS / ABSL ....................................................... 170
    3.10.1.1 Inputs .......................................................... 170
    3.10.1.2 Outputs .......................................................... 170
    3.10.1.3 Remarks .......................................................... 170
    3.10.1.4 ST Language .................................................... 170
    3.10.1.5 FBD Language .................................................. 170
    3.10.1.6 FFLD Language .................................................. 170
    3.10.1.7 IL Language: .................................................... 170
  3.10.2 EXPT ............................................................. 171
    3.10.2.1 Inputs .......................................................... 171
    3.10.2.2 Outputs .......................................................... 171
    3.10.2.3 Remarks .......................................................... 171
    3.10.2.4 ST Language .................................................... 171
    3.10.2.5 FBD Language .................................................. 171
    3.10.2.6 FFLD Language .................................................. 171
    3.10.2.7 IL Language: .................................................... 171
  3.10.3 EXP / EXPL ....................................................... 172
    3.10.3.1 Inputs .......................................................... 172
<table>
<thead>
<tr>
<th>Section</th>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.10.3.2</td>
<td>Outputs</td>
<td>172</td>
</tr>
<tr>
<td>3.10.3.3</td>
<td>Remarks</td>
<td>172</td>
</tr>
<tr>
<td>3.10.3.4</td>
<td>ST Language</td>
<td>172</td>
</tr>
<tr>
<td>3.10.3.5</td>
<td>FBD Language</td>
<td>172</td>
</tr>
<tr>
<td>3.10.3.6</td>
<td>FFLD Language</td>
<td>172</td>
</tr>
<tr>
<td>3.10.3.7</td>
<td>IL Language</td>
<td>172</td>
</tr>
<tr>
<td>3.10.4</td>
<td>LOG / LOGL</td>
<td>173</td>
</tr>
<tr>
<td>3.10.4.1</td>
<td>Inputs</td>
<td>173</td>
</tr>
<tr>
<td>3.10.4.2</td>
<td>Outputs</td>
<td>173</td>
</tr>
<tr>
<td>3.10.4.3</td>
<td>Remarks</td>
<td>173</td>
</tr>
<tr>
<td>3.10.4.4</td>
<td>ST Language</td>
<td>173</td>
</tr>
<tr>
<td>3.10.4.5</td>
<td>FBD Language</td>
<td>173</td>
</tr>
<tr>
<td>3.10.4.6</td>
<td>FFLD Language</td>
<td>173</td>
</tr>
<tr>
<td>3.10.4.7</td>
<td>IL Language</td>
<td>173</td>
</tr>
<tr>
<td>3.10.5</td>
<td>LN / LNL</td>
<td>174</td>
</tr>
<tr>
<td>3.10.5.1</td>
<td>Inputs</td>
<td>174</td>
</tr>
<tr>
<td>3.10.5.2</td>
<td>Outputs</td>
<td>174</td>
</tr>
<tr>
<td>3.10.5.3</td>
<td>Remarks</td>
<td>174</td>
</tr>
<tr>
<td>3.10.5.4</td>
<td>ST Language</td>
<td>174</td>
</tr>
<tr>
<td>3.10.5.5</td>
<td>FBD Language</td>
<td>174</td>
</tr>
<tr>
<td>3.10.5.6</td>
<td>FFLD Language</td>
<td>174</td>
</tr>
<tr>
<td>3.10.5.7</td>
<td>IL Language</td>
<td>174</td>
</tr>
<tr>
<td>3.10.6</td>
<td>POW ** POWL</td>
<td>175</td>
</tr>
<tr>
<td>3.10.6.1</td>
<td>Inputs</td>
<td>175</td>
</tr>
<tr>
<td>3.10.6.2</td>
<td>Outputs</td>
<td>175</td>
</tr>
<tr>
<td>3.10.6.3</td>
<td>Remarks</td>
<td>175</td>
</tr>
<tr>
<td>3.10.6.4</td>
<td>ST Language</td>
<td>175</td>
</tr>
<tr>
<td>3.10.6.5</td>
<td>FBD Language</td>
<td>175</td>
</tr>
<tr>
<td>3.10.6.6</td>
<td>FFLD Language</td>
<td>175</td>
</tr>
<tr>
<td>3.10.6.7</td>
<td>IL Language</td>
<td>175</td>
</tr>
<tr>
<td>3.10.7</td>
<td>ROOT</td>
<td>176</td>
</tr>
<tr>
<td>3.10.7.1</td>
<td>Inputs</td>
<td>176</td>
</tr>
<tr>
<td>3.10.7.2</td>
<td>Outputs</td>
<td>176</td>
</tr>
<tr>
<td>3.10.7.3</td>
<td>Remarks</td>
<td>176</td>
</tr>
<tr>
<td>3.10.7.4</td>
<td>ST Language</td>
<td>176</td>
</tr>
<tr>
<td>3.10.7.5</td>
<td>FBD Language</td>
<td>176</td>
</tr>
<tr>
<td>3.10.7.6</td>
<td>FFLD Language</td>
<td>176</td>
</tr>
<tr>
<td>3.10.7.7</td>
<td>IL Language</td>
<td>176</td>
</tr>
<tr>
<td>3.10.8</td>
<td>ScaleLin</td>
<td>177</td>
</tr>
<tr>
<td>3.10.8.1</td>
<td>Inputs</td>
<td>177</td>
</tr>
<tr>
<td>3.10.8.2</td>
<td>Outputs</td>
<td>177</td>
</tr>
<tr>
<td>3.10.8.3</td>
<td>Truth table</td>
<td>177</td>
</tr>
<tr>
<td>3.10.8.4</td>
<td>Remarks</td>
<td>177</td>
</tr>
<tr>
<td>3.10.8.5</td>
<td>ST Language</td>
<td>177</td>
</tr>
<tr>
<td>3.10.8.6</td>
<td>FBD Language</td>
<td>177</td>
</tr>
<tr>
<td>3.10.8.7</td>
<td>FFLD Language</td>
<td>178</td>
</tr>
<tr>
<td>3.10.8.8</td>
<td>IL Language</td>
<td>178</td>
</tr>
</tbody>
</table>
3.10.9 SQRT / SQRTL .......................................................... 179
  3.10.9.1 Inputs .......................................................... 179
  3.10.9.2 Outputs ....................................................... 179
  3.10.9.3 Remarks ...................................................... 179
  3.10.9.4 ST Language ............................................... 179
  3.10.9.5 FBD Language ............................................. 179
  3.10.9.6 FFLD Language ........................................... 179
  3.10.9.7 IL Language: .............................................. 179
3.10.10 TRUNC / TRUNCL ................................................. 180
  3.10.10.1 Inputs .................................................... 180
  3.10.10.2 Outputs .................................................. 180
  3.10.10.3 Remarks .................................................. 180
  3.10.10.4 ST Language ............................................. 180
  3.10.10.5 FBD Language .......................................... 180
  3.10.10.6 FFLD Language ......................................... 180
  3.10.10.7 IL Language: ........................................... 180
3.11 Trigonometric functions ............................................ 181
  3.11.1 ACOS / ACOSL ................................................ 182
    3.11.1.1 Inputs .................................................. 182
    3.11.1.2 Outputs ................................................ 182
    3.11.1.3 Remarks ............................................... 182
    3.11.1.4 ST Language .......................................... 182
    3.11.1.5 FBD Language ....................................... 182
    3.11.1.6 FFLD Language ..................................... 182
    3.11.1.7 IL Language: ........................................ 182
  3.11.2 ASIN / ASINL ................................................ 183
    3.11.2.1 Inputs ................................................ 183
    3.11.2.2 Outputs .............................................. 183
    3.11.2.3 Remarks ............................................... 183
    3.11.2.4 ST Language .......................................... 183
    3.11.2.5 FBD Language ....................................... 183
    3.11.2.6 FFLD Language ..................................... 183
    3.11.2.7 IL Language: ........................................ 183
  3.11.3 ATAN / ATANL ................................................ 184
    3.11.3.1 Inputs ................................................ 184
    3.11.3.2 Outputs .............................................. 184
    3.11.3.3 Remarks ............................................... 184
    3.11.3.4 ST Language .......................................... 184
    3.11.3.5 FBD Language ....................................... 184
    3.11.3.6 FFLD Language ..................................... 184
    3.11.3.7 IL Language: ........................................ 184
  3.11.4 ATAN2 / ATAN2L .............................................. 185
    3.11.4.1 Inputs ................................................ 185
    3.11.4.2 Outputs .............................................. 185
    3.11.4.3 Remarks ............................................... 185
    3.11.4.4 ST Language .......................................... 185
    3.11.4.5 FBD Language ....................................... 185
3.11.6.7 IL Language: .................................................. 187
3.12 String operations .................................................. 190
3.12.1 ArrayToString / ArrayToStringU ................................ 191
3.12.1.1 Inputs .................................................. 191
3.12.1.2 Outputs .................................................. 191
3.12.1.3 Remarks .................................................. 191
3.12.1.4 ST Language .................................................. 191
3.12.1.5 FBD Language .................................................. 191
3.12.1.6 FFLD Language .................................................. 191
3.12.1.7 IL Language .................................................. 191
3.12.2 ASCII .................................................. 192
3.12.2.1 Inputs .................................................. 192
3.12.2.2 Outputs .................................................. 192
3.12.2.3 Remarks .................................................. 192
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.12.2.4</td>
<td>ST Language</td>
<td>192</td>
</tr>
<tr>
<td>3.12.2.5</td>
<td>FBD Language</td>
<td>192</td>
</tr>
<tr>
<td>3.12.2.6</td>
<td>FFLD Language</td>
<td>192</td>
</tr>
<tr>
<td>3.12.2.7</td>
<td>IL Language</td>
<td>192</td>
</tr>
<tr>
<td>3.12.3</td>
<td>ATOH</td>
<td>193</td>
</tr>
<tr>
<td>3.12.3.1</td>
<td>Inputs</td>
<td>193</td>
</tr>
<tr>
<td>3.12.3.2</td>
<td>Outputs</td>
<td>193</td>
</tr>
<tr>
<td>3.12.3.3</td>
<td>Truth table (examples)</td>
<td>193</td>
</tr>
<tr>
<td>3.12.3.4</td>
<td>Remarks</td>
<td>193</td>
</tr>
<tr>
<td>3.12.3.5</td>
<td>ST Language</td>
<td>193</td>
</tr>
<tr>
<td>3.12.3.6</td>
<td>FBD Language</td>
<td>193</td>
</tr>
<tr>
<td>3.12.3.7</td>
<td>FFLD Language</td>
<td>193</td>
</tr>
<tr>
<td>3.12.3.8</td>
<td>IL Language</td>
<td>193</td>
</tr>
<tr>
<td>3.12.4</td>
<td>CHAR</td>
<td>194</td>
</tr>
<tr>
<td>3.12.4.1</td>
<td>Inputs</td>
<td>194</td>
</tr>
<tr>
<td>3.12.4.2</td>
<td>Outputs</td>
<td>194</td>
</tr>
<tr>
<td>3.12.4.3</td>
<td>Remarks</td>
<td>194</td>
</tr>
<tr>
<td>3.12.4.4</td>
<td>ST Language</td>
<td>194</td>
</tr>
<tr>
<td>3.12.4.5</td>
<td>FBD Language</td>
<td>194</td>
</tr>
<tr>
<td>3.12.4.6</td>
<td>FFLD Language</td>
<td>194</td>
</tr>
<tr>
<td>3.12.4.7</td>
<td>IL Language</td>
<td>194</td>
</tr>
<tr>
<td>3.12.5</td>
<td>CONCAT</td>
<td>195</td>
</tr>
<tr>
<td>3.12.5.1</td>
<td>Inputs</td>
<td>195</td>
</tr>
<tr>
<td>3.12.5.2</td>
<td>Outputs</td>
<td>195</td>
</tr>
<tr>
<td>3.12.5.3</td>
<td>Remarks</td>
<td>195</td>
</tr>
<tr>
<td>3.12.5.4</td>
<td>ST Language</td>
<td>195</td>
</tr>
<tr>
<td>3.12.5.5</td>
<td>FBD Language</td>
<td>195</td>
</tr>
<tr>
<td>3.12.5.6</td>
<td>FFLD Language</td>
<td>195</td>
</tr>
<tr>
<td>3.12.5.7</td>
<td>IL Language</td>
<td>195</td>
</tr>
<tr>
<td>3.12.6</td>
<td>CRC16</td>
<td>196</td>
</tr>
<tr>
<td>3.12.6.1</td>
<td>Inputs</td>
<td>196</td>
</tr>
<tr>
<td>3.12.6.2</td>
<td>Outputs</td>
<td>196</td>
</tr>
<tr>
<td>3.12.6.3</td>
<td>Remarks</td>
<td>196</td>
</tr>
<tr>
<td>3.12.6.4</td>
<td>ST Language</td>
<td>196</td>
</tr>
<tr>
<td>3.12.6.5</td>
<td>FBD Language</td>
<td>196</td>
</tr>
<tr>
<td>3.12.6.6</td>
<td>FFLD Language</td>
<td>196</td>
</tr>
<tr>
<td>3.12.6.7</td>
<td>IL Language</td>
<td>196</td>
</tr>
<tr>
<td>3.12.7</td>
<td>DELETE</td>
<td>197</td>
</tr>
<tr>
<td>3.12.7.1</td>
<td>Inputs</td>
<td>197</td>
</tr>
<tr>
<td>3.12.7.2</td>
<td>Outputs</td>
<td>197</td>
</tr>
<tr>
<td>3.12.7.3</td>
<td>Remarks</td>
<td>197</td>
</tr>
<tr>
<td>3.12.7.4</td>
<td>ST Language</td>
<td>197</td>
</tr>
<tr>
<td>3.12.7.5</td>
<td>FBD Language</td>
<td>197</td>
</tr>
<tr>
<td>3.12.7.6</td>
<td>FFLD Language</td>
<td>197</td>
</tr>
<tr>
<td>3.12.7.7</td>
<td>IL Language</td>
<td>197</td>
</tr>
<tr>
<td>3.12.8</td>
<td>FIND</td>
<td>198</td>
</tr>
<tr>
<td>3.12.8.1</td>
<td>Inputs</td>
<td>198</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>3.12.8.2 Outputs</td>
<td>198</td>
<td></td>
</tr>
<tr>
<td>3.12.8.3 Remarks</td>
<td>198</td>
<td></td>
</tr>
<tr>
<td>3.12.8.4 ST Language</td>
<td>198</td>
<td></td>
</tr>
<tr>
<td>3.12.8.5 FBD Language</td>
<td>198</td>
<td></td>
</tr>
<tr>
<td>3.12.8.6 FFLD Language</td>
<td>198</td>
<td></td>
</tr>
<tr>
<td>3.12.8.7 IL Language:</td>
<td>198</td>
<td></td>
</tr>
<tr>
<td>3.12.9 HTOA</td>
<td>199</td>
<td></td>
</tr>
<tr>
<td>3.12.9.1 Inputs</td>
<td>199</td>
<td></td>
</tr>
<tr>
<td>3.12.9.2 Outputs</td>
<td>199</td>
<td></td>
</tr>
<tr>
<td>3.12.9.3 Truth table (examples)</td>
<td>199</td>
<td></td>
</tr>
<tr>
<td>3.12.9.4 Remarks</td>
<td>199</td>
<td></td>
</tr>
<tr>
<td>3.12.9.5 ST Language</td>
<td>199</td>
<td></td>
</tr>
<tr>
<td>3.12.9.6 FBD Language</td>
<td>199</td>
<td></td>
</tr>
<tr>
<td>3.12.9.7 FFLD Language</td>
<td>199</td>
<td></td>
</tr>
<tr>
<td>3.12.9.8 IL Language:</td>
<td>199</td>
<td></td>
</tr>
<tr>
<td>3.12.10 INSERT</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>3.12.10.1 Inputs</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>3.12.10.2 Outputs</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>3.12.10.3 Remarks</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>3.12.10.4 ST Language</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>3.12.10.5 FBD Language</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>3.12.10.6 FFLD Language</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>3.12.10.7 IL Language:</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>3.12.11 LEFT</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>3.12.11.1 Inputs</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>3.12.11.2 Outputs</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>3.12.11.3 Remarks</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>3.12.11.4 ST Language</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>3.12.11.5 FBD Language</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>3.12.11.6 FFLD Language</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>3.12.11.7 IL Language:</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>3.12.12 LoadString</td>
<td>202</td>
<td></td>
</tr>
<tr>
<td>3.12.12.1 Inputs</td>
<td>202</td>
<td></td>
</tr>
<tr>
<td>3.12.12.2 Outputs</td>
<td>202</td>
<td></td>
</tr>
<tr>
<td>3.12.12.3 Remarks</td>
<td>202</td>
<td></td>
</tr>
<tr>
<td>3.12.12.4 ST Language</td>
<td>202</td>
<td></td>
</tr>
<tr>
<td>3.12.12.5 FBD Language</td>
<td>202</td>
<td></td>
</tr>
<tr>
<td>3.12.12.6 FFLD Language</td>
<td>202</td>
<td></td>
</tr>
<tr>
<td>3.12.12.7 IL Language:</td>
<td>202</td>
<td></td>
</tr>
<tr>
<td>3.12.13 MID</td>
<td>203</td>
<td></td>
</tr>
<tr>
<td>3.12.13.1 Inputs</td>
<td>203</td>
<td></td>
</tr>
<tr>
<td>3.12.13.2 Outputs</td>
<td>203</td>
<td></td>
</tr>
<tr>
<td>3.12.13.3 Remarks</td>
<td>203</td>
<td></td>
</tr>
<tr>
<td>3.12.13.4 ST Language</td>
<td>203</td>
<td></td>
</tr>
<tr>
<td>3.12.13.5 FBD Language</td>
<td>203</td>
<td></td>
</tr>
<tr>
<td>3.12.13.6 FFLD Language</td>
<td>203</td>
<td></td>
</tr>
<tr>
<td>3.12.13.7 IL Language:</td>
<td>203</td>
<td></td>
</tr>
</tbody>
</table>
3.12.14 MLEN ................................................................. 204
3.12.14.1 Inputs .......................................................... 204
3.12.14.2 Outputs ........................................................ 204
3.12.14.3 Remarks ....................................................... 204
3.12.14.4 ST Language ............................................... 204
3.12.14.5 FBD Language ............................................ 204
3.12.14.6 FFLD Language ........................................... 204
3.12.14.7 IL Language: ............................................... 204
3.12.15 REPLACE ......................................................... 205
3.12.15.1 Inputs .......................................................... 205
3.12.15.2 Outputs ........................................................ 205
3.12.15.3 Remarks ....................................................... 205
3.12.15.4 ST Language ............................................... 205
3.12.15.5 FBD Language ............................................ 205
3.12.15.6 FFLD Language ........................................... 205
3.12.15.7 IL Language: ............................................... 205
3.12.16 RIGHT ............................................................. 206
3.12.16.1 Inputs .......................................................... 206
3.12.16.2 Outputs ........................................................ 206
3.12.16.3 Remarks ....................................................... 206
3.12.16.4 ST Language ............................................... 206
3.12.16.5 FBD Language ............................................ 206
3.12.16.6 FFLD Language ........................................... 206
3.12.16.7 IL Language: ............................................... 206
3.12.17 StringTable ....................................................... 207
3.12.17.1 Inputs .......................................................... 207
3.12.17.2 Outputs ........................................................ 207
3.12.17.3 Remarks ....................................................... 207
3.12.17.4 ST Language ............................................... 207
3.12.17.5 FBD Language ............................................ 207
3.12.17.6 FFLD Language ........................................... 207
3.12.17.7 IL Language: ............................................... 207
3.12.18 StringToArray / StringToArrayU ................................ 208
3.12.18.1 Inputs .......................................................... 208
3.12.18.2 Outputs ........................................................ 208
3.12.18.3 Remarks ....................................................... 208
3.12.18.4 ST Language ............................................... 208
3.12.18.5 FBD Language ............................................ 208
3.12.18.6 FFLD Language ........................................... 208
3.12.18.7 IL Language: ............................................... 208
3.13 UDP Functions for PDMM/PCMM & Simulator .................. 209
3.13.1 udpAddrMake ................................................... 210
3.13.1.1 Description ................................................ 210
3.13.1.2 Arguments .................................................. 210
3.13.1.5 Examples .................................................... 211
3.13.2 udpClose .......................................................... 212
3.13.2.1 Description ................................................ 212
3.13.2.2 Arguments .......................................................... 212
3.13.2.5 Examples .......................................................... 213
3.13.3 udpCreate ............................................................. 214
  3.13.3.1 Description ....................................................... 214
  3.13.3.2 Arguments ....................................................... 214
  3.13.3.5 Examples ....................................................... 215
3.13.4 udplValid ............................................................. 216
  3.13.4.1 Description ....................................................... 216
  3.13.4.2 Arguments ....................................................... 216
  3.13.4.5 Examples ....................................................... 217
3.13.5 udpRcvFrom .......................................................... 218
  3.13.5.1 Description ....................................................... 218
  3.13.5.2 Arguments ....................................................... 218
  3.13.5.5 Examples ....................................................... 219
3.13.6 udpRcvFromArray .................................................... 220
  3.13.6.1 Description ....................................................... 220
  3.13.6.2 Arguments ....................................................... 220
  3.13.6.5 Examples ....................................................... 221
3.13.7 udpSendTo ............................................................. 222
  3.13.7.1 Description ....................................................... 222
  3.13.7.2 Arguments ....................................................... 222
  3.13.7.5 Examples ....................................................... 223
3.13.8 udpSendToArray ...................................................... 224
  3.13.8.1 Description ....................................................... 224
  3.13.8.2 Arguments ....................................................... 224
  3.13.8.5 Examples ....................................................... 225
4 PLC Advanced Libraries .................................................. 226
  4.1 ALARM_A ................................................................. 228
    4.1.1 Inputs ............................................................. 228
    4.1.2 Outputs ............................................................ 228
    4.1.3 Sequence .......................................................... 228
    4.1.4 Remarks ............................................................ 228
    4.1.5 ST Language ....................................................... 228
    4.1.6 FBD Language ...................................................... 228
    4.1.7 FFLD Language ..................................................... 229
    4.1.8 IL Language ....................................................... 229
  4.2 ALARM_M ................................................................. 230
    4.2.1 Inputs ............................................................. 230
    4.2.2 Outputs ............................................................ 230
    4.2.3 Sequence .......................................................... 230
    4.2.4 Remarks ............................................................ 230
    4.2.5 ST Language ....................................................... 230
    4.2.6 FBD Language ...................................................... 230
    4.2.7 FFLD Language ..................................................... 231
    4.2.8 IL Language ....................................................... 231
  4.3 ApplyRecipeColumn ................................................... 232
    4.3.1 Inputs ............................................................. 232
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3.2</td>
<td>Outputs</td>
<td>233</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Remarks</td>
<td>233</td>
</tr>
<tr>
<td>4.3.4</td>
<td>ST Language</td>
<td>233</td>
</tr>
<tr>
<td>4.3.5</td>
<td>FBD Language</td>
<td>235</td>
</tr>
<tr>
<td>4.3.6</td>
<td>FFLD Language</td>
<td>235</td>
</tr>
<tr>
<td>4.3.7</td>
<td>IL Language</td>
<td>235</td>
</tr>
<tr>
<td>4.4</td>
<td>AS-interface Functions</td>
<td>236</td>
</tr>
<tr>
<td>4.5</td>
<td>AVERAGE / AVERAGEL</td>
<td>237</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Inputs</td>
<td>237</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Outputs</td>
<td>237</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Remarks</td>
<td>237</td>
</tr>
<tr>
<td>4.5.4</td>
<td>ST Language</td>
<td>237</td>
</tr>
<tr>
<td>4.5.5</td>
<td>FBD Language</td>
<td>237</td>
</tr>
<tr>
<td>4.5.6</td>
<td>FFLD Language</td>
<td>237</td>
</tr>
<tr>
<td>4.5.7</td>
<td>IL Language</td>
<td>238</td>
</tr>
<tr>
<td>4.6</td>
<td>CurveLin</td>
<td>239</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Inputs</td>
<td>239</td>
</tr>
<tr>
<td>4.6.2</td>
<td>Outputs</td>
<td>239</td>
</tr>
<tr>
<td>4.6.3</td>
<td>Remarks</td>
<td>239</td>
</tr>
<tr>
<td>4.7</td>
<td>DERIVATE</td>
<td>240</td>
</tr>
<tr>
<td>4.7.1</td>
<td>Inputs</td>
<td>240</td>
</tr>
<tr>
<td>4.7.2</td>
<td>Outputs</td>
<td>240</td>
</tr>
<tr>
<td>4.7.3</td>
<td>Remarks</td>
<td>240</td>
</tr>
<tr>
<td>4.7.4</td>
<td>ST Language</td>
<td>240</td>
</tr>
<tr>
<td>4.7.5</td>
<td>FBD Language</td>
<td>240</td>
</tr>
<tr>
<td>4.7.6</td>
<td>FFLD Language</td>
<td>240</td>
</tr>
<tr>
<td>4.7.7</td>
<td>IL Language</td>
<td>240</td>
</tr>
<tr>
<td>4.8</td>
<td>EnableEvents</td>
<td>241</td>
</tr>
<tr>
<td>4.8.1</td>
<td>Inputs</td>
<td>241</td>
</tr>
<tr>
<td>4.8.2</td>
<td>Outputs</td>
<td>241</td>
</tr>
<tr>
<td>4.8.3</td>
<td>Remarks</td>
<td>241</td>
</tr>
<tr>
<td>4.8.4</td>
<td>ST Language</td>
<td>241</td>
</tr>
<tr>
<td>4.8.5</td>
<td>FBD Language</td>
<td>241</td>
</tr>
<tr>
<td>4.8.6</td>
<td>FFLD Language</td>
<td>241</td>
</tr>
<tr>
<td>4.8.7</td>
<td>IL Language</td>
<td>241</td>
</tr>
<tr>
<td>4.9</td>
<td>FIFO</td>
<td>242</td>
</tr>
<tr>
<td>4.9.1</td>
<td>Inputs</td>
<td>242</td>
</tr>
<tr>
<td>4.9.2</td>
<td>Outputs</td>
<td>242</td>
</tr>
<tr>
<td>4.9.3</td>
<td>Remarks</td>
<td>242</td>
</tr>
<tr>
<td>4.9.4</td>
<td>ST Language</td>
<td>242</td>
</tr>
<tr>
<td>4.9.5</td>
<td>FBD Language</td>
<td>242</td>
</tr>
<tr>
<td>4.9.6</td>
<td>FFLD Language</td>
<td>243</td>
</tr>
<tr>
<td>4.9.7</td>
<td>IL Language</td>
<td>243</td>
</tr>
<tr>
<td>4.10</td>
<td>FilterOrder1</td>
<td>243</td>
</tr>
<tr>
<td>4.10.1</td>
<td>Inputs</td>
<td>243</td>
</tr>
<tr>
<td>4.10.2</td>
<td>Outputs</td>
<td>244</td>
</tr>
<tr>
<td>4.10.3</td>
<td>Remarks</td>
<td>244</td>
</tr>
</tbody>
</table>
4.10.4 ST Language ................................................................. 244
4.10.5 Example ................................................................. 244
4.11 File Management .......................................................... 245
  4.11.1 SD Card Access .................................................. 247
  4.11.2 File Path Conventions ........................................ 248
    4.11.2.1 Shared Directory Path Conventions ................. 249
    4.11.2.2 SD Card Path Conventions ............................ 250
    4.11.2.3 File Name Warning & Limitations .................. 251
  4.11.3 File Management Functions ................................. 252
    4.11.3.1 F_AOPEN ................................................ 252
    4.11.3.2 F_CLOSE ............................................... 252
    4.11.3.3 F_COPY ............................................. 252
    4.11.3.4 F_DELETE ........................................... 253
    4.11.3.5 F_EOF ................................................ 253
    4.11.3.6 F_EXIST ............................................. 253
    4.11.3.7 F_GETSIZE .......................................... 253
    4.11.3.8 F_RENAME ........................................... 253
    4.11.3.9 F_ROPEN ............................................ 254
    4.11.3.10 F_SEEK ............................................ 254
    4.11.3.14 F_WOPEN ........................................... 254
    4.11.3.15 FA_READ ........................................... 255
    4.11.3.16 FA_WRITE ........................................... 255
    4.11.3.17 FB_READ ............................................ 255
    4.11.3.18 FB_WRITE .......................................... 255
    4.11.3.19 FM_READ ........................................... 256
    4.11.3.20 FM_WRITE ......................................... 256
    4.11.3.21 SD_MOUNT ........................................... 257
    4.11.3.22 SD_UNMOUNT ........................................ 257
    4.11.3.23 SD_ISREADY ......................................... 257
  4.11.4 File Management Function Examples .................. 258
4.12 GETSYSINFO ............................................................. 259
  4.12.1 Inputs ........................................................... 259
  4.12.2 Outputs .......................................................... 259
  4.12.3 Remarks .......................................................... 259
  4.12.4 ST Language .................................................... 259
  4.12.5 FBD Language .................................................. 259
  4.12.6 FFLD Language ................................................. 260
  4.12.7 IL Language: ................................................... 260
4.13 HYSTER ................................................................. 261
  4.13.1 Inputs ........................................................... 261
  4.13.2 Outputs .......................................................... 261
  4.13.3 Remarks .......................................................... 261
  4.13.4 ST Language .................................................... 261
  4.13.5 FBD Language .................................................. 261
  4.13.6 FFLD Language ................................................. 261
  4.13.7 IL Language: ................................................... 261
4.14 INTEGRAL ............................................................... 262
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.14.1</td>
<td>Inputs</td>
</tr>
<tr>
<td>4.14.2</td>
<td>Outputs</td>
</tr>
<tr>
<td>4.14.3</td>
<td>Remarks</td>
</tr>
<tr>
<td>4.14.4</td>
<td>ST Language</td>
</tr>
<tr>
<td>4.14.5</td>
<td>FBD Language</td>
</tr>
<tr>
<td>4.14.6</td>
<td>FFLD Language</td>
</tr>
<tr>
<td>4.14.7</td>
<td>IL Language</td>
</tr>
<tr>
<td>4.15</td>
<td>LIFO</td>
</tr>
<tr>
<td>4.15.1</td>
<td>Inputs</td>
</tr>
<tr>
<td>4.15.2</td>
<td>Outputs</td>
</tr>
<tr>
<td>4.15.3</td>
<td>Remarks</td>
</tr>
<tr>
<td>4.15.4</td>
<td>ST Language</td>
</tr>
<tr>
<td>4.15.5</td>
<td>FBD Language</td>
</tr>
<tr>
<td>4.15.6</td>
<td>FFLD Language</td>
</tr>
<tr>
<td>4.15.7</td>
<td>IL Language</td>
</tr>
<tr>
<td>4.16</td>
<td>LIM_ALRM</td>
</tr>
<tr>
<td>4.16.1</td>
<td>Inputs</td>
</tr>
<tr>
<td>4.16.2</td>
<td>Outputs</td>
</tr>
<tr>
<td>4.16.3</td>
<td>Remarks</td>
</tr>
<tr>
<td>4.16.4</td>
<td>ST Language</td>
</tr>
<tr>
<td>4.16.5</td>
<td>FBD Language</td>
</tr>
<tr>
<td>4.16.6</td>
<td>FFLD Language</td>
</tr>
<tr>
<td>4.16.7</td>
<td>IL Language</td>
</tr>
<tr>
<td>4.17</td>
<td>LogFileCSV</td>
</tr>
<tr>
<td>4.17.1</td>
<td>Inputs</td>
</tr>
<tr>
<td>4.17.2</td>
<td>Outputs</td>
</tr>
<tr>
<td>4.17.3</td>
<td>Remarks</td>
</tr>
<tr>
<td>4.17.4</td>
<td>ST Language</td>
</tr>
<tr>
<td>4.17.5</td>
<td>FBD Language</td>
</tr>
<tr>
<td>4.17.6</td>
<td>FFLD Language</td>
</tr>
<tr>
<td>4.17.7</td>
<td>IL Language</td>
</tr>
<tr>
<td>4.18</td>
<td>PID</td>
</tr>
<tr>
<td>4.18.1</td>
<td>Inputs</td>
</tr>
<tr>
<td>4.18.2</td>
<td>Outputs</td>
</tr>
<tr>
<td>4.18.3</td>
<td>Diagram</td>
</tr>
<tr>
<td>4.18.4</td>
<td>Remarks</td>
</tr>
<tr>
<td>4.18.5</td>
<td>ST Language</td>
</tr>
<tr>
<td>4.18.6</td>
<td>FBD Language</td>
</tr>
<tr>
<td>4.18.7</td>
<td>FFLD Language</td>
</tr>
<tr>
<td>4.18.8</td>
<td>IL Language</td>
</tr>
<tr>
<td>4.19</td>
<td>PWM</td>
</tr>
<tr>
<td>4.19.1</td>
<td>Inputs</td>
</tr>
<tr>
<td>4.19.2</td>
<td>Outputs</td>
</tr>
<tr>
<td>4.19.3</td>
<td>Remarks</td>
</tr>
<tr>
<td>4.19.4</td>
<td>ST Language</td>
</tr>
<tr>
<td>4.19.5</td>
<td>Example</td>
</tr>
<tr>
<td>4.20</td>
<td>RAMP</td>
</tr>
</tbody>
</table>

4.20 RAMP
4.20.1 Inputs .............................................................. 276
4.20.2 Outputs ............................................................. 276
4.20.3 Time diagram ....................................................... 276
4.20.4 Remarks ............................................................ 276
4.20.5 ST Language ....................................................... 276
4.20.6 FBD Language ..................................................... 276
4.20.7 FFLD Language .................................................... 277
4.20.8 IL Language ....................................................... 277

4.21 Real Time Clock Management Functions .......... 278

4.21.1 DAY_TIME ....................................................... 280
  4.21.1.1 Inputs .......................................................... 280
  4.21.1.2 Outputs ......................................................... 280
  4.21.1.3 Remarks ........................................................ 280
  4.21.1.4 ST Language .................................................. 280
  4.21.1.5 FBD Language ................................................. 280
  4.21.1.6 FFLD Language ................................................. 280
  4.21.1.7 IL Language: .................................................. 280

4.21.2 DTFORMAT ....................................................... 281
  4.21.2.1 Inputs .......................................................... 281
  4.21.2.2 Outputs ......................................................... 281
  4.21.2.3 Remarks ........................................................ 281
  4.21.2.4 ST Language .................................................. 281
  4.21.2.5 FBD Language ................................................. 281
  4.21.2.6 FFLD Language ................................................. 281
  4.21.2.7 IL Language: .................................................. 282

4.21.3 DTAT .............................................................. 283
  4.21.3.1 Inputs .......................................................... 283
  4.21.3.2 Outputs ......................................................... 283
  4.21.3.3 Remarks ........................................................ 283
  4.21.3.4 ST Language .................................................. 283
  4.21.3.5 FBD Language ................................................. 283
  4.21.3.6 FFLD Language ................................................. 284
  4.21.3.7 IL Language: .................................................. 284

4.21.4 DTEVERY ......................................................... 285
  4.21.4.1 Inputs .......................................................... 285
  4.21.4.2 Outputs ......................................................... 285
  4.21.4.3 Remarks ........................................................ 285
  4.21.4.4 ST Language .................................................. 285
  4.21.4.5 FBD Language ................................................. 285
  4.21.4.6 FFLD Language ................................................. 285
  4.21.4.7 IL Language: .................................................. 285

4.22 Serializeln ........................................................ 286
  4.22.1 Description ...................................................... 286
  4.22.2 Arguments ....................................................... 286
    4.22.2.1 Input .......................................................... 286
    4.22.2.2 Output ........................................................ 287
  4.22.3 Examples ....................................................... 287
<table>
<thead>
<tr>
<th>4.29.3 Remarks</th>
<th>297</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.29.4 ST Language</td>
<td>297</td>
</tr>
<tr>
<td>4.29.5 FBD Language</td>
<td>297</td>
</tr>
<tr>
<td>4.29.6 FFLD Language</td>
<td>297</td>
</tr>
<tr>
<td>4.29.7 IL Language</td>
<td>297</td>
</tr>
</tbody>
</table>
2 Programming Languages

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

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

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

Use of ST instructions in graphic languages

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

2.1 Sequential Function Chart (SFC)

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

**IMPORTANT**

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

Below are basic components of an SFC chart:

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

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

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

- "Hierarchy of SFC programs" (p. 34)
- Controlling a SFC child program

2.1.1 SFC Execution at Runtime

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

Within a chart, all valid transitions are evaluated first, and then actions of active steps are performed. The chart is evaluated from the left to the right and from the top to the bottom. Below is an example:
In case of a divergence, all conditions are considered as exclusive, according to a "left to right" priority order. It means that a transition is considered as FALSE if at least one of the transitions connected to the same divergence on its left side is TRUE.

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

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

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

- When a transition is crossed:
  - all steps linked to the top of the transition (before) are deactivated
  - all steps linked to the bottom of the transition (after) are activated

**IMPORTANT**

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

**IMPORTANT**

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

### 2.1.2 Hierarchy of SFC programs

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

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

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

A child program is controlled (started or stopped) from the action blocks of its parent program. Designing a child program is a simple way to program an action block in SFC language.
Using child programs is very useful for designing a complex process and separate operations due to different aspects of the process. For instance, it is common to manage the execution modes in a parent program and to handle details of the process operations in child programs.

2.2 Free Form Ladder Diagram (FFLD)

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

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

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

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

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

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

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

IN1 IN2 = 1 ( ) Q
```

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

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

EN IN1 IN2 > ( ) Q
```

- Below is the example of the "SEL" function, having a first Boolean input, but an integer output. This block has an "ENO" output in FFLD language:
Finally, below is the example of an addition, having only numerical arguments. This block has both "EN" and "ENO" pins in FFLD language:

2.2.2 Contacts and coils

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

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

2.2.2.1 FFLD Contacts

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

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

<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>
</tbody>
</table>
Contacts | Description
---|---
`boolVariable` | Negated: The flow on the right is the Boolean AND operation between:
(1) the flow on the left and (2) the negation of the associated variable.

`boolVariable` | Positive Transition: The flow on the right is TRUE when the flow on the left is TRUE and the associated variable is TRUE and was FALSE the last time this contact was scanned (rising edge).

`boolVariable` | Negative Transition: The flow on the right is TRUE when the flow on the left is TRUE and the associated variable is FALSE and was TRUE last time this contact was scanned (falling edge).

`boolVariable` | Normally Closed Positive Transition: The flow on the right is TRUE when the flow on the left is TRUE and the associated variable does not change from FALSE to TRUE from the last scan of this contact to this scan (NOT rising edge).

`boolVariable` | Normally Closed Negative Transition: The flow on the right is TRUE when the flow on the left is TRUE and the associated variable does not change from TRUE to FALSE from the last scan of this contact to this scan (NOT falling edge).

Serialized and Parallel contacts
Two serial normal contacts represent an AND operation.

Two contacts in parallel represent an OR operation.

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

For example:
MC_GrpEnable executes and turns on its Done output immediately. In the following code, the GroupEnableDone positive transition contact will only provide a TRUE the first time MC_GrpEnable is executed. For all subsequent executions, the positive transition contact will not provide a TRUE since GroupEnableDone will be TRUE every time the contact is scanned.
To remedy this, the following code uses the SET and RESET of a Boolean (i.e. EnableRequest) to provide a way to detect each successful execution of the function block:

When a contact or coil is selected, you can press the **Spacebar** to change its type (normal, negated...)
When your application is running, you can select a contact and press the **Spacebar** to swap its value between TRUE and FALSE

### 2.2.2.2 FFLD Coils

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

Below are the six possible coil symbols:

<table>
<thead>
<tr>
<th>Coils</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolVariable ( )</td>
<td><strong>Normal:</strong> the associated variable is forced to the value of the flow on the left of the coil.</td>
</tr>
<tr>
<td>boolVariable ( / )</td>
<td><strong>Negated:</strong> the associated variable is forced to the negation of the flow on the left of the coil.</td>
</tr>
</tbody>
</table>
### Coils

<table>
<thead>
<tr>
<th>Description</th>
<th>Rules for Set coil animation:</th>
</tr>
</thead>
</table>
| **boolVariable** Set: the associated variable is forced to TRUE if the flow on the left is TRUE. (no action if the flow is FALSE) | - Power Flow on left is TRUE:  
  - The horizontal wires on either side of the (S) are red  
  - The variable and the (S) are red  
  - In all other cases:  
    - The horizontal wires are black  
    - The variable and the (S) are black |
| **boolVariable** Reset: the associated variable is forced to FALSE if the flow on the left is TRUE. (no action if the rung state is FALSE) | - 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 |
| **boolVariable** Positive transition: the associated variable is forced to TRUE if the flow on the left changes from FALSE to TRUE (and forced to FALSE in all other cases) | **boolVariable** Negative transition: the associated variable is forced to TRUE if the flow on the left changes from TRUE to FALSE (and forced to FALSE in all other cases) |

**TIP**

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

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

**IMPORTANT**

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

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

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

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

Below are the language features for basic data manipulation:

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

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

- Labels
- Jumps
- RETURN

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

<table>
<thead>
<tr>
<th>IF</th>
<th>Conditional execution of statements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHILE</td>
<td>Repeat statements while a condition is TRUE.</td>
</tr>
<tr>
<td>REPEAT</td>
<td>Repeat statements until a condition is TRUE.</td>
</tr>
<tr>
<td>FOR</td>
<td>Execute iterations of statements.</td>
</tr>
<tr>
<td>CASE</td>
<td>Switch to one of various possible statements.</td>
</tr>
<tr>
<td>EXIT</td>
<td>Exit from a loop instruction.</td>
</tr>
<tr>
<td>WAIT</td>
<td>Delay program execution.</td>
</tr>
<tr>
<td>ON</td>
<td>Conditional execution.</td>
</tr>
</tbody>
</table>
3.1.1 := \textbf{PLCopen} \checkmark

Operator - variable assignment.

3.1.1.1 Inputs

\textbf{IN} : \textbf{ANY} \ Any variable or complex expression

3.1.1.2 Outputs

\textbf{Q} : \textbf{ANY} \ Forced variable

3.1.1.3 Remarks

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

3.1.1.4 ST Language

\begin{verbatim}
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 *)
\end{verbatim}

3.1.1.5 FBD Language

\begin{center}
\includegraphics[width=0.5\textwidth]{fbd_diagram.png}
\end{center}

3.1.1.6 FFLD Language

\begin{verbatim}
(* The copy is executed only if EN is TRUE *)

\begin{center}
\begin{array}{c}
1 \quad 2 \quad \text{Assign\textit{OK}} \\
\hline
\text{En} \quad \text{OK} \quad \text{Assign\textit{OK}} \\
\text{IN} \quad \text{Q} \quad \text{Assign\textit{OK}} \\
\end{array}
\end{center}
\end{verbatim}

3.1.1.7 IL Language:

\begin{verbatim}
Op1: FFLD IN (* current result is: IN *)
ST Q (* Q is: IN *)
FFLDN IN1 (* current result is: NOT (IN1) *)
ST Q (* Q is: NOT (IN1) *)
FFLD IN2 (* current result is: IN2 *)
STN Q (* Q is: NOT (IN2) *)
\end{verbatim}

\textbf{See also:}

Parenthesis
3.1.2 Access to bits of an integer

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

    Variable.BitNo

Where:

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

The variable can have one of the following data types:

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

0 always represents the less significant bit.
3.1.3 Differences Between Functions and Function Blocks

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

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

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

\[\text{Function} \quad \text{(1)} \]
\[\text{Function Block} \quad \text{(2)} \]

---

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

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

3.1.4.1 ST Language

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

```plaintext
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:

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

3.1.4.2 FBD and FFLD Languages

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

3.1.4.3 IL Language

To call a sub-program in IL language, you must use the CAL instruction with the name of the sub-program, followed by the input parameters written between parentheses and separated by 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>

Here is an example:

```plaintext
Op1: CAL MySubProg (i1, i2)
FFLD MySubProg.Q1
ST Res1
FFLD MySubProg.Q2
ST Res2
```
3.1.5 CASE OF ELSE END_CASE

*Statement* - switch between enumerated statements.

### 3.1.5.1 Syntax

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

### 3.1.5.2 Remarks

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

### 3.1.5.3 ST Language

(* this example check first prime numbers *)

```plaintext
CASE iNumber OF
  0 :
    Alarm := TRUE;
    AlarmText := '0 gives no result';
  1 .. 3, 5 :
    bPrime := TRUE;
  4, 6 :
    bPrime := FALSE;
ELSE
  Alarm := TRUE;
  AlarmText := 'I don't know after 6!';
END_CASE;
```

### 3.1.5.4 FBD Language

Not available

### 3.1.5.5 FFLD Language

Not available

### 3.1.5.6 IL Language

Not available

See also

IF  WHILE  REPEAT  FOR  EXIT
3.1.6 COUNTOF

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

**3.1.6.1 Inputs**

ARR : ANY  Declared array

**3.1.6.2 Outputs**

Q : DINT  Total number of items in the array

**3.1.6.3 Remarks**

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

Example:

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

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

**Examples**

<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>

**3.1.6.4 ST Language**

Q := CountOf (ARR);

**3.1.6.5 FBD Language**

```
CountOf
Arr[]
```

**3.1.6.6 FFLD Language**

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

(* ENO keeps the same value as EN *)

**3.1.6.7 IL Language**

*Not available*
3.1.7 DEC

*Function* - Decrease a numerical variable

### 3.1.7.1 Inputs

IN : ANY   Numerical variable (increased after call).

### 3.1.7.2 Outputs

Q : ANY   Decreased value

### 3.1.7.3 Remarks

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

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

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

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

### 3.1.7.4 ST Language

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

### 3.1.7.5 FBD Language

![FBD Diagram]

### 3.1.7.6 FFLD Language

![FFLD Diagram]

### 3.1.7.7 IL Language

*not available*
3.1.8 EXIT

Statement - Exit from a loop statement

3.1.8.1 Remarks

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

⚠️ IMPORTANT

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

3.1.8.2 ST Language

(* this program searches for the first non null item of an array *)

iFound = -1; (* means: not found *)
FOR iPos := 0 TO (iArrayDim - 1) DO
  IF iPos <> 0 THEN
    iFound := iPos;
    EXIT;
  END_IF;
END_FOR;

3.1.8.3 FBD Language

Not available

3.1.8.4 FFLD Language

Not available

3.1.8.5 IL Language

Not available

See also

IF WHILE REPEAT FOR CASE
3.1.9 FOR TO BY END_FOR

Statement - Iteration of statement execution.

3.1.9.1 Syntax

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

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

3.1.9.2 Remarks

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

3.1.9.3 ST Language

```
iArrayDim := 10;

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

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

3.1.9.4 FBD Language

*Not available*

3.1.9.5 FFLD Language

*Not available*

3.1.9.6 IL Language

*Not available*

See also

IF WHILE REPEAT CASE EXIT
3.1.10 IF THEN ELSE ELSIF END_IF

Statement - Conditional execution of statements.

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

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

3.1.10.3 ST Language

(* simple condition *)

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

(* binary selection *)

IF bCond THEN
  Q1 := IN1;
  Q2 := TRUE;
ELSE
  Q1 := IN2;
  Q2 := FALSE;
END_IF;

(* enumerated conditions *)

IF bCond1 THEN
  Q1 := IN1;
ELSIF bCond2 THEN
  Q1 := IN2;
ELSIF bCond3 THEN
  Q1 := IN3;
ELSE
  Q1 := IN4;
END_IF;

3.1.10.4 FBD Language

Not available

3.1.10.5 FFLD Language

Not available
3.1.10.6 IL Language

*Not available*

**See also**

WHILE REPEAT FOR CASE EXIT
3.1.11 INC

*Function* - Increase a numerical variable

### 3.1.11.1 Inputs

IN : ANY Numerical variable (increased after call).

### 3.1.11.2 Outputs

Q : ANY Increased value

### 3.1.11.3 Remarks

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

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

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

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

### 3.1.11.4 ST Language

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

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

### 3.1.11.5 FBD Language

### 3.1.11.6 FFLD Language

### 3.1.11.7 IL Language

*not available*
3.1.12 MOVEBLOCK

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

### 3.1.12.1 Inputs

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

(*) SRC and DST cannot be a STRING

### 3.1.12.2 Outputs

OK : BOOL TRUE if successful

### 3.1.12.3 Remarks

Arrays of string are not supported by this function.

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

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

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

### 3.1.12.4 ST Language

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

### 3.1.12.5 FBD Language

![FBD diagram]

### 3.1.12.6 FFLD Language
(* The function is executed only if EN is TRUE *)

3.1.12.7 IL Language

_Not available_
### 3.1.13 NEG - \( \text{PLCopen} \)

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

#### 3.1.13.1 Inputs

| IN : DINT | Integer value |

#### 3.1.13.2 Outputs

| Q : DINT | Integer negation of the input |

#### 3.1.13.3 Truth table (examples)

<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>

#### 3.1.13.4 Remarks

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

#### 3.1.13.5 ST Language

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

#### 3.1.13.6 FBD Language

![FBD Diagram]

#### 3.1.13.7 FFLD Language

("The negation is executed only if EN is TRUE")

("ENO keeps the same value as EN")

![FFLD Diagram]
3.1.14 ON

Statement - Conditional execution of statements.

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

3.1.14.1 Syntax

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

3.1.14.2 Remarks

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

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

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

⚠️ IMPORTANT

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

3.1.14.3 ST Language

```
(* This example counts the rising edges of variable bIN *)
ON bIN DO
   diCount := diCount + 1;
END_DO;
```
3.1.15 ( )

Operator - force the evaluation order in a complex expression.

3.1.15.1 Remarks

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

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

<table>
<thead>
<tr>
<th>Unary operators</th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiply/Divide</td>
<td>*/</td>
</tr>
<tr>
<td>Add/Subtract</td>
<td>+ -</td>
</tr>
<tr>
<td>Comparisons</td>
<td>&lt; &lt;= &gt;= = &lt;&gt;</td>
</tr>
<tr>
<td>Boolean And</td>
<td>&amp; AND</td>
</tr>
<tr>
<td>Boolean Or</td>
<td>OR</td>
</tr>
<tr>
<td>Exclusive OR</td>
<td>XOR</td>
</tr>
</tbody>
</table>

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

3.1.15.2 ST Language

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

3.1.15.3 FBD Language

Not available

3.1.15.4 FFLD Language

Not available

3.1.15.5 IL Language

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

See also
Assignment
3.1.16 REPEAT UNTIL END_REPEAT

Statement - Repeat a list of statements.

3.1.16.1 Syntax

REPEAT
  <statements>
UNTIL <BOOL expression> END_REPEAT;

3.1.16.2 Remarks

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

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

3.1.16.3 ST Language

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

3.1.16.4 FBD Language

Not available

3.1.16.5 FFLD Language

Not available

3.1.16.6 IL Language

Not available

See also

IF  WHILE  FOR  CASE  EXIT
3.1.17 RETURN  RET  RETC  RETNC  RETCN

Statement - Jump to the end of the program.

3.1.17.1 Remarks

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

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

3.1.17.2 ST Language

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

3.1.17.3 FBD Language

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

3.1.17.4 FFLD Language

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

Below is the meaning of possible instructions:

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

```
Start: FFLD IN1
       RETC (* Jump to the end if IN1 is TRUE *)
       FFLD IN2 (* these instructions are not executed *)
       ST Q2  (* if IN1 is TRUE *)
       RET  (* Jump to the end unconditionally *)
       FFLD IN3 (* these instructions are never executed *)
       ST Q3
```

See also

Labels | Jumps
3.1.18 WAIT / WAIT_TIME

Statement - Suspend the execution of a ST program.

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

3.1.18.1 Syntax

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

3.1.18.2 Remarks

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

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

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

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

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

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

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

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

**IMPORTANT**

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

3.1.18.3 ST Language

```plaintext
(* use of WAIT with different kinds of BOOL expressions *)
WAIT BoolVariable;
WAIT (dilevel > 100) AND NOT bAlarm;
WAIT SubProgCall ();
(* use of WAIT_TIME with different kinds of TIME expressions *)
WAIT_TIME t#2s;
WAIT_TIME TimeVariable;
```
3.1.19 WHILE DO END WHILE

Statement - Repeat a list of statements.

3.1.19.1 Syntax

WHILE <BOOL expression> DO
   <statements>
END WHILE;

3.1.19.2 Remarks

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

⚠️ IMPORTANT

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

3.1.19.3 ST Language

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

3.1.19.4 FBD Language

Not available

3.1.19.5 FFLD Language

Not available

3.1.19.6 IL Language

Not available

See also

IF  REPEAT  FOR  CASE  EXIT
3.2 Boolean operations

Below are the standard operators for managing Booleans:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>performs a Boolean AND</td>
</tr>
<tr>
<td>OR</td>
<td>performs a Boolean OR</td>
</tr>
<tr>
<td>XOR</td>
<td>performs an exclusive OR</td>
</tr>
<tr>
<td>NOT</td>
<td>performs a Boolean negation of its input</td>
</tr>
<tr>
<td>&quot;QOR&quot; (p. 70)</td>
<td>qualified OR</td>
</tr>
<tr>
<td>S</td>
<td>force a Boolean output to TRUE</td>
</tr>
<tr>
<td>R</td>
<td>force a Boolean output to FALSE</td>
</tr>
</tbody>
</table>

Below are the available blocks for managing Boolean signals:

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td>reset dominant bistable</td>
</tr>
<tr>
<td>SR</td>
<td>set dominant bistable</td>
</tr>
<tr>
<td>R_TRIG</td>
<td>rising pulse detection</td>
</tr>
<tr>
<td>F_TRIG</td>
<td>falling pulse detection</td>
</tr>
<tr>
<td>SEMA</td>
<td>semaphore</td>
</tr>
<tr>
<td>FLIPFLOP</td>
<td>flipflop*bistable</td>
</tr>
</tbody>
</table>
3.2.1 FLIPFLOP

*Function Block* - Flipflop bistable.

3.2.1.1 Inputs

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

3.2.1.2 Outputs

Q : BOOL   Output

3.2.1.3 Remarks

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

3.2.1.4 ST Language

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

3.2.1.5 FBD Language

3.2.1.6 FFLD Language

3.2.1.7 IL Language

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

See also

R  S  SR
3.2.2 F_TRIG

*Function Block* - Falling pulse detection

3.2.2.1 Inputs

CLK : BOOL  Boolean signal

3.2.2.2 Outputs

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

3.2.2.3 Truth table

<table>
<thead>
<tr>
<th>CLK</th>
<th>CLK 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>

3.2.2.4 Remarks

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

3.2.2.5 ST Language

(* MyTrigger is declared as an instance of F_TRIG function block *)

MyTrigger (CLK);
Q := MyTrigger.Q;

3.2.2.6 FBD Language

3.2.2.7 FFLD Language
3.2.2.8 IL Language:

(* MyTrigger is declared as an instance of F_TRIG function block *)

Op1: CAL MyTrigger (CLK)
LD MyTrigger.Q
ST Q

See also

R_TRIG
3.2.3 NOT

Operator - Performs a Boolean negation of the input.

3.2.3.1 Inputs
IN : BOOL  Boolean value

3.2.3.2 Outputs
Q : BOOL  Boolean negation of the input

3.2.3.3 Truth table

<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>

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

3.2.3.5 ST Language
Q := NOT IN;
Q := NOT (IN1 OR IN2);

3.2.3.6 FBD Language
(* explicit use of the "NOT" block *)

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

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

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

Op1: FFLDN IN1
    OR IN2
    ST Q (* Q is equal to: (NOT IN1) OR IN2 *)

Op2: FFLD IN1
    AND IN2
    STN Q (* Q is equal to: NOT (IN1 AND IN2) *)

See also
AND OR XOR
3.2.4 **QOR**

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

### 3.2.4.1 Inputs

\[ \text{IN1} \ldots \text{INn} : \text{BOOL} \] Boolean inputs

### 3.2.4.2 Outputs

\[ Q : \text{DINT} \] Number of inputs being TRUE

### 3.2.4.3 Remarks

The block accepts a non-fixed number of inputs.

### 3.2.4.4 ST Language

\[
Q := \text{QOR (IN1, IN2)}; \\
Q := \text{QOR (IN1, IN2, IN3, IN4, IN5, IN6)};
\]

### 3.2.4.5 FBD Language

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

![FBD Diagram]

### 3.2.4.6 FFLD Language

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

![FFLD Diagram]

### 3.2.4.7 IL Language

**Op1:**

\[
\text{LD IN1} \\
\text{QOR IN2, IN3} \\
\text{ST Q}
\]
3.2.5 R

Operator - Force a Boolean output to FALSE.

3.2.5.1 Inputs
RESET : BOOL Condition

3.2.5.2 Outputs
Q : BOOL Output to be forced

3.2.5.3 Truth table

<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>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

3.2.5.5 ST Language
Not available.

3.2.5.6 FBD Language
Not available. Use RS or SR function blocks.

3.2.5.7 FFLD Language

(* use of "R" coil *)

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

See also
S  RS  SR
3.2.6 RS

*Function Block* - Reset dominant bistable.

### 3.2.6.1 Inputs

**SET** : BOOL  Condition for forcing to TRUE

**RESET1** : BOOL  Condition for forcing to FALSE (highest priority command)

### 3.2.6.2 Outputs

**Q1** : BOOL  Output to be forced

### 3.2.6.3 Truth table

<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>

### 3.2.6.4 Remarks

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

### 3.2.6.5 ST Language

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

### 3.2.6.6 FBD Language

[Diagram of FBD]
3.2.6.7 FFLD Language

3.2.6.8 IL Language:

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

See also
R S SR
3.2.7 R_TRIG

Function Block - Rising pulse detection

3.2.7.1 Inputs
CLK : BOOL Boolean signal

3.2.7.2 Outputs
Q : BOOL TRUE when the input changes from FALSE to TRUE

3.2.7.3 Truth table

<table>
<thead>
<tr>
<th>CLK</th>
<th>CLK 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>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

3.2.7.5 ST Language

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

3.2.7.6 FBD Language
3.2.7.7 FFLD Language

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

3.2.7.8 IL Language:

(* MyTrigger is declared as an instance of R_TRIG function block *

Op1: CAL MyTrigger (CLK)
FFLD MyTrigger.Q
ST Q

See also
F_TRIG
3.2.8 S

Operator - Force a Boolean output to TRUE.

3.2.8.1 Inputs

SET : BOOL Condition

3.2.8.2 Outputs

Q : BOOL Output to be forced

3.2.8.3 Truth table

<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>

3.2.8.4 Remarks

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

3.2.8.5 ST Language

Not available.

3.2.8.6 FBD Language

Not available. Use RS or SR function blocks.

3.2.8.7 FFLD Language

(* use of "S" coil *)

3.2.8.8 IL Language:

Op1: FFLD SET

S Q (* Q is forced to TRUE if SET is TRUE *)

(* Q is unchanged if SET is FALSE *)

See also

R RS SR
3.2.9 SEMA

*Function Block* - Semaphore.

### 3.2.9.1 Inputs

- **CLAIM** : BOOL  Takes the semaphore
- **RELEASE** : BOOL  Releases the semaphore

### 3.2.9.2 Outputs

- **BUSY** : BOOL  True if semaphore is busy

### 3.2.9.3 Remarks

The function block implements the following algorithm:

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

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

### 3.2.9.4 ST Language

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

```plaintext
MySema (CLAIM, RELEASE);
BUSY := MyBlinker.BUSY;
```

### 3.2.9.5 FBD Language

![FBD Diagram](image)

### 3.2.9.6 FFLD Language

![FFLD Diagram](image)

### 3.2.9.7 IL Language:

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

```plaintext
Op1: CAL MySema (CLAIM, RELEASE)
    FFLD  MyBlinker.BUSY
    ST  BUSY
```


3.2.10 SR

*Function Block* - Set dominant bistable.

### 3.2.10.1 Inputs

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

### 3.2.10.2 Outputs

- **Q1**: BOOL Output to be forced

### 3.2.10.3 Truth table

<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>

### 3.2.10.4 Remarks

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

### 3.2.10.5 ST Language

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

### 3.2.10.6 FBD Language

```
SET1  --SR
      |
      --Q1
RESET
```
3.2.10.7 FFLD Language

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

\[
\text{SET1} \quad \text{SR} \quad Q1 \\
\text{RESET} \\
\]

3.2.10.8 IL Language:

(* MySR is declared as an instance of SR function block *)

Op1: CAL MySR (SET1, RESET)

  FFLD  MySR.Q1
  ST    Q1

See also

R  S  RS
### 3.2.11 XOR XORN

Operator - Performs an exclusive OR of all inputs.

#### 3.2.11.1 Inputs

- **IN1**: BOOL  First Boolean input
- **IN2**: BOOL  Second Boolean input

#### 3.2.11.2 Outputs

- **Q**: BOOL  Exclusive OR of all inputs

#### 3.2.11.3 Truth table

<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>
<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>0</td>
</tr>
</tbody>
</table>

#### 3.2.11.4 Remarks

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

#### 3.2.11.5 ST Language

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

#### 3.2.11.6 FBD Language

```
  =1
IN1
IN2
```
```
  =1
IN1
IN2
IN3
```

#### 3.2.11.7 FFLD Language

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

```
IN1 [ ] [ ] [ ]
IN2
```
```
=1
```
3.2.11.8 IL Language

Op1: FFLD IN1
    XOR IN2
    ST Q (* Q is equal to: IN1 XOR IN2 *)

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

See also

AND OR NOT
3.3 Arithmetic operations

Below are the standard operators that perform arithmetic operations:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+</code></td>
<td>Addition (p. 83)</td>
</tr>
<tr>
<td><code>-</code></td>
<td>Subtraction (p. 95)</td>
</tr>
<tr>
<td><code>*</code></td>
<td>Multiply (p. 92)</td>
</tr>
<tr>
<td><code>/</code></td>
<td>Divide (p. 85)</td>
</tr>
<tr>
<td><code>(NEG)</code></td>
<td>Integer negation (unary operator)</td>
</tr>
</tbody>
</table>

Below are the standard functions that perform arithmetic operations:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MIN</code></td>
<td>Get the minimum of two integers or an ANY</td>
</tr>
<tr>
<td><code>MAX</code></td>
<td>Get the maximum of two integers or an ANY</td>
</tr>
<tr>
<td><code>LIMIT</code></td>
<td>Bound an integer to low and high limits or an ANY</td>
</tr>
<tr>
<td><code>MOD</code></td>
<td>Modulo</td>
</tr>
<tr>
<td><code>ODD</code></td>
<td>Test if an integer is odd</td>
</tr>
<tr>
<td><code>&quot;SetWithin&quot;</code> (p. 94)</td>
<td>Force a value when within an interval</td>
</tr>
</tbody>
</table>
3.3.1 + Addition

Operator - Performs an addition of all inputs.

3.3.1.1 Inputs

<table>
<thead>
<tr>
<th>IN1</th>
<th>ANY</th>
<th>First input</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN2</td>
<td>ANY</td>
<td>Second input</td>
</tr>
</tbody>
</table>

3.3.1.2 Outputs

| Q     | ANY  | Result: IN1 + IN2 |

3.3.1.3 Remarks

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

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

3.3.1.4 ST Language

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

3.3.1.5 FBD Language

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

3.3.1.6 FFLD Language

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

(* ENO is equal to EN *)
3.3.1.7 IL Language

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

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

See also
- *!
3.3.2 / Divide

Operator - Performs a division of inputs.

3.3.2.1 Inputs

IN1 : ANY_NUM   First input
IN2 : ANY_NUM   Second input

3.3.2.2 Outputs

Q : ANY_NUM   Result: IN1 / IN2

3.3.2.3 Remarks

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

3.3.2.4 ST Language

Q := IN1 / IN2;

3.3.2.5 FBD Language

```
IN1   /   Q
IN2   
```

3.3.2.6 FFLD Language

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

```
EN    /
|     |
| IN1 |
| IN2 |
|     |
| Q   |
| EMO |
```

3.3.2.7 IL Language:

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

See also

+ - *
3.3.3 NEG -

Operator - Performs an integer negation of the input.

3.3.3.1 Inputs

| IN | DINT | Integer value |

3.3.3.2 Outputs

| Q | DINT | Integer negation of the input |

3.3.3.3 Truth table (examples)

<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>

3.3.3.4 Remarks

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

3.3.3.5 ST Language

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

3.3.3.6 FBD Language

3.3.3.7 FFLD Language

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

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

3.3.4.1 Inputs

IMIN: DINT  Low bound
IN: DINT  Input value
IMAX: DINT  High bound

3.3.4.2 Outputs

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

3.3.4.3 Function diagram

3.3.4.4 Remarks

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

3.3.4.5 ST Language

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

3.3.4.6 FBD Language

3.3.4.7 FFLD Language

(* The comparison is executed only if EN is TRUE *)
(* ENO has the same value as EN *)
3.3.4.8 IL Language:

Op1: LD  IMIN
    LIMIT IN, IMAX
    ST  Q

See also
MIN  MAX  MOD  ODD
3.3.5 MAX

Function - Get the maximum of two integers.

3.3.5.1 Inputs
IN1 : DINT  First input
IN2 : DINT  Second input

3.3.5.2 Outputs
Q : DINT  IN1 if IN1 > IN2; IN2 otherwise

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

3.3.5.4 ST Language
Q := MAX (IN1, IN2);

3.3.5.5 FBD Language

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

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

See also
MIN  LIMIT  MOD  ODD
3.3.6 **MIN**

*Function* - Get the minimum of two integers.

3.3.6.1 **Inputs**

IN1 : DINT  First input  
IN2 : DINT  Second input

3.3.6.2 **Outputs**

Q : DINT  IN1 if IN1 < IN2; IN2 otherwise

3.3.6.3 **Remarks**

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

3.3.6.4 **ST Language**

Q := MIN (IN1, IN2);

3.3.6.5 **FBD Language**

```
MIN
IN1
IN2
```

3.3.6.6 **FFLD Language**

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

```
[ EN ]
[ ]
MIN
( )
ENO
IN1
IN2
```

3.3.6.7 **IL Language:**

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

**See also**

MAX  LIMIT  MOD  ODD
### 3.3.7 MOD / MODR / MODLR

*Function - Calculation of modulo.*

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>DINT</td>
<td>REAL</td>
</tr>
<tr>
<td>BASE</td>
<td>DINT</td>
<td>REAL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td>REAL</td>
</tr>
</tbody>
</table>

**Remarks**

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

#### 3.3.7.2 ST Language

\[
Q := \text{MOD}(\text{IN}, \text{BASE});
\]

#### 3.3.7.3 FBD Language

![FBD Diagram](image)

#### 3.3.7.4 FFLD Language

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

(* ENO has the same value as EN *)

#### 3.3.7.5 IL Language

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

**See also**

MIN MAX LIMIT ODD
3.3.8 * Multiply

*Operator* - Performs a multiplication of all inputs.

3.3.8.1 Inputs

IN1 : ANY_NUM  First input
IN2 : ANY_NUM  Second input

3.3.8.2 Outputs

Q : ANY_NUM  Result: IN1 * IN2

3.3.8.3 Remarks

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

3.3.8.4 ST Language

Q := IN1 * IN2;

3.3.8.5 FBD Language

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

3.3.8.6 FFLD Language

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

(* ENO is equal to EN *)

3.3.8.7 IL Language:

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

See also

+ - /
3.3.9 ODD

Function - Test if an integer is odd

3.3.9.1 Inputs
IN : DINT  Input value

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

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

3.3.9.4 ST Language
Q := ODD (IN);

3.3.9.5 FBD Language

3.3.9.6 FFLD Language
(* The function is executed only if EN is TRUE *)

3.3.9.7 IL Language:
Op1: LD IN
   ODD
   ODD
   ST Q  (* Q is TRUE if IN is odd *)

See also
MIN  MAX  LIMIT  MOD
3.3.10 SetWithin

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

### 3.3.10.1 Inputs

<table>
<thead>
<tr>
<th>IN</th>
<th>ANY</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN</td>
<td>ANY</td>
<td>Low limit of the interval</td>
</tr>
<tr>
<td>MAX</td>
<td>ANY</td>
<td>High limit of the interval</td>
</tr>
<tr>
<td>VAL</td>
<td>ANY</td>
<td>Value to apply when inside the interval</td>
</tr>
</tbody>
</table>

### 3.3.10.2 Outputs

| Q    | ANY | Result |

### 3.3.10.3 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>

### 3.3.10.4 Remarks

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

Operator - Performs a subtraction of inputs.

3.3.11.1 Inputs
IN1 : ANY_NUM / TIME  First input
IN2 : ANY_NUM / TIME  Second input

3.3.11.2 Outputs
Q : ANY_NUM / TIME  Result: IN1 - IN2

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

3.3.11.4 ST Language
Q := IN1 - IN2;

3.3.11.5 FBD Language

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

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

See also
+  *  /
3.4 Comparison Operations

Below are the standard operators and blocks that perform comparisons:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less or equal</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater or equal</td>
</tr>
<tr>
<td><code>==</code></td>
<td>is equal</td>
</tr>
<tr>
<td><code>&lt;&gt;</code></td>
<td>is not equal</td>
</tr>
<tr>
<td><strong>CMP</strong></td>
<td>detailed comparison</td>
</tr>
</tbody>
</table>
3.4.1 CMP

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

### 3.4.1.1 Inputs

IN1 : DINT  First value  
IN2 : DINT  Second value

### 3.4.1.2 Outputs

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

### 3.4.1.3 Remarks

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

### 3.4.1.4 ST Language

(* MyCmp is declared as an instance of CMP function block *)

MyCmp(IN1, IN2);

bLT := MyCmp.LT;
bEQ := MyCmp.EQ;
bGT := MyCmp.GT;

### 3.4.1.5 FBD Language

```
CMP
IN1   -LT
IN2   -EQ
     -GT
```

### 3.4.1.6 FFLD Language

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

```
EN   [   ]   [   ]
IN1   CMP   LT
     -EQ   -GT
```

### 3.4.1.7 IL Language:

(* MyCmp is declared as an instance of CMP function block *)

Op1: CAL MyCmp(IN1, IN2)

LD  MyCmp.LT
ST  bLT
LD  MyCmp.EQ
ST  bEQ
LD  MyCmp.GT
ST  bGT

See also

>  <  >=  <=  =  <>
### 3.4.2 \( >= \) **GE**

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

#### 3.4.2.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>First input</td>
</tr>
<tr>
<td>IN2</td>
<td>Second input</td>
</tr>
</tbody>
</table>

#### 3.4.2.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>TRUE if ( \text{IN1} \geq \text{IN2} )</td>
</tr>
</tbody>
</table>

#### 3.4.2.3 Remarks

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

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

#### 3.4.2.4 ST Language

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

#### 3.4.2.5 FBD Language

![FBD Diagram]

#### 3.4.2.6 FFLD Language

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

![FFLD Diagram]

#### 3.4.2.7 IL Language:

```
Op1: FFLD IN1
    GE IN2
    ST Q (* Q is true if IN1 \( \geq \) IN2 *)
```

**See also**

\( > \) \( < \) **\( <= \)** **\( = \)** **\( <> \)** **CMP**
### 3.4.3 > GT

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

#### 3.4.3.1 Inputs

<table>
<thead>
<tr>
<th>IN1</th>
<th>ANY</th>
<th>First input</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN2</td>
<td>ANY</td>
<td>Second input</td>
</tr>
</tbody>
</table>

#### 3.4.3.2 Outputs

| Q     | BOOL  | TRUE if IN1 > IN2 |

#### 3.4.3.3 Remarks

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

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

#### 3.4.3.4 ST Language

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

#### 3.4.3.5 FBD Language

\[ \text{IN1} \quad > \quad \text{IN2} \quad \text{Q} \]

#### 3.4.3.6 FFLD Language

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

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

#### 3.4.3.7 IL Language:

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

See also

\[ < \quad \geq \quad \leq \quad = \quad <> \quad \text{CMP} \]
3.4.4 = **EQ**

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

### 3.4.4.1 Inputs

IN1 : ANY  First input  
IN2 : ANY  Second input

### 3.4.4.2 Outputs

Q : BOOL  TRUE if \( \text{IN1} = \text{IN2} \)

### 3.4.4.3 Remarks

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

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

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

### 3.4.4.4 ST Language

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

### 3.4.4.5 FBD Language

```
IN1 = IN2  
\quad \rightarrow \quad Q
```

### 3.4.4.6 FFLD Language

("The comparison is executed only if \( \text{EN} \) is TRUE")

```
[  ]  
\quad \text{IN1} = \text{IN2}  
\quad \rightarrow \quad Q
```

### 3.4.4.7 IL Language:

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

See also

>  <  \geq  \leq  <\rightarrow  \ CMP
3.4.5 <> NE

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

3.4.5.1 Inputs
IN1 : ANY  First input
IN2 : ANY  Second input

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

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

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

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

3.4.5.4 ST Language
Q := IN1 <> IN2;

3.4.5.5 FBD Language

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

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

See also
> < >= <= = CMP
3.4.6 <= LE

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

3.4.6.1 Inputs
IN1 : ANY First input
IN2 : ANY Second input

3.4.6.2 Outputs
Q : BOOL TRUE if IN1 <= IN2

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

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

3.4.6.4 ST Language
Q := IN1 <= IN2;

3.4.6.5 FBD Language

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

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

See also
> < >= <= <> CMP
3.4.7 < LT

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

### 3.4.7.1 Inputs

- **IN1**: ANY  
  First input
- **IN2**: ANY  
  Second input

### 3.4.7.2 Outputs

- **Q**: BOOL  
  TRUE if IN1 < IN2

### 3.4.7.3 Remarks

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

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

### 3.4.7.4 ST Language

Q := IN1 < IN2;

### 3.4.7.5 FBD Language

![FBD Diagram]

### 3.4.7.6 FFLD Language

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

![FFLD Diagram]

### 3.4.7.7 IL Language:

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

See also

>  >=  <=  =  <>  CMP
3.5 Type conversion functions

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANY_TO_BOOL</td>
<td>converts to Boolean</td>
</tr>
<tr>
<td>ANY_TO_SINT/ANY_TO_USINT</td>
<td>converts to small (8 bit) integer</td>
</tr>
<tr>
<td>ANY_TO_INT/ANY_TO_UINT</td>
<td>converts to 16 bit integer</td>
</tr>
<tr>
<td>ANY_TO_DINT/ANY_TO_UDINT</td>
<td>converts to integer (32 bit - default)</td>
</tr>
<tr>
<td>ANY_TO_LINT/ANY_TO_ULINT</td>
<td>converts to long (64 bit) integer</td>
</tr>
<tr>
<td>ANY_TO_REAL</td>
<td>converts to real</td>
</tr>
<tr>
<td>ANY_TO_LREAL</td>
<td>converts to double precision real</td>
</tr>
<tr>
<td>ANY_TO_TIME</td>
<td>converts to time</td>
</tr>
<tr>
<td>ANY_TO_STRING</td>
<td>converts to character string</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIN_TO_BCD</td>
<td>converts a binary value to a BCD value</td>
</tr>
<tr>
<td>BCD_TO_BIN</td>
<td>converts a BCD value to a binary value</td>
</tr>
</tbody>
</table>

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

Operator - Converts the input into Boolean value.

3.5.1.1 Inputs
IN : ANY   Input value

3.5.1.2 Outputs
Q : BOOL   Value converted to Boolean

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

3.5.1.4 ST Language
Q := ANY_TO_BOOL (IN);

3.5.1.5 FBD Language

3.5.1.6 FFLD Language
(* The conversion is executed only if EN is TRUE *)
(* The output rung is the result of the conversion *)
(* The output rung is FALSE if the EN is FALSE *)

3.5.1.7 IL Language:
Op1: FFLD IN
      ANY_TO_BOOL
      ST Q

3.5.1.8 See also
ANY_TO_SINT   ANY_TO_INT   ANY_TO_DINT   ANY_TO_LINT   ANY_TO_REAL   ANY_TO_LREAL
ANY_TO_TIME    ANY_TO_STRING
3.5.2 ANY_TO_DINT / ANY_TO_UDINT

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

3.5.2.1 Inputs

IN : ANY  Input value

3.5.2.2 Outputs

Q : DINT  Value converted to integer

3.5.2.3 Remarks

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

3.5.2.4 ST Language

Q := ANY_TO_DINT (IN);

3.5.2.5 FBD Language

```
IN
```
```
ANY_TO_DINT
```
```
Q
```

3.5.2.6 FFLD Language

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

```
EN
```
```
[ ANY_TO_DINT ]
```
```
ENO
```
```
Q
```

3.5.2.7 IL Language:

```
Op1: FFLD IN
ANY_TO_DINT
ST Q
```

3.5.2.8 See also

ANY_TO_BOOL  ANY_TO_SINT  ANY_TO_INT  ANY_TO_LINT  ANY_TO_REAL  ANY_TO_LREAL
ANY_TO_TIME   ANY_TO_STRING
3.5.3 ANY_TO_INT / ANY_TO_UINT

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

3.5.3.1 Inputs
IN : ANY Input value

3.5.3.2 Outputs
Q : INT Value converted to 16 bit integer

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

3.5.3.4 ST Language
Q := ANY_TO_INT (IN);

3.5.3.5 FBD Language

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

3.5.3.7 IL Language:
Op1: FFLD IN
      ANY_TO_INT
      ST Q

3.5.3.8 See also
ANY_TO_BOOL  ANY_TO_SINT  ANY_TO_DINT  ANY_TO_LINT  ANY_TO_REAL  ANY_TO_LREAL
ANY_TO_TIME  ANY_TO_STRING
3.5.4 ANY_TO_LINT / ANY_TO_ULINT

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

3.5.4.1 Inputs

IN : ANY  Input value

3.5.4.2 Outputs

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

3.5.4.3 Remarks

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

3.5.4.4 ST Language

Q := ANY_TO_LINT (IN);

3.5.4.5 FBD Language

![FBD Diagram]

3.5.4.6 FFLD Language

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

3.5.4.7 IL Language:

Op1: FFLD IN
    ANY_TO_LINT
    ST Q

3.5.4.8 See also

ANY_TO_BOOL  ANY_TO_SINT  ANY_TO_INT  ANY_TO_DINT  ANY_TO_REAL  ANY_TO_LREAL
ANY_TO_TIME   ANY_TO_STRING
3.5.5 ANY_TO_LREAL

Operator - Converts the input into double precision real value.

3.5.5.1 Inputs

IN : ANY  Input value

3.5.5.2 Outputs

Q : LREAL  Value converted to double precision real

3.5.5.3 Remarks

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

3.5.5.4 ST Language

Q := ANY_TO_LREAL (IN);

3.5.5.5 FBD Language

\[
\begin{array}{c}
\text{IN} \\
\hline
\text{ANY_TO_LREAL} \\
\hline
\text{Q}
\end{array}
\]

3.5.5.6 FFLD Language

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

\[
\begin{array}{c}
\text{EN} \\
\hline
\text{IN} \\
\text{ANY_TO_LREAL} \\
\hline
\text{ENO} \\
\text{Q}
\end{array}
\]

3.5.5.7 IL Language:

Op1: FFLD IN
    ANY_TO_LREAL
    ST Q

3.5.5.8 See also

ANY_TO_BOOL  ANY_TO_SINT  ANY_TO_INT  ANY_TO_DINT  ANY_TO_LINT  ANY_TO_REAL
ANY_TO_TIME   ANY_TO_STRING
3.5.6 ANY_TO_REAL

Operator - Converts the input into real value.

3.5.6.1 Inputs

IN : ANY  Input value

3.5.6.2 Outputs

Q : REAL  Value converted to real

3.5.6.3 Remarks

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

3.5.6.4 ST Language

Q := ANY_TO_REAL (IN);

3.5.6.5 FBD Language

\[
\begin{array}{c}
\text{ANY_TO_REAL} \\
\text{IN} \\
\hline
\end{array}
\quad \begin{array}{c}
\text{Q}
\end{array}
\]

3.5.6.6 FFLD Language

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

\[
\begin{array}{c}
\text{EN} \\
\hline
\text{IN} \\
\text{ANY_TO_REAL} \\
\hline
\text{ENO} \\
\hline
\text{Q}
\end{array}
\]

3.5.6.7 IL Language:

Op1: FFLD IN
ANY_TO_REAL
ST Q

3.5.6.8 See also

ANY_TO_BOOL  ANY_TO_SINT  ANY_TO_INT  ANY_TO_DINT  ANY_TO_LINT  ANY_TO_LREAL
ANY_TO_TIME  ANY_TO_STRING
3.5.7 ANY_TO_TIME

Operator - Converts the input into time value.

3.5.7.1 Inputs
IN : ANY Input value

3.5.7.2 Outputs
Q : TIME Value converted to time

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

3.5.7.4 ST Language
Q := ANY_TO_TIME (IN);

3.5.7.5 FBD Language

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

3.5.7.7 IL Language:
Op1: FFLD IN
ANY_TO_TIME
ST Q

3.5.7.8 See also
ANY_TO_BOOL  ANY_TO_SINT  ANY_TO_INT  ANY_TO_DINT  ANY_TO_LINT  ANY_TO_REAL
ANY_TO_LREAL  ANY_TO_STRING
3.5.8 ANY_TO_SINT / ANY_TO_USINT

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

3.5.8.1 Inputs

IN : ANY  Input value

3.5.8.2 Outputs

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

3.5.8.3 Remarks

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

3.5.8.4 ST Language

Q := ANY_TO_SINT (IN);

3.5.8.5 FBD Language

```
IN  ANY_TO_SINT  Q
```

3.5.8.6 FFLD Language

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

```
EN [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
IN  ANY_TO_SINT  ENO

( )

Q
```

3.5.8.7 IL Language

Op1: FFLD IN
     ANY_TO_SINT
     ST Q

3.5.8.8 See also

ANY_TO_BOOL  ANY_TO_INT  ANY_TO_DINT  ANY_TO_LINT  ANY_TO_REAL  ANY_TO_LREAL
ANY_TO_TIME  ANY_TO_STRING
3.5.9 **ANY_TO_STRING**

*Operator* - Converts the input into string value.

### 3.5.9.1 Inputs

**IN** : ANY Input value

### 3.5.9.2 Outputs

**Q** : STRING Value converted to string

### 3.5.9.3 Remarks

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

### 3.5.9.4 ST Language

Q := ANY_TO_STRING(IN);

### 3.5.9.5 FBD Language

```
IN   ANY_TO_STRING   Q
```

### 3.5.9.6 FFLD Language

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

(* ENO keeps the same value as EN *)

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

### 3.5.9.7 IL Language:

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

### 3.5.9.8 See also

ANY_TO_BOOL, ANY_TO_SINT, ANY_TO_INT, ANY_TO_DINT, ANY_TO_LINT, ANY_TO_REAL, ANY_TO_LREAL, ANY_TO_TIME
3.5.10 NUM_TO_STRING

Function- Converts a number into string value.

3.5.10.1 Inputs

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

3.5.10.2 Outputs

Q : STRING  Value converted to string.

3.5.10.3 Remarks

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

If WIDTH is 0, the string is formatted with the necessary length.
If WIDTH is greater than 0, the string is completed with heading blank characters in order to match the value of WIDTH.
If WIDTH is greater than 0, the string is completed with trailing blank characters in order to match the absolute value of WIDTH.
If DIGITS is 0 then neither decimal part nor point are added.
If DIGITS is greater than 0, the corresponding number of decimal digits are added. '0' digits are added if necessary
If the value is too long for the specified width, then the string is filled with '*' characters.

3.5.10.4 Examples

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

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

### 3.5.11.1 Inputs

IN : DINT  Integer value in BCD

### 3.5.11.2 Outputs

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

### 3.5.11.3.1 Truth table (examples)

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

### 3.5.11.4 Remarks

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

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

### 3.5.11.5 ST Language

Q := BCD_TO_BIN (IN);

### 3.5.11.6 FBD Language

![FBD Diagram](image)

### 3.5.11.7 FFLD Language

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

(* ENO keeps the same value as EN *)

### 3.5.11.8 IL Language

Op1: LD IN
BCD_TO_BIN
ST Q

See also

**BIN_TO_BCD**
3.5.12 BIN_TO_BCD

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

3.5.12.1 Inputs

IN : DINT Integer value

3.5.12.2 Outputs

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

3.5.12.3 Truth table (examples)

<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>

3.5.12.4 Remarks

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

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

3.5.12.5 ST Language

Q := BIN_TO_BCD (IN);

3.5.12.6 FBD Language

3.5.12.7 FFLD Language

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

3.5.12.8 IL Language:

Op1: LD IN
     BIN_TO_BCD
     ST Q

See also

BCD_TO_BIN
3.6 Selectors

Below are the standard functions that perform data selection:

<table>
<thead>
<tr>
<th>Function</th>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEL</td>
<td>2 integer inputs</td>
</tr>
<tr>
<td>MUX4</td>
<td>4 integer inputs</td>
</tr>
<tr>
<td>MUX8</td>
<td>8 integer inputs</td>
</tr>
</tbody>
</table>
3.6.1 MUX4

Function - Select one of the inputs - 4 inputs.

3.6.1.1 Inputs

<table>
<thead>
<tr>
<th>SELECT</th>
<th>DINT</th>
<th>Selection command</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>ANY</td>
<td>First input</td>
</tr>
<tr>
<td>IN2</td>
<td>ANY</td>
<td>Second input</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN4</td>
<td>ANY</td>
<td>Last input</td>
</tr>
</tbody>
</table>

3.6.1.2 Outputs

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

3.6.1.3 Truth table

<table>
<thead>
<tr>
<th>SELECT</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IN1</td>
</tr>
<tr>
<td>1</td>
<td>IN2</td>
</tr>
<tr>
<td>2</td>
<td>IN3</td>
</tr>
<tr>
<td>3</td>
<td>IN4</td>
</tr>
<tr>
<td>other</td>
<td>0</td>
</tr>
</tbody>
</table>

3.6.1.4 Remarks

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

3.6.1.5 ST Language

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

3.6.1.6 FBD Language

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

3.6.1.8 IL Language

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

See also
    SEL MUX8
### 3.6.2 MUX8

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

#### 3.6.2.1 Inputs

<table>
<thead>
<tr>
<th>SELECT</th>
<th>DINT</th>
<th>Selection command</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>ANY</td>
<td>First input</td>
</tr>
<tr>
<td>IN2</td>
<td>ANY</td>
<td>Second input</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN8</td>
<td>ANY</td>
<td>Last input</td>
</tr>
</tbody>
</table>

#### 3.6.2.2 Outputs

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

#### 3.6.2.3 Truth table

<table>
<thead>
<tr>
<th>SELECT</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IN1</td>
</tr>
<tr>
<td>1</td>
<td>IN2</td>
</tr>
<tr>
<td>2</td>
<td>IN3</td>
</tr>
<tr>
<td>3</td>
<td>IN4</td>
</tr>
<tr>
<td>4</td>
<td>IN5</td>
</tr>
<tr>
<td>5</td>
<td>IN6</td>
</tr>
<tr>
<td>6</td>
<td>IN7</td>
</tr>
<tr>
<td>7</td>
<td>IN8</td>
</tr>
<tr>
<td>other</td>
<td>0</td>
</tr>
</tbody>
</table>

#### 3.6.2.4 Remarks

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

#### 3.6.2.5 ST Language

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

#### 3.6.2.6 FBD Language

![FBD Diagram]

#### 3.6.2.7 FFLD Language
(the selection is performed only if EN is TRUE *)
(*) ENO has the same value as EN *)

3.6.2.8 IL Language

Not available

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

See also
   SEL MUX4
3.6.3 SEL

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

### 3.6.3.1 Inputs

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

### 3.6.3.2 Outputs

- **Q** : ANY  IN1 if SELECT is FALSE; IN2 if SELECT is TRUE

### 3.6.3.3 Truth table

<table>
<thead>
<tr>
<th>SELECT</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IN1</td>
</tr>
<tr>
<td>1</td>
<td>IN2</td>
</tr>
</tbody>
</table>

### 3.6.3.4 Remarks

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

### 3.6.3.5 ST Language

\[ Q := \text{SEL(} \text{SELECT, IN1, IN2)}; \]

### 3.6.3.6 FBD Language

![FBD Diagram](image)

### 3.6.3.7 FFLD Language

(* the input rung is the selector *)

(* ENO has the same value as SELECT *)

![FFLD Diagram](image)

### 3.6.3.8 IL Language

\[ \text{Op1: LD SELECT} \]
\[ \text{SEL IN1, IN2} \]
\[ \text{ST Q} \]

**See also**

- MUX4
- MUX8
3.7 Registers

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

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

Below are advanced functions for register manipulation:

- **MBShift**  multibyte shift / rotate

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

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

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

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

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

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

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

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

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

3.7.1.1 Inputs

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

3.7.1.2 Outputs

Q : ANY  AND mask between IN and MSK inputs

3.7.1.3 Remarks

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

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

3.7.1.4 ST Language

\[ Q := \text{AND}\_\text{MASK}\ (\text{IN}, \text{MSK}); \]

3.7.1.5 FBD Language

```
<table>
<thead>
<tr>
<th>IN</th>
<th>AND_MASK</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

3.7.1.6 FFLD Language

```
EN \[ \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \Quad
3.7.2 HIBYTE

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

3.7.2.1 Inputs

IN : UINT  16 bit register

3.7.2.2 Outputs

Q : USINT  Most significant byte

3.7.2.3 Remarks

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

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

3.7.2.4 ST Language

Q := HIBYTE (IN);

3.7.2.5 FBD Language

![FBD Diagram]

3.7.2.6 FFLD Language

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

3.7.2.7 IL Language:

Op1: LD IN
HIBYTE
ST Q

See also

LOBYTE  LOWORD  HIWORD  MAKEWORD  MAKEDWORD
3.7.3 LOBYTE

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

3.7.3.1 Inputs

IN : UINT 16 bit register

3.7.3.2 Outputs

Q : USINT Lowest significant byte

3.7.3.3 Remarks

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

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

3.7.3.4 ST Language

Q := LOBYTE (IN);

3.7.3.5 FBD Language

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

3.7.3.6 FFLD Language

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

("ENO keeps the same value as EN")

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

3.7.3.7 IL Language:

Op1: LD IN
LOBYTE
ST Q

See also

HIBYTE LOWORD HIWORD MAKEWORD MAKEDWORD
3.7.4 HIWORD

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

**3.7.4.1 Inputs**

IN : UDINT 32 bit register

**3.7.4.2 Outputs**

Q : UINT Most significant word

**3.7.4.3 Remarks**

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

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

**3.7.4.4 ST Language**

Q := HIWORD (IN);

**3.7.4.5 FBD Language**

```
IN  HiWord  Q
```

**3.7.4.6 FFLD Language**

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

(* ENO keeps the same value as EN *)

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

**3.7.4.7 IL Language:**

Op1: LD IN
HIWORD
ST Q

See also

LOBYTE, HIBYTE, LOWORD, MAKEWORD, MAKEDWORD
3.7.5 LOWORD

Function - Get the less significant word of a double word

3.7.5.1 Inputs
IN : UDINT 32 bit register

3.7.5.2 Outputs
Q : UINT Lowest significant word

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

3.7.5.4 ST Language
Q := LOWORD (IN);

3.7.5.5 FBD Language

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

3.7.5.7 IL Language:
Op1: LD IN
LOWORD
ST Q

See also
LOBYTE  HIBYTE  HIWORD  MAKEWORD  MAKEDWORD
3.7.6 MAKEDWORD

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

### 3.7.6.1 Inputs

- **HI**: USINT  Highest significant word
- **LO**: USINT  Lowest significant word

### 3.7.6.2 Outputs

- **Q**: UINT  32 bit register

### 3.7.6.3 Remarks

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

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

### 3.7.6.4 ST Language

\[ Q := \text{MAKEDWORD}(\text{HI}, \text{LO}); \]

### 3.7.6.5 FBD Language

![FBD Diagram]

### 3.7.6.6 FFLD Language

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

(* ENO keeps the same value as EN *)

![FFLD Diagram]

### 3.7.6.7 IL Language:

Op1: LD  \text{HI}  
MAKEDWORD  \text{LO}  
ST  \text{Q}  

See also

- LOBYTE  HIBYTE  LOWORD  HIWORD  MAKEWORD
3.7.7 **MAKEWORD**

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

**3.7.7.1 Inputs**

<table>
<thead>
<tr>
<th>HI</th>
<th>USINT</th>
<th>Highest significant byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>LO</td>
<td>USINT</td>
<td>Lowest significant byte</td>
</tr>
</tbody>
</table>

**3.7.7.2 Outputs**

| Q    | UINT   | 16 bit register          |

**3.7.7.3 Remarks**

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

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

**3.7.7.4 ST Language**

\[ Q := \text{MAKEWORD}(\text{HI}, \text{LO}); \]

**3.7.7.5 FBD Language**

![FBD Diagram]

**3.7.7.6 FFLD Language**

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

(* ENO keeps the same value as EN *)

![FFLD Diagram]

**3.7.7.7 IL Language:**

Op1: LD HI
MAKEDWORD LO
ST Q

See also

| LOBYTE | HIBYTE | LOWORD | HIWORD | MAKEDWORD |
3.7.8 MB SHIFT

Function - Multibyte shift / rotate

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

3.7.8.2 Outputs
Q : BOOL  TRUE if successful

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

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

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

3.7.8.5 FBD Language

3.7.8.6 FFLD Language
(* the function is called only if EN is TRUE *)

3.7.8.7 IL Language:
Not available
3.7.9 NOT\_MASK

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

### 3.7.9.1 Inputs

\[
\text{IN} : \text{ANY Integer input}
\]

### 3.7.9.2 Outputs

\[
\text{Q} : \text{ANY Bit to bit negation of the input}
\]

### 3.7.9.3 Remarks

Arguments can be signed or unsigned integers from 8 to 32 bits. In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the same value as the input rung. In IL language, the parameter (IN) must be loaded in the current result before calling the function.

#### 3.7.9.4 ST Language

\[
\text{Q} := \text{NOT\_MASK (IN)};
\]

#### 3.7.9.5 FBD Language

\[
\begin{array}{c}
\text{IN} \\
\hline \\
\text{NOT\_MASK} \\
\hline \\
\text{Q}
\end{array}
\]

#### 3.7.9.6 FFLD Language

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

\[
\begin{array}{c}
\text{EN} \\
\hline \\
\text{IN} \\
\hline \\
\text{NOT\_MASK} \\
\hline \\
\text{ENO} \\
\hline \\
\text{Q}
\end{array}
\]

#### 3.7.9.7 IL Language:

\[
\text{Op1: LD IN NOT\_MASK ST Q}
\]

See also

- AND\_MASK
- OR\_MASK
- XOR\_MASK
3.7.10 OR\_MASK

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

### 3.7.10.1 Inputs

- **IN**: ANY First input
- **MSK**: ANY Second input (OR mask)

### 3.7.10.2 Outputs

- **Q**: ANY OR mask between IN and MSK inputs

### 3.7.10.3 Remarks

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

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

### 3.7.10.4 ST Language

\[
Q := \text{OR\_MASK} (\text{IN}, \text{MSK});
\]

### 3.7.10.5 FBD Language

![FBD Diagram]

### 3.7.10.6 FFLD Language

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

### 3.7.10.7 IL Language:

```
Op1: LD IN
    OR\_MASK MSK
    ST Q
```

See also

- AND\_MASK
- XOR\_MASK
- NOT\_MASK
3.7.11 PACK8

*Function* - Builds a byte with bits

### 3.7.11.1 Inputs

- **IN0**: BOOL  Less significant bit
- **...**
- **IN7**: BOOL  Most significant bit

### 3.7.11.2 Outputs

- **Q**: USINT  Byte built with input bits

### 3.7.11.3 Remarks

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

### 3.7.11.4 ST Language

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

### 3.7.11.5 FBD Language

![FBD Diagram](https://example.com/fbd_diagram.png)
3.7.11.6 FFLD Language

(* ENO keeps the same value as EN *)

3.7.11.7 IL Language

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

See also
UNPACK8
3.7.12 ROL

Function - Rotate bits of a register to the left.

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

3.7.12.2 Outputs
Q : ANY Rotated register

3.7.12.3 Diagram

3.7.12.4 Remarks
Arguments can be signed or unsigned integers from 8 to 32 bits.
In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the state of the input rung. In IL language, the first input must be loaded before the function call. The second input is the operand of the function.

3.7.12.5 ST Language
Q := ROL (IN, NBR);

3.7.12.6 FBD Language

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

3.7.12.8 IL Language:
Op1: LD IN
ROL NBR
ST Q

See also
SHL SHR ROR
3.7.13 ROR

Function - Rotate bits of a register to the right.

3.7.13.1 Inputs

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

3.7.13.2 Outputs

Q : ANY Rotated register

3.7.13.3 Diagram

3.7.13.4 Remarks

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

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

3.7.13.5 ST Language

Q := ROR (IN, NBR);

3.7.13.6 FBD Language

3.7.13.7 FFLD Language

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

3.7.13.8 IL Language:

Op1: LD IN
    ROR NBR
    ST Q

See also

SHL, SHR, ROL
3.7.14 RORb / ROR_SINT / ROR_USINT / ROR_BYTE

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

3.7.14.1 Inputs

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

3.7.14.2 Outputs

Q : SINT Rotated register

3.7.14.3 Diagram

3.7.14.4 Remarks

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

3.7.14.5 ST Language

Q := RORb (IN, NBR);

3.7.14.6 FBD Language

3.7.14.7 FFLD Language

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

3.7.14.8 IL Language:

Op1: FFLD IN
    RORb NBR
    ST Q

3.7.14.9 See also

SHL, SHR, ROL, ROR, SHLb, SHRb, ROLb, SHLw, SHRw, ROLw, RORw
3.7.15 RORw / ROR_INT / ROR_UINT / ROR_WORD

Function - Rotate bits of a register to the right.

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

3.7.15.2 Outputs
Q : INT Rotated register

3.7.15.3 Diagram

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

3.7.15.5 ST Language
Q := RORw (IN, NBR);

3.7.15.6 FBD Language

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

3.7.15.8 IL Language:
Op1: FFLD IN
    RORw NBR
    ST Q

3.7.15.9 See also
SHL  SHR  ROL  ROR  SHLb SHRb ROLb RORb  SHLw SHRw ROLw
3.7.16 SETBIT

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

3.7.16.1 Inputs

IN : ANY 8 to 32 bit integer register
BIT : DINT Bit number (0 = less significant bit)
VAL : BOOL Bit value to apply

3.7.16.2 Outputs

Q : ANY Modified register

3.7.16.3 Remarks

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

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

3.7.16.4 ST Language

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

3.7.16.5 FBD Language

```
SetBit
IN
BIT
VAL
```

3.7.16.6 FFLD Language

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

```
EN
[ ]
IN
BIT
VAL
SetBit
ENO
( )
```

3.7.16.7 IL Language

Not available

See also

TESTBIT
3.7.17 SHL

Function - Shift bits of a register to the left.

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

3.7.17.2 Outputs
Q : ANY Shifted register

3.7.17.3 Diagram

```
  NBS
  |
  |
  |
  0 ← 0
```

3.7.17.4 Remarks
Arguments can be signed or unsigned integers from 8 to 32 bits.
In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the state of the input rung. In IL language, the first input must be loaded before the function call. The second input is the operand of the function.

3.7.17.5 ST Language
Q := SHL (IN, NBS);

3.7.17.6 FBD Language

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

3.7.17.8 IL Language:
Op1: LD IN
     SHL NBS
     ST Q

See also
SHR  ROL  ROR
3.7.18 SHR

Function - Shift bits of a register to the right.

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

3.7.18.2 Outputs
Q : ANY Shifted register

3.7.18.3 Diagram

3.7.18.4 Remarks
Arguments can be signed or unsigned integers from 8 to 32 bits.
In FFLD language, the input rung (EN) enables the operation, and the output rung keeps the state of the input rung. In IL language, the first input must be loaded before the function call. The second input is the operand of the function.

3.7.18.5 ST Language
Q := SHR (IN, NBS);

3.7.18.6 FBD Language

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

3.7.18.8 IL Language:
Op1: LD IN
     SHR NBS
     ST Q

See also
SHL, ROL, ROR
3.7.19 TESTBIT

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

3.7.19.1 Inputs

IN : ANY 8 to 32 bit integer register
BIT : DINT Bit number (0 = less significant bit)

3.7.19.2 Outputs

Q : BOOL Bit value

3.7.19.3 Remarks

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

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

3.7.19.4 ST Language

Q := TESTBIT (IN, BIT);

3.7.19.5 FBD Language

![FBD Diagram]

3.7.19.6 FFLD Language

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

![FFLD Diagram]

3.7.19.7 IL Language

*Not available*

See also

SETBIT
### 3.7.20 UNPACK8

*Function block - Extract bits of a byte*

#### 3.7.20.1 Inputs

**IN**: USINT 8 bit register

#### 3.7.20.2 Outputs

**Q0**: BOOL Less significant bit

...  

**Q7**: BOOL Most significant bit

#### 3.7.20.3 Remarks

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

#### 3.7.20.4 ST Language

(* MyUnpack is a declared instance of the UNPACK8 function block * )

MyUnpack (IN);

Q0 := MyUnpack.Q0;
Q1 := MyUnpack.Q1;
Q2 := MyUnpack.Q2;
Q3 := MyUnpack.Q3;
Q4 := MyUnpack.Q4;
Q5 := MyUnpack.Q5;
Q6 := MyUnpack.Q6;
Q7 := MyUnpack.Q7;

#### 3.7.20.5 FBD Language

![FBD Diagram](image)

#### 3.7.20.6 FFLD Language
(* The operation is performed if EN = TRUE *)

3.7.20.7 IL Language:

(* MyUnpack is a declared instance of the UNPACK8 function block *)

Op1: CAL MyUnpack (IN)
    FFLD MyUnpack.Q0
    ST Q0
    (* ... *)
    FFLD MyUnpack.Q7
    ST Q7

See also

PACK8
### 3.7.21 XOR\_MASK

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

#### 3.7.21.1 Inputs

- **IN** : ANY  
  First input

- **MSK** : ANY  
  Second input (XOR mask)

#### 3.7.21.2 Outputs

- **Q** : ANY  
  Exclusive OR mask between IN and MSK inputs

#### 3.7.21.3 Remarks

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

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

#### 3.7.21.4 ST Language

```
Q := XOR\_MASK (IN, MSK);
```

#### 3.7.21.5 FBD Language

```
IN  MSK
   XOR\_MASK       Q
```

#### 3.7.21.6 FFLD Language

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

(* ENO is equal to EN *)

```
IN  MSK
   XOR\_MASK ( ) Q

ENO
```

#### 3.7.21.7 IL Language:

```
Op1: LD  IN
     XOR\_MASK MSK
     ST   Q
```

**See also**

AND\_MASK  OR\_MASK  NOT\_MASK
3.8 Counters

Below are the standard blocks for managing counters:

<table>
<thead>
<tr>
<th>CTU</th>
<th>Up counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTD</td>
<td>Down Counter</td>
</tr>
<tr>
<td>CTUD</td>
<td>Up / Down Counter</td>
</tr>
</tbody>
</table>
3.8.1 CTD / CTDr

Function Block - Down counter.

3.8.1.1 Inputs

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

3.8.1.2 Outputs

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

3.8.1.3 Remarks

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

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

3.8.1.4 ST Language

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

3.8.1.5 FBD Language


3.8.1.6 FFLD Language


3.8.1.7 IL Language:

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

See also
CTU  CTUD
3.8.2 CTU / CTUr

Function Block - Up counter.

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

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

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

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

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

3.8.2.5 FBD Language

3.8.2.6 FFLD Language

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

See also
CTD CTUD
3.8.3 CTUD / CTUDr

Function Block - Up/down counter.

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

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

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

3.8.3.4 ST Language

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

3.8.3.5 FBD Language

```
CTUD
CU  QU
CD  QD
RESET CV
LOAD
PV
```
3.8.3.6 FFLD Language

```
CU [ CTUD QU ]
 CD
 RESET
 LOAD
 PV
```

3.8.3.7 IL Language:

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

See also

CTU CTD
3.9 Timers

Below are the standard functions for managing timers:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TON</td>
<td>On timer</td>
</tr>
<tr>
<td>TOF</td>
<td>Off timer</td>
</tr>
<tr>
<td>TP</td>
<td>Pulse timer</td>
</tr>
<tr>
<td>BLINK</td>
<td>Blinker</td>
</tr>
<tr>
<td>BLINKA</td>
<td>Asymmetric blinker</td>
</tr>
<tr>
<td>PLS</td>
<td>Pulse signal generator</td>
</tr>
<tr>
<td>TMU</td>
<td>Up-counting stop watch</td>
</tr>
<tr>
<td>TMUsec</td>
<td>Up-counting stop watch (seconds)</td>
</tr>
<tr>
<td>TMD</td>
<td>Down-counting stop watch</td>
</tr>
</tbody>
</table>
3.9.1 BLINK

*Function Block* - Blinker.

3.9.1.1 Inputs
- **RUN** : BOOL   Enabling command
- **CYCLE** : TIME Blinking period

3.9.1.2 Outputs
- **Q** : BOOL   Output blinking signal

3.9.1.3 Time diagram

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

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

3.9.1.6 FBD Language

3.9.1.7 FFLD Language

```
RUN [ ] CYCLE [ ]

Q

BLINK
```

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

See also
- **TON**
- **TOF**
- **TP**
3.9.2 BLINKA

*Function Block* - Asymmetric blinker.

### 3.9.2.1 Inputs

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

### 3.9.2.2 Outputs

- **Q**: BOOL  
  Output blinking signal

### 3.9.2.3 Time diagram

![Time diagram for BLINKA function block]

### 3.9.2.4 Remarks

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

### 3.9.2.5 ST Language

(* MyBlinker is a declared instance of BLINKA function block *)

```plaintext
MyBlinker (RUN, TM0, TM1);
Q := MyBlinker.Q;
```

### 3.9.2.6 FBD Language

```
RUN
TM0
TM1
```

### 3.9.2.7 FFLD Language

```
RUN
TM0
TM1
```

![FFLD diagram for BLINKA function block]
3.9.2.8  IL Language:

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

See also

TON  TOF  TP
3.9.3 PLS

*Function Block* - Pulse signal generator

### 3.9.3.1 Inputs

- **RUN**: BOOL  Enabling command
- **CYCLE**: TIME  Signal period

### 3.9.3.2 Outputs

- **Q**: BOOL  Output pulse signal

### 3.9.3.3 Time diagram

![Time diagram](image)

### 3.9.3.4 Remarks

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

### 3.9.3.5 ST Language

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

### 3.9.3.6 FBD Language

![FBD diagram](image)

### 3.9.3.7 FFLD Language

```
RUN [CYCLE] [PLS] Q
```

156  Kollmorgen | kdn.kollmorgen.com | July 2019
3.9.3.8 IL Language

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

See also
TON TOF TP
3.9.4 Sig_Gen

*Function Block* - Generator of pseudo-analogical Signal

3.9.4.1 Inputs

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

3.9.4.2 Outputs

This FB generates signals of the four following types:

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

3.9.4.3 FFLD Language

```
siggen
  sig_gen
  RUN
  PERIOD
  MAXIMUM
  PULSE
  UP
  END
  SINE
```
3.9.5 TMD

Function Block - Down-counting stop watch.

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

3.9.5.2 Outputs
Q : BOOL  Timer elapsed output signal
ET : TIME  Elapsed time

3.9.5.3 Time diagram

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

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

3.9.5.6 FBD Language

3.9.5.7 FFLD Language
3.9.5.8 IL Language

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

See also
TMU
3.9.6 TMU / TMUsec

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

### 3.9.6.1 Inputs

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

### 3.9.6.2 Outputs

<table>
<thead>
<tr>
<th>Q</th>
<th>BOOL</th>
<th>Timer elapsed output signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>ET</td>
<td>TIME</td>
<td>Elapsed time</td>
</tr>
<tr>
<td>ETsec</td>
<td>UDINT</td>
<td>Elapsed time. (TMU - seconds)</td>
</tr>
</tbody>
</table>

### 3.9.6.3 Time diagram

![Time diagram](image)

### 3.9.6.4 Remarks

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

### 3.9.6.5 ST Language

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

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

### 3.9.6.6 FBD Language
3.9.6.7 FFLD Language

3.9.6.8 IL Language:
(* MyTimer is a declared instance of TMU function block *)

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

See also

TMD
3.9.7 TOF / TOFR

Function Block - Off timer.

3.9.7.1 Inputs
IN : BOOL   Timer command
PT : TIME   Programmed time
RST : BOOL  Reset (TOFR only)

3.9.7.2 Outputs
Q : BOOL    Timer elapsed output signal
ET : TIME   Elapsed time

3.9.7.3 Time diagram

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

TOFR is same as TOF but has an extra input for resetting the timer
In FFLD language, the input rung is the IN command. The output rung is Q the output signal.

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

3.9.7.6 FBD Language

3.9.7.7 FFLD Language
3.9.7.8 IL Language:

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

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

See also

TON TP BLINK
3.9.8 TON

*PLCopen*

Function Block - On timer.

3.9.8.1 Inputs

IN : BOOL    Timer command
PT : TIME    Programmed time

3.9.8.2 Outputs

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

3.9.8.3 Time diagram

3.9.8.4 Remarks

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

3.9.8.5 ST Language

(* Inst_TON is a declared instance of TON function block *)

Inst_TON( FALSE, T#2s );
Q := Inst_TON.Q;
ET := Inst_TON.ET;

3.9.8.6 FBD Language

```
TON
IN
Q
PT
ET
```
3.9.8.7 FFLD Language

```
IN  [ ]  TON  ( )  Q
PT  [ ]  ET
```

3.9.8.8 IL Language:

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

See also

TOF  TP  BLINK
3.9.9 TP / TPR

*Function Block* - Pulse timer.

### 3.9.9.1 Inputs

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

### 3.9.9.2 Outputs

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

### 3.9.9.3 Time diagram

![Time diagram](image)

### 3.9.9.4 Remarks

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

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

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

### 3.9.9.5 ST Language

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

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

### 3.9.9.6 FBD Language

![FBD diagram](image)

### 3.9.9.7 FFLD Language

```
IN  [TP]
PT  Q
    [ET]
```

Kollmorgen | kdn.kollmorgen.com | July 2019  167
3.9.9.8 IL Language:

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

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

See also

TON TOF BLINK
### 3.10 Mathematic operations

Below are the standard functions that perform mathematic calculation:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ABS</strong></td>
<td>absolute value</td>
</tr>
<tr>
<td><strong>TRUNC</strong></td>
<td>integer part</td>
</tr>
<tr>
<td><strong>LOG, &quot;LN / LNL&quot; (p. 174)</strong></td>
<td>logarithm, natural logarithm</td>
</tr>
<tr>
<td><strong>POW, EXPT, &quot;EXP / EXPL&quot; (p. 172)</strong></td>
<td>power</td>
</tr>
<tr>
<td><strong>SQRT, &quot;ROOT&quot; (p. 176)</strong></td>
<td>square root, root extraction</td>
</tr>
<tr>
<td><strong>SCALELIN</strong></td>
<td>scaling - linear conversion</td>
</tr>
</tbody>
</table>
3.10.1 ABS / ABSL

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

3.10.1.1 Inputs

\[ \text{IN} : \text{REAL/LREAL} \quad \text{ANY value} \]

3.10.1.2 Outputs

\[ \text{Q} : \text{REAL/LREAL} \quad \text{Result: absolute value of IN} \]

3.10.1.3 Remarks

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

3.10.1.4 ST Language

\[ \text{Q} := \text{ABS} (\text{IN}); \]

3.10.1.5 FBD Language

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

3.10.1.6 FFLD Language

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

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

3.10.1.7 IL Language

\[ \text{Op1: LD IN} \\text{ABS} \\text{ST Q} \quad \text{(* Q is: ABS (IN) *)} \]

See also

TRUNC  LOG  POW  SQRT
3.10.2 EXPT

*Function* - Calculates a power.

### 3.10.2.1 Inputs

- **IN**: REAL  Real value
- **EXP**: DINT  Exponent

### 3.10.2.2 Outputs

- **Q**: REAL  Result: IN at the 'EXP' power

### 3.10.2.3 Remarks

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

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

### 3.10.2.4 ST Language

\[ Q := \text{EXPT}(\text{IN}, \text{EXP}); \]

### 3.10.2.5 FBD Language

![FBD Diagram]

### 3.10.2.6 FFLD Language

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

### 3.10.2.7 IL Language:

Op1: LD  \text{IN}
       \text{EXPT}  \text{EXP}
       \text{ST}  Q  (* Q is: (IN ** EXP) *)

**See also**

- ABS
- TRUNC
- LOG
- SQRT
3.10.3 EXP / EXPL

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

3.10.3.1 Inputs

\[ \text{IN} : \text{REAL/LREAL} \quad \text{Real value} \]

3.10.3.2 Outputs

\[ \text{Q} : \text{REAL/LREAL} \quad \text{Result: natural exponential of IN.} \]

3.10.3.3 Remarks

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

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

3.10.3.4 ST Language

\[
Q := \text{EXP (IN)};
\]

3.10.3.5 FBD Language

```
EXP
IN  -->  Q
```

3.10.3.6 FFLD Language

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

(* ENO keeps the same value as EN *)

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

3.10.3.7 IL Language:

```
Op1: LD  IN
     EXP
     ST  Q  (* Q is: EXP (IN) *)
```
3.10.4 LOG / LOGL

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

**3.10.4.1 Inputs**

IN : REAL/LREAL Real value

**3.10.4.2 Outputs**

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

**3.10.4.3 Remarks**

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

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

**3.10.4.4 ST Language**

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

**3.10.4.5 FBD Language**

```
IN   LOG   Q
```

**3.10.4.6 FFLD Language**

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

(* ENO keeps the same value as EN *)

```
EN   LOG   ENO
    IN   Q
```

**3.10.4.7 IL Language:**

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

See also

ABS  TRUNC  POW  SQRT
3.10.5 LN / LNL

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

3.10.5.1 Inputs

\( \text{IN} : \text{REAL}/\text{LREAL} \) Real value

3.10.5.2 Outputs

\( \text{Q} : \text{REAL}/\text{LREAL} \) Result: natural logarithm of \( \text{IN} \)

3.10.5.3 Remarks

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

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

3.10.5.4 ST Language

\[
\text{Q} := \text{LN} \ (\text{IN});
\]

3.10.5.5 FBD Language

[Diagram: FBD Language]

3.10.5.6 FFLD Language

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

(* ENO keeps the same value as EN *)

[Diagram: FFLD Language]

3.10.5.7 IL Language:

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

*Function* - Calculates a power.

**3.10.6.1 Inputs**

IN : REAL/LREAL  Real value
EXP : REAL/LREAL  Exponent

**3.10.6.2 Outputs**

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

**3.10.6.3 Remarks**

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

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

**3.10.6.4 ST Language**

\[ Q := \text{POW}(\text{IN}, \text{EXP}); \]
\[ Q := \text{IN}^{\text{EXP}}; \]

**3.10.6.5 FBD Language**

![FBD Diagram](image)

**3.10.6.6 FFLD Language**

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

\[
\begin{array}{c}
\text{POW} \\
\text{IN} \\
\text{EXP} \\
\end{array}
\begin{array}{c}
\text{ENO} \\
\text{Q} \\
\end{array}
\]

**3.10.6.7 IL Language:**

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

See also

ABS  TRUNC  LOG  SQRT
3.10.7 ROOT

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

### 3.10.7.1 Inputs

- **IN**: REAL  
  Real value
- **N**: DINT  
  Root level

### 3.10.7.2 Outputs

- **Q**: REAL  
  Result: Nth root of IN

### 3.10.7.3 Remarks

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

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

### 3.10.7.4 ST Language

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

### 3.10.7.5 FBD Language

![FBD Diagram]

### 3.10.7.6 FFLD Language

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

![FFLD Diagram]

### 3.10.7.7 IL Language:

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

*Function* - Scaling - linear conversion.

### 3.10.8.1 Inputs

<table>
<thead>
<tr>
<th>IN</th>
<th>REAL</th>
<th>Real value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMIN</td>
<td>REAL</td>
<td>Minimum input value</td>
</tr>
<tr>
<td>IMAX</td>
<td>REAL</td>
<td>Maximum 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>Maximum output value</td>
</tr>
</tbody>
</table>

### 3.10.8.2 Outputs

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

### 3.10.8.3 Truth table

<table>
<thead>
<tr>
<th>Inputs</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMIN &gt;= IMAX</td>
<td>= IN</td>
</tr>
<tr>
<td>IN &lt; IMIN</td>
<td>= IMIN</td>
</tr>
<tr>
<td>IN &gt; IMAX</td>
<td>= IMAX</td>
</tr>
<tr>
<td>other</td>
<td>= OMIN + IN * (OMAX - OMIN) / (IMAX - IMIN)</td>
</tr>
</tbody>
</table>

### 3.10.8.4 Remarks

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

### 3.10.8.5 ST Language

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

### 3.10.8.6 FBD Language

```
  ScaleLin
  \arrow{IN}{IN}
  \arrow{IMIN}{IMIN}
  \arrow{IMAX}{IMAX}
  \arrow{OMIN}{OMIN}
  \arrow{OMAX}{OMAX}
  \arrow{OUT}{OUT}
```
### 3.10.8.7 FFLD Language

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

```
EN   ScaleLin
IN   ENO
IMIN
IMAX
OMIN
OMAX
```

### 3.10.8.8 IL Language

Op1: LD IN

```
ScaleLin IMAX, IMIN, OMAX, OMIN
ST OUT
```
3.10.9 SQRT / SQRTL

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

### 3.10.9.1 Inputs
IN : REAL/LREAL  Real value

### 3.10.9.2 Outputs
Q : REAL/LREAL  Result: square root of IN

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

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

### 3.10.9.4 ST Language
Q := SQRT (IN);

### 3.10.9.5 FBD Language

```
IN  | SQRT | Q
```

### 3.10.9.6 FFLD Language

```
IN  [ SQRT ] ENO
    | IN   | Q
```

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

(* ENO keeps the same value as EN *)

### 3.10.9.7 IL Language:

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

See also

ABS  TRUNC  LOG  POW
3.10.10 TRUNC / TRUNCL

Function - Truncates the decimal part of the input.

3.10.10.1 Inputs

IN : REAL/LREAL  Real value

3.10.10.2 Outputs

Q : REAL/LREAL  Result: integer part of IN

3.10.10.3 Remarks

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

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

3.10.10.4 ST Language

Q := TRUNC (IN);

3.10.10.5 FBD Language

```
IN  TRUNC  Q
```

3.10.10.6 FFLD Language

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

```
EN \[\] IN  TRUNC  \{\}  ENO \[\] Q
```

3.10.10.7 IL Language:

Op1: LD IN
TRUNC
ST Q  (* Q is the integer part of IN *)

See also

ABS  LOG  POW  SQRT
3.11 Trigonometric functions

Below are the standard functions for trigonometric calculation:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIN</td>
<td>sine</td>
</tr>
<tr>
<td>COS</td>
<td>cosine</td>
</tr>
<tr>
<td>TAN</td>
<td>tangent</td>
</tr>
<tr>
<td>ASIN</td>
<td>arc-sine</td>
</tr>
<tr>
<td>ACOS</td>
<td>arc-cosine</td>
</tr>
<tr>
<td>ATAN</td>
<td>arc-tangent</td>
</tr>
<tr>
<td>ATAN2</td>
<td>arc-tangent of Y / X</td>
</tr>
</tbody>
</table>

See Also:

Use Degrees
3.11.1 ACOS / ACOSL

*Function* - Calculate an arc-cosine.

### 3.11.1.1 Inputs

IN : REAL/LREAL  Real value

### 3.11.1.2 Outputs

Q : REAL/LREAL  Result: arc-cosine of IN

### 3.11.1.3 Remarks

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

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

### 3.11.1.4 ST Language

Q := ACOS (IN);

### 3.11.1.5 FBD Language

\[
\begin{array}{c}
\text{IN} \\
\hline
\text{ACOS} \\
\hline
\text{Q}
\end{array}
\]

### 3.11.1.6 FFLD Language

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

(* ENO keeps the same value as EN *)

\[
\begin{array}{c}
\text{EN} \\
\hline
\text{IN} \\
\hline
\text{ACOS} \\
\hline
\text{ENO} \\
\hline
2
\end{array}
\]

### 3.11.1.7 IL Language:

Op1: LD IN

ACOS

ST Q (* Q is: ACOS (IN) *)

See also

SIN  COS  TAN  ASIN  ATAN  ATAN2
3.11.2 ASIN / ASINL

*Function* - Calculate an arc-sine.

3.11.2.1 Inputs
IN : REAL/LREAL  Real value

3.11.2.2 Outputs
Q : REAL/LREAL  Result: arc-sine of IN

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

3.11.2.4 ST Language
Q := ASIN (IN);

3.11.2.5 FBD Language

```
IN  ASIN  Q
```

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

```
EN  ASIN  ENO
    IN  Q
```

3.11.2.7 IL Language:
Op1: LD  IN
     ASIN
     ST  Q  (* Q is: ASIN (IN) *)

See also
SIN  COS  TAN  ACOS  ATAN  ATAN2
3.11.3 ATAN / ATANL

*Function* - Calculate an arc-tangent.

3.11.3.1 Inputs

IN : REAL/LREAL  Real value

3.11.3.2 Outputs

Q : REAL/LREAL  Result: arc-tangent of IN

3.11.3.3 Remarks

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

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

3.11.3.4 ST Language

Q := ATAN (IN);

3.11.3.5 FBD Language

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

3.11.3.6 FFLD Language

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

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

3.11.3.7 IL Language:

Op1: LD IN
ATAN
ATAN
ST Q  (* Q is: ATAN (IN) *)

See also

SIN  COS  TAN  ASIN  ACOS  ATAN2
3.11.4 ATAN2 / ATAN2L

*Function* - Calculate arc-tangent of Y/X

### 3.11.4.1 Inputs

- Y : REAL/LREAL  Real value
- X : REAL/LREAL  Real value

### 3.11.4.2 Outputs

- Q : REAL/LREAL  Result: arc-tangent of Y / X

### 3.11.4.3 Remarks

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

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

### 3.11.4.4 ST Language

```plaintext
Q := ATAN2 (IN);
```

### 3.11.4.5 FBD Language

![FBD Diagram]

### 3.11.4.6 FFLD Language

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

(* ENO keeps the same value as EN *)

### 3.11.4.7 IL Language

Op1: LD Y
ATAN2 X
ST Q  (* Q is: ATAN2 (Y / X) *)

See also

[SIN COS TAN ASIN ACOS ATAN]
3.11.5 COS / COSL

*Function* - Calculate a cosine.

### 3.11.5.1 Inputs

IN : REAL/LREAL  Real value

### 3.11.5.2 Outputs

Q : REAL/LREAL  Result: cosine of IN

### 3.11.5.3 Remarks

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

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

### 3.11.5.4 ST Language

Q := COS (IN);

### 3.11.5.5 FBD Language

![FBD Diagram]

### 3.11.5.6 FFLD Language

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

(* ENO keeps the same value as EN *)

![FFLD Diagram]

### 3.11.5.7 IL Language:

Op1: LD IN
COS
ST Q  (* Q is: COS (IN) *)

*See also*

SIN  TAN  ASIN  ACOS  ATAN  ATAN2
3.11.6 SIN / SINL

*Function* - Calculate a sine.

3.11.6.1 Inputs

IN : REAL/LREAL  Real value

3.11.6.2 Outputs

Q : REAL/LREAL  Result: sine of IN

3.11.6.3 Remarks

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

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

3.11.6.4 ST Language

Q := SIN (IN);

3.11.6.5 FBD Language

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

3.11.6.6 FFLD Language

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

(* ENO keeps the same value as EN *)

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

3.11.6.7 IL Language:

Op1: LD IN

SIN

ST Q  (* Q is: SIN (IN) *)

See also

COS  TAN  ASIN  ACOS  ATAN  ATAN2
3.11.7 TAN / TANL

Function - Calculate a tangent.

3.11.7.1 Inputs
IN : REAL/LREAL  Real value

3.11.7.2 Outputs
Q : REAL/LREAL  Result: tangent of IN

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

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

3.11.7.4 ST Language
Q := TAN (IN);

3.11.7.5 FBD Language

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

3.11.7.7 IL Language:
Op1: LD IN  
   TAN  
   TAN  
   ST Q  (* Q is: TAN (IN) *)

See also
SIN  COS  ASIN  ACOS  ATAN  ATAN2
3.11.8 UseDegrees

*Function* - Sets the unit for angles in all trigonometric functions.

### 3.11.8.1 Inputs

**IN** : **BOOL**

- If TRUE, turn all trigonometric functions to use degrees
- If FALSE, turn all trigonometric functions to use radians (default)

### 3.11.8.2 Outputs

**Q** : **BOOL**

- TRUE if functions use degrees before the call

### 3.11.8.3 Remarks

This function sets the working unit for the following functions:

- `SIN` sine
- `COS` cosine
- `TAN` tangent
- `ASIN` arc-sine
- `ACOS` arc-cosine
- `ATAN` arc-tangent
- `ATAN2` arc-tangent of Y / X

### 3.11.8.4 ST Language

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

### 3.11.8.5 FBD Language

```
IN   UseDegrees   Q
```

### 3.11.8.6 FFLD Language

```
IN [ UseDegrees Q ]
```

(* Input is the rung. The rung is the output *)

### 3.11.8.7 IL Language

```
Op1: LD IN
   UseDegrees
   ST Q
```
3.12 String operations

Below are the standard operators and functions that manage character strings:

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

Other functions are available for managing string tables as resources:

- **StringTable**: Select the active string table resource
- **LoadString**: Load a string from the active string table
3.12.1 ArrayToString / ArrayToStringU

**Function** - Copy an array of SINT to a STRING.

### 3.12.1.1 Inputs

- **SRC** : SINT  
  Source array of SINT small integers (USINT for ArrayToStringU)
- **DST** : STRING  
  Destination STRING
- **COUNT** : DINT  
  Numbers of characters to be copied

### 3.12.1.2 Outputs

- **Q** : DINT  
  Number of characters copied

### 3.12.1.3 Remarks

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

This function copies the COUNT first elements of the SRC array to the characters of the DST string. The function checks the maximum size of the destination string and adjust the COUNT number if necessary.

### 3.12.1.4 ST Language

Q := ArrayToString (SRC, DST, COUNT);

### 3.12.1.5 FBD Language

![FBD Diagram]

### 3.12.1.6 FFLD Language

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

(* ENO keeps the same value as EN *)

### 3.12.1.7 IL Language

*Not available*

**See also**

[StringToArray]
3.12.2 ASCII

Function - Get the ASCII code of a character within a string

3.12.2.1 Inputs
IN : STRING Input string
POS : DINT Position of the character within the string
(The first valid position is 1)

3.12.2.2 Outputs
CODE : DINT ASCII code of the selected character
or 0 if position is invalid

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

3.12.2.4 ST Language
CODE := ASCII (IN, POS);

3.12.2.5 FBD Language

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

3.12.2.7 IL Language:
Op1: LD IN
AND_Mask MSK
ST CODE

See also
CHAR
### 3.12.3 ATOH

**Function** - Converts string to integer using hexadecimal basis

#### 3.12.3.1 Inputs

**IN** : STRING  String representing an integer in hexadecimal format

#### 3.12.3.2 Outputs

**Q** : DINT  Integer represented by the string

#### 3.12.3.3 Truth table (examples)

<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>

#### 3.12.3.4 Remarks

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

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

#### 3.12.3.5 ST Language

Q := ATOH (IN);

#### 3.12.3.6 FBD Language

```
IN  ATOH  Q
```

#### 3.12.3.7 FFLD Language

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

(* ENO keeps the same value as EN *)

```
EN  ATOH  ENO
IN  Q
```

#### 3.12.3.8 IL Language:

```
Op1: LD IN
    ATOH
    ST Q
```

**See also**

HTOA
3.12.4 CHAR

*Function* - Builds a single character string

### 3.12.4.1 Inputs

- **CODE**: DINT, ASCII code of the wished character

### 3.12.4.2 Outputs

- **Q**: STRING, STRING containing only the specified character

### 3.12.4.3 Remarks

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

### 3.12.4.4 ST Language

\[
Q := \text{CHAR} (\text{CODE});
\]

### 3.12.4.5 FBD Language

![FBD Diagram]

### 3.12.4.6 FFLD Language

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

(* ENO is equal to EN *)

### 3.12.4.7 IL Language:

```
Op1: LD CODE
     CHAR
     ST Q
```

See also

ASCII
3.12.5 CONCAT Function

Function - Concatenate strings.

3.12.5.1 Inputs

- IN_1 : STRING Any string variable or constant expression
- ... IN_N : STRING Any string variable or constant expression

3.12.5.2 Outputs

- Q : STRING Concatenation of all inputs

3.12.5.3 Remarks

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

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

3.12.5.4 ST Language

Q := CONCAT ('AB', 'CD', 'E'); (* now Q is 'ABCDE' *)

3.12.5.5 FBD Language

3.12.5.6 FFLD Language

3.12.5.7 IL Language

Op1: FFLD 'AB'
    CONCAT 'CD', 'E'
    ST Q (* Q is now 'ABCDE' *)
3.12.6 CRC16

*Function* - calculates a CRC16 on the characters of a string

3.12.6.1 Inputs

\[ \text{IN} : \text{STRING} \quad \text{character string} \]

3.12.6.2 Outputs

\[ \text{Q} : \text{INT} \quad \text{CRC16 calculated on all the characters of the string.} \]

3.12.6.3 Remarks

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

The function calculates a Modbus CRC16, initialized at 16#FFFF value.

3.12.6.4 ST Language

\[ \text{Q} := \text{CRC16} (\text{IN}); \]

3.12.6.5 FBD Language

```
IN  CRC16  Q
```

3.12.6.6 FFLD Language

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

(* ENO is equal to EN *)

```
EN IN  CRC16  Q
```

3.12.6.7 IL Language:

Op1: LD  IN  CRC16
      ST  Q
3.12.7 DELETE

*Function* - Delete characters in a string.

### 3.12.7.1 Inputs

- **IN**: STRING  Character string
- **NBC**: DINT  Number of characters to be deleted
- **POS**: DINT  Position of the first deleted character (first character position is 1)

### 3.12.7.2 Outputs

- **Q**: STRING  Modified string.

### 3.12.7.3 Remarks

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

In IL, the first input (the string) must be loaded in the current result before calling the function. Other arguments are operands of the function, separated by commas.

### 3.12.7.4 ST Language

\[ Q := \text{DELETE} (\text{IN}, \text{NBC}, \text{POS}); \]

### 3.12.7.5 FBD Language

```
IN  NBC  POS  Q
DELETE
```

### 3.12.7.6 FFLD Language

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

(* ENO keeps the same value as EN *)

```
EN \[ \] \[ \] \[
IN  NBC  POS  Q  ENO
DELETE
```

### 3.12.7.7 IL Language:

```
Op1: LD  IN
  DELETE NBC, POS
  ST Q
```

**See also**

+ MLEN  INSERT  FIND  REPLACE  LEFT  RIGHT  MID
3.12.8 FIND

Function - Find position of characters in a string.

3.12.8.1 Inputs

<table>
<thead>
<tr>
<th>IN</th>
<th>STRING</th>
<th>Character string</th>
</tr>
</thead>
<tbody>
<tr>
<td>STR</td>
<td>STRING</td>
<td>Specific characters to search for within the STRING</td>
</tr>
</tbody>
</table>

3.12.8.2 Outputs

| POS | DINT | Position of the first character of STR in IN, or 0 if not found. |

3.12.8.3 Remarks

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

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

3.12.8.4 ST Language

POS := FIND (IN, STR);

3.12.8.5 FBD Language

```
IN   FIND  POS
STR
```

3.12.8.6 FFLD Language

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

```
EN  FIND  ENO
|   [    ]   |   {    }   |
IN  FIND  ENO
|   [    ]   |   {    }   |
STR  POS
```

3.12.8.7 IL Language:

```
Op1: LD  IN
     FIND  STR
     ST  POS
```

See also

+ MLEN  DELETE  INSERT  REPLACE  LEFT  RIGHT  MID
3.12.9 HTOA

Function - Converts integer to string using hexadecimal basis

3.12.9.1 Inputs
IN : DINT Integer value

3.12.9.2 Outputs
Q : STRING String representing the integer in hexadecimal format

3.12.9.3 Truth table (examples)

<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>

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

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

3.12.9.5 ST Language
Q := HTOA(IN);

3.12.9.6 FBD Language

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

3.12.9.8 IL Language:
Op1: LD IN
    HTOA
    ST Q

See also
ATOH
3.12.10 INSERT

*Function* - Insert characters in a string.

3.12.10.1 Inputs

IN : STRING Character string
STR : STRING String containing characters to be inserted
POS : DINT Position of the first inserted character (first character position is 1)

3.12.10.2 Outputs

Q : STRING Modified string.

3.12.10.3 Remarks

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

In IL, the first input (the string) must be loaded in the current result before calling the function. Other arguments are operands of the function, separated by commas.

3.12.10.4 ST Language

Q := INSERT (IN, STR, POS);

3.12.10.5 FBD Language

```
         INSERT
IN      STR      POS
         Q
```

3.12.10.6 FFLD Language

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

```
[EN] [INSERT] [ENO]
[  IN] [  STR]  [  Q]
[  POS]
```

3.12.10.7 IL Language:

Op1: LD IN
     INSERT STR, POS
     ST Q

See also

+ MLEN DELETE FIND REPLACE LEFT RIGHT MID
3.12.11 LEFT

*Function* - Extract characters of a string on the left.

**3.12.11.1 Inputs**

IN : STRING  Character string  
NBC : DINT    Number of characters to extract

**3.12.11.2 Outputs**

Q : STRING  String containing the first NBC characters of IN.

**3.12.11.3 Remarks**

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

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

**3.12.11.4 ST Language**

Q := LEFT (IN, NBC);

**3.12.11.5 FBD Language**

```
IN
NBC
LEFT
Q
```

**3.12.11.6 FFLD Language**

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

(* ENO keeps the same value as EN *)

```
EN
IN
NBC
LEFT
Q
ENO
```

**3.12.11.7 IL Language:**

Op1: LD  IN
    LEFT  NBC
    ST  Q

**See also**

+ MLEN  DELETE  INSERT  FIND  REPLACE  RIGHT  MID
3.12.12 LoadString

*Function* - Load a string from the active string table.

**3.12.12.1 Inputs**

**ID**: DINT ID of the string as declared in the string table

**3.12.12.2 Outputs**

**Q**: STRING Loaded string or empty string in case of error

**3.12.12.3 Remarks**

This function loads a string from the active string table and stores it into a STRING variable. The `StringTable()` function is used for selecting the active string table. The "ID" input (the string item identifier) is an identifier such as declared within the string table resource. You don't need to "define" again this identifier. The system does it for you.

**3.12.12.4 ST Language**

\[
Q := \text{LoadString}(\text{ID});
\]

**3.12.12.5 FBD Language**

```
ID       LoadString       Q
```

**3.12.12.6 FFLD Language**

```
EN [ LoadString ( ) ] ENO
| [ ID ] Q |
```

**3.12.12.7 IL Language:**

\[
\text{Op1: } \text{LD ID LoadString ST Q}
\]

**See also**

`StringTable` String tables
3.12.13 MID

*Function* - Extract characters of a string at any position.

### 3.12.13.1 Inputs

- IN : STRING  
  Character string
- NBC : DINT  
  Number of characters to extract
- POS : DINT  
  Position of the first character to extract (first character of IN is at position 1)

### 3.12.13.2 Outputs

- Q : STRING  
  String containing the first NBC characters of IN.

### 3.12.13.3 Remarks

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

In IL, the first input (the string) must be loaded in the current result before calling the function. Other arguments are operands of the function, separated by commas.

### 3.12.13.4 ST Language

\[ Q := \text{MID} \ (\text{IN}, \ \text{NBC}, \ \text{POS}); \]

### 3.12.13.5 FBD Language

```
MID
IN
NBC
POS
```

### 3.12.13.6 FFLD Language

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

(* ENO keeps the same value as EN *)

```
EN
[ ]
IN
NBC
POS
MID
ENO
( )
```

### 3.12.13.7 IL Language:

```
Op1: LD IN
    MID NBC, POS
    ST Q
```

See also

+ MLEN DELETE INSERT FIND REPLACE LEFT RIGHT
3.12.14 MLEN

*Function* - Get the number of characters in a string.

### Inputs

**IN** : STRING  Character string

### Outputs

**NBC** : DINT  Number of characters currently in the string. 0 if string is empty.

### 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.

### ST Language

NBC := MLEN (IN);

### FBD Language

```
IN  MLEN  NBC
```

### FFLD Language

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

(* ENO keeps the same value as EN *)

```
EN   MLEN   ENO
IN   ( )   NBC
```

### IL Language:

Op1: LD  IN
     MLEN
     ST  NBC

**See also**

+ DELETE  INSERT  FIND  REPLACE  LEFT  RIGHT  MID
3.12.15 REPLACE 

*Function* - Replace characters in a string.

**3.12.15.1 Inputs**

<table>
<thead>
<tr>
<th>IN</th>
<th>STRING</th>
<th>Character string</th>
</tr>
</thead>
<tbody>
<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>

**3.12.15.2 Outputs**

| Q  | STRING | Modified string. |

**3.12.15.3 Remarks**

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

In IL, the first input (the string) must be loaded in the current result before calling the function. Other arguments are operands of the function, separated by commas.

**3.12.15.4 ST Language**

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

**3.12.15.5 FBD Language**

![FBD Diagram]

**3.12.15.6 FFLD Language**

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

(* ENO keeps the same value as EN *)

**3.12.15.7 IL Language:**

Opt1: LD IN

REPLACE STR, NDEL, POS

ST Q

See also

+ MLEN DELETE INSERT FIND LEFT RIGHT MID
3.12.16 **RIGHT**

*Function* - Extract characters of a string on the right.

### 3.12.16.1 Inputs

| IN : STRING | Character string |
| NBC : DINT | Number of characters to extract |

### 3.12.16.2 Outputs

| Q : STRING | String containing the last NBC characters of IN. |

### 3.12.16.3 Remarks

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

In **IL**, the first input (the string) must be loaded in the current result before calling the function. The second argument is the operand of the function.

### 3.12.16.4 ST Language

Q := RIGHT(IN, NBC);

### 3.12.16.5 FBD Language

![FBD Diagram]

### 3.12.16.6 **FFLD** Language

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

(* ENO keeps the same value as EN *)

### 3.12.16.7 **IL** Language:

Op1: LD IN
    RIGHT NBC
    ST Q

*See also*

+ **MLEN**  **DELETE**  **INSERT**  **FIND**  **REPLACE**  **LEFT**  **MID**
3.12.17 StringTable

Function - Selects the active string table.

3.12.17.1 Inputs

| TABLE : STRING | Name of the Sting Table resource - must be a constant |
| COL : STRING   | Name of the column in the table - must be a constant |

3.12.17.2 Outputs

| OK : BOOL | TRUE if OK |

3.12.17.3 Remarks

This function selects a column of a valid String Table resource to become the active string table. The LoadString() function always refers to the active string table. Arguments must be constant string expressions and must fit to a declared string table and a valid column name within this table.

If you have only one string table with only one column defined in your project, you do not need to call this function as it will be the default string table anyway.

3.12.17.4 ST Language

OK := StringTable ('MyTable', 'FirstColumn');

3.12.17.5 FBD Language

![FBD Diagram]

3.12.17.6 FFLD Language

![FFLD Diagram]

3.12.17.7 IL Language:

Op1: LD 'MyTable'
StringTable 'First Column'
ST OK

See also

LoadString String tables
3.12.18 StringToArray / StringToArrayU

Function - Copies the characters of a STRING to an array of SINT.

3.12.18.1 Inputs
SRC : STRING  Source STRING
DST : SINT   Destination array of SINT small integers (USINT for StringToArrayU)

3.12.18.2 Outputs
Q : DINT  Number of characters copied

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

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

This function copies the characters of the SRC string to the first characters of the DST array. The function checks the maximum size destination arrays and reduces the number of copied characters if necessary.

3.12.18.4 ST Language
Q := StringToArray (SRC, DST);

3.12.18.5 FBD Language

```
Src  StringToArray
    Dst
```

3.12.18.6 FFLD Language

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

```
[ | ] [ ] StringToArray [ ]
EN
Src
DST
( | ) ( | )
ENO
Q
```

3.12.18.7 IL Language:

Op1: LD  SRC
StringToArray DST
ST  Q

See also
ArrayToString
3.13 UDP Functions for PDMM/PCMM & Simulator

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

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

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

See Wikipedia for more information on the UDP protocol.
3.13.1 udpAddrMake

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

3.13.1.2 Arguments

3.13.1.3.1 Input

<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>En</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>IPaddr</th>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPaddr</td>
<td>IP address in the form XXX.XXX.XXX.XXX</td>
<td>STRING</td>
<td>[0.0.0.0,255.255.255.255]</td>
<td>n/a</td>
<td>—</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>port</th>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>IP port number</td>
<td>DINT</td>
<td>[0,+65535]</td>
<td>n/a</td>
<td>—</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>add[]</th>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>add[]</td>
<td>Buffer where to store the UDP address (filled on output)</td>
<td>USINT</td>
<td>[0,32]</td>
<td>n/a</td>
<td>—</td>
</tr>
</tbody>
</table>
3.13.1.4.2 Output

<table>
<thead>
<tr>
<th>OK</th>
<th>Description</th>
<th>Returns true when the function successfully executes. See Function - General rules.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data type</td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td>Unit</td>
<td>n/a</td>
<td></td>
</tr>
</tbody>
</table>

3.13.1.5 Examples

3.13.1.6.1 Structured Text

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

3.13.1.7.2 Ladder Diagram

3.13.1.8.3 Function Block Diagram
3.13.2 udpClose

3.13.2.1 Description
This function closes a socket.

3.13.2.2 Arguments

3.13.2.3.1 Input

<table>
<thead>
<tr>
<th>En</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Execute the function</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data type</th>
<th>BOOL</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sock</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ID of the socket</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data type</th>
<th>DINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>[0,+65535]</td>
</tr>
<tr>
<td>Unit</td>
<td>n/a</td>
</tr>
<tr>
<td>Default</td>
<td>—</td>
</tr>
</tbody>
</table>

3.13.2.4.2 Output

<table>
<thead>
<tr>
<th>OK</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns true when the function successfully executes. See Function - General rules.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data type</th>
<th>BOOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit</td>
<td>n/a</td>
</tr>
</tbody>
</table>
3.13.2.5 Examples

3.13.2.6.1 Structured Text

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

3.13.2.7.2 Ladder Diagram

3.13.2.8.3 Function Block Diagram
3.13.3 udpCreate

3.13.3.1 Description
This function creates a UDP socket.

3.13.3.2 Arguments

3.13.3.3.1 Input

<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>
<tr>
<td>port</td>
<td>Description</td>
<td>DINT</td>
<td>[0,+65535]</td>
<td>n/a</td>
<td>—</td>
</tr>
</tbody>
</table>

3.13.3.4.2 Output

<table>
<thead>
<tr>
<th>OK</th>
<th>Description</th>
<th>Data type</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns true when the function successfully executes. See Function - General rules</td>
<td>BOOL</td>
<td>n/a</td>
</tr>
<tr>
<td>sock</td>
<td>Description</td>
<td>DINT</td>
<td>n/a</td>
</tr>
</tbody>
</table>

ID of the new socket
3.13.3.5 Examples

3.13.3.6.1 Structured Text

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

3.13.3.7.2 Ladder Diagram

3.13.3.8.3 Function Block Diagram
3.13.4 udpIsValid

3.13.4.1 Description
This function states whether a socket is valid or not.

3.13.4.2 Arguments

3.13.4.3.1 Input

<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>En</td>
<td>Execute the function</td>
<td>BOOL</td>
<td>[0,1]</td>
<td>n/a</td>
<td>—</td>
</tr>
<tr>
<td>sock</td>
<td>ID of the socket</td>
<td>DINT</td>
<td>[0,+65535]</td>
<td>n/a</td>
<td>—</td>
</tr>
</tbody>
</table>

3.13.4.4.2 Output

<table>
<thead>
<tr>
<th>OK</th>
<th>Description</th>
<th>Data type</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>Returns true when the function successfully executes. See Function - General rules.</td>
<td>BOOL</td>
<td>n/a</td>
</tr>
</tbody>
</table>
### 3.13.4.5 Examples

#### 3.13.4.6.1 Structured Text

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

#### 3.13.4.7.2 Ladder Diagram

![Ladder Diagram]

#### 3.13.4.8.3 Function Block Diagram

![Function Block Diagram]
3.13.5 **udpRcvFrom**

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

3.13.5.2 **Arguments**

3.13.5.3.1 **Input**

<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>En</td>
<td>Execute the function</td>
<td>BOOL</td>
<td>[0,1]</td>
<td>n/a</td>
<td>—</td>
</tr>
<tr>
<td>sock</td>
<td>Description</td>
<td>DINT</td>
<td>[0,+65535]</td>
<td>n/a</td>
<td>—</td>
</tr>
<tr>
<td>nb</td>
<td>Maximum number of characters received</td>
<td>DINT</td>
<td>[0,+65535]</td>
<td>n/a</td>
<td>—</td>
</tr>
<tr>
<td>add[]</td>
<td>Buffer containing the UDP address of the transmitter (filled on output)</td>
<td>USINT</td>
<td>[0,32]</td>
<td>n/a</td>
<td>—</td>
</tr>
<tr>
<td>data</td>
<td>Buffer where to store received characters</td>
<td>STRING</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 3.13.5.4.2 Output

<table>
<thead>
<tr>
<th>Description</th>
<th>Returns true when the function successfully executes. See Function - General rules.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data type</td>
<td>BOOL</td>
</tr>
<tr>
<td>Unit</td>
<td>n/a</td>
</tr>
</tbody>
</table>

### 3.13.5.5 Examples

#### 3.13.5.6.1 Structured Text

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

#### 3.13.5.7.2 Ladder Diagram

```
<table>
<thead>
<tr>
<th>En</th>
<th>udpRcvFrom</th>
<th>En</th>
<th>OK</th>
<th>ok</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Socket_Number</td>
<td>sock</td>
<td>Q</td>
<td>Number/BytesReceived</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nb</td>
<td>r8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>add[]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>data</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

#### 3.13.5.8.3 Function Block Diagram

```
```

### Range

<table>
<thead>
<tr>
<th>Range</th>
<th>0,255</th>
</tr>
</thead>
</table>

### Default

<table>
<thead>
<tr>
<th>Default</th>
<th>_</th>
</tr>
</thead>
</table>

### Unit

<table>
<thead>
<tr>
<th>Unit</th>
<th>n/a</th>
</tr>
</thead>
</table>

### Data type

<table>
<thead>
<tr>
<th>Data type</th>
<th>BOOL</th>
</tr>
</thead>
</table>

### Unit

<table>
<thead>
<tr>
<th>Unit</th>
<th>n/a</th>
</tr>
</thead>
</table>

### Description

**Q**

Actual number of received characters

<table>
<thead>
<tr>
<th>Data type</th>
<th>DINT</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Unit</th>
<th>n/a</th>
</tr>
</thead>
</table>

### Examples

#### 3.13.5.6.1 Structured Text

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

#### 3.13.5.7.2 Ladder Diagram

```
```

#### 3.13.5.8.3 Function Block Diagram

```
```
### 3.13.6 udpRcvFromArray

#### Description
This function receives an array of bytes.

#### Arguments

##### 3.13.6.3.1 Input

<table>
<thead>
<tr>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>En</td>
<td>BOOL</td>
<td>[0,1]</td>
<td>n/a</td>
<td>—</td>
</tr>
<tr>
<td>sock</td>
<td>DINT</td>
<td>[0,+65535]</td>
<td>n/a</td>
<td>—</td>
</tr>
<tr>
<td>nb</td>
<td>DINT</td>
<td>[0,+65535]</td>
<td>n/a</td>
<td>—</td>
</tr>
<tr>
<td>add[]</td>
<td>USINT</td>
<td>[0,32]</td>
<td>n/a</td>
<td>—</td>
</tr>
<tr>
<td>data[]</td>
<td>USINT</td>
<td>[0,+65535]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 3.13.6.4.2 Output

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

<table>
<thead>
<tr>
<th>Q</th>
<th>Description</th>
<th>Number of bytes received</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data type</td>
<td>DINT</td>
<td></td>
</tr>
<tr>
<td>Unit</td>
<td>n/a</td>
<td></td>
</tr>
</tbody>
</table>

### 3.13.6.5 Examples

#### 3.13.6.6.1 Structured Text

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

#### 3.13.6.7.2 Ladder Diagram

![Ladder Diagram](image)

#### 3.13.6.8.3 Function Block Diagram

![Function Block Diagram](image)
### 3.13.7 udpSendTo

#### 3.13.7.1 Description
This function sends UDP data to a server.

#### 3.13.7.2 Arguments

##### 3.13.7.3.1 Input

<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>En</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>sock</th>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>sock</td>
<td>ID of the client socket</td>
<td>DINT</td>
<td>[0,+65535]</td>
<td>n/a</td>
<td>—</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>nb</th>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nb</td>
<td>Number of bytes of data to send</td>
<td>DINT</td>
<td>[0,65535]</td>
<td>n/a</td>
<td>—</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>add[]</th>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>add[]</td>
<td>Buffer containing the UDP address</td>
<td>USINT</td>
<td>[0,32]</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>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>The characters to send</td>
<td>STRING</td>
<td>[0,255]</td>
</tr>
</tbody>
</table>
### 3.13.7.4.2 Output

<table>
<thead>
<tr>
<th>Description</th>
<th>Returns true when the function successfully executes. See <a href="#">Function - General rules</a>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data type</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

### 3.13.7.5 Examples

#### 3.13.7.6.1 Structured Text

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

#### 3.13.7.7.2 Ladder Diagram

![Ladder Diagram](image)

#### 3.13.7.8.3 Function Block Diagram

![Function Block Diagram](image)
3.13.8 udpSendToArray

3.13.8.1 Description
This function sends an array of bytes.

3.13.8.2 Arguments

3.13.8.3.1 Input

<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>En</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>sock</th>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>sock</td>
<td>Socket number, return value from &quot;udpCreate&quot; (p. 214)</td>
<td>DINT</td>
<td>[0,+65535]</td>
<td>n/a</td>
<td>—</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>nb</th>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nb</td>
<td>Number of bytes to be transferred</td>
<td>DINT</td>
<td>[0,+65535]</td>
<td>n/a</td>
<td>—</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>add[]</th>
<th>Description</th>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>add[]</td>
<td>Array which contains information about the server</td>
<td>USINT</td>
<td>[0,32]</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>
</tr>
</thead>
<tbody>
<tr>
<td>data[]</td>
<td>Array of bytes to be transferred</td>
<td>USINT</td>
<td>[0,+65535]</td>
</tr>
</tbody>
</table>
### 3.13.8.4.2 Output

<table>
<thead>
<tr>
<th>OK</th>
<th>Description</th>
<th>Data type</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>Returns true when the function successfully executes. See Function - General rules.</td>
<td>BOOL</td>
<td>n/a</td>
</tr>
</tbody>
</table>

#### Data type

**BOOL**

#### Unit

n/a

### 3.13.8.5 Examples

#### 3.13.8.6.1 Structured Text

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

#### 3.13.8.7.2 Ladder Diagram

![Ladder Diagram](image)

#### 3.13.8.8.3 Function Block Diagram

![Function Block Diagram](image)
## 4 PLC Advanced Libraries

Below are the standard blocks that perform advanced operations.

### Analog signal processing:

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average / AverageL</td>
<td>Calculate the average of signal samples</td>
</tr>
<tr>
<td>Integral</td>
<td>Calculate the integral of a signal</td>
</tr>
<tr>
<td>Derivate</td>
<td>Derive a signal</td>
</tr>
<tr>
<td>PID</td>
<td>PID loop</td>
</tr>
<tr>
<td>Ramp</td>
<td>Ramp signal</td>
</tr>
<tr>
<td>Rand</td>
<td>Give a Random value modulo the input value</td>
</tr>
<tr>
<td>Lim_Alrm</td>
<td>Low / High level detection</td>
</tr>
<tr>
<td>Hyst, HysterAcc</td>
<td>Hysteresis calculation</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>CurveLin</td>
<td>Linear interpolation on a curve</td>
</tr>
<tr>
<td>SurfLin</td>
<td>Linear interpolation on a surface</td>
</tr>
</tbody>
</table>

### Alarm management:

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lim_Alrm</td>
<td>Low / High level detection</td>
</tr>
<tr>
<td>Alarm_M</td>
<td>Alarm with manual reset</td>
</tr>
<tr>
<td>&quot;ALARM_A&quot; (p. 228)</td>
<td>Alarm with automatic reset</td>
</tr>
</tbody>
</table>

### Data collections and serialization:

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StackInt</td>
<td>Stack of integers</td>
</tr>
<tr>
<td>FIFO</td>
<td>&quot;First in / first out&quot; list</td>
</tr>
<tr>
<td>LIFO</td>
<td>&quot;Last in / first out&quot; stack</td>
</tr>
<tr>
<td>&quot;SerializeIn&quot; (p. 286)</td>
<td>Extract data from a binary frame.</td>
</tr>
<tr>
<td>&quot;SerializeOut&quot; (p. 288)</td>
<td>Write data to a binary frame.</td>
</tr>
</tbody>
</table>

### Data Logging:

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogFileCSV</td>
<td>Log values of variables to a CSV file</td>
</tr>
</tbody>
</table>

### Special operations:

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetSysInfo</td>
<td>Get system information</td>
</tr>
<tr>
<td>Printf</td>
<td>Trace messages</td>
</tr>
<tr>
<td>CycleStop</td>
<td>Sets the application in cycle stepping mode</td>
</tr>
<tr>
<td>FatalStop</td>
<td>Breaks the cycle and stop with fatal error</td>
</tr>
<tr>
<td>EnableEvents</td>
<td>Enable / disable produced events for binding</td>
</tr>
<tr>
<td>ApplyRecipeColumn</td>
<td>Apply the values of a column from a recipe file</td>
</tr>
<tr>
<td>Block</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>VLID</td>
<td>Get the ID of an embedded list of variables</td>
</tr>
<tr>
<td>SigID</td>
<td>Get the ID of a signal resource</td>
</tr>
</tbody>
</table>

Communication:
- **AS-interface**

Others:
- **Real Time Clock**
- "File Management Functions" (p. 252)
4.1 **ALARM_A**

*Function Block* - Alarm with automatic reset

4.1.1 Inputs

<table>
<thead>
<tr>
<th>IN</th>
<th>BOOL</th>
<th>Process signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>BOOL</td>
<td>Acknowledge command</td>
</tr>
</tbody>
</table>

4.1.2 Outputs

<table>
<thead>
<tr>
<th>Q</th>
<th>BOOL</th>
<th>TRUE if alarm is active</th>
</tr>
</thead>
<tbody>
<tr>
<td>QACK</td>
<td>BOOL</td>
<td>TRUE if alarm is acknowledged</td>
</tr>
</tbody>
</table>

4.1.3 Sequence

![Sequence Diagram]

4.1.4 Remarks

Combine this block with the **LIM_ALRM** block for managing analog alarms.

4.1.5 ST Language

```st
(* MyALARM is declared as an instance of ALARM_A function block *)
MyALARM (IN, ACK);
Q := MyALARM.Q;
QACK := MyALARM.QACK;
```

4.1.6 FBD Language

```
   IN  -- ALARM_A
      |
      V  Q  QACK
```
4.1.7 FFLD Language

4.1.8 IL Language

```plaintext
(* MyALARM is declared as an instance of ALARM_A function block *)
Op1: CAL
MyALARM (IN, ACK)
FFLD MyALARM.Q
ST Q
FFLD MyALARM.QACK
ST QACK
```

See also
ALARM_M, LIM_ALRM
4.2 ALARM_M

*Function Block* - Alarm with manual reset

4.2.1 Inputs

<table>
<thead>
<tr>
<th>IN</th>
<th>BOOL</th>
<th>Process signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>BOOL</td>
<td>Acknowledge command</td>
</tr>
<tr>
<td>RST</td>
<td>BOOL</td>
<td>Reset command</td>
</tr>
</tbody>
</table>

4.2.2 Outputs

<table>
<thead>
<tr>
<th>Q</th>
<th>BOOL</th>
<th>TRUE if alarm is active</th>
</tr>
</thead>
<tbody>
<tr>
<td>QACK</td>
<td>BOOL</td>
<td>TRUE if alarm is acknowledged</td>
</tr>
</tbody>
</table>

4.2.3 Sequence

![Sequence Diagram]

4.2.4 Remarks

Combine this block with the LIM_ALRM block for managing analog alarms.

4.2.5 ST Language

```plaintext
(* MyALARM is declared as an instance of ALARM_M function block *)
MyALARM (IN, ACK, RST);
Q := MyALARM.Q;
QACK := MyALARM.QACK;
```

4.2.6 FBD Language

![FB Diagram]
4.2.7 FFLD Language

4.2.8 IL Language

(* MyALARM is declared as an instance of ALARM_M function block *)
Op1: CAL
MyALARM (IN, ACK, RST)
FFLD MyALARM.Q
ST Q
FFLD MyALARM.QACK
ST QACK

See also
ALARM_A LIM_ALRM
4.3 ApplyRecipeColumn

*Function* - Apply the values of a column from a recipe file

### 4.3.1 Inputs

- **FILE** : STRING  
  Path name of the recipe file (.RCP or .CSV) - *must be a constant value!*

- **COL** : DINT  
  Index of the column in the recipe (0 based)

#### See an example of RCP file

```plaintext
@COLNAME=Col3 Col4
@SIZECOL1=100
@SIZECOL2=100
@SIZECOL3=100
@SIZECOL4=100
bCommand
tPerio
bFast
Blink1
test_var
bOut
@EXPANDED=Blink1
```

#### See an example of CSV file

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

```plaintext
comment lines here
TravelSpeed;100;200;300;400;500
MasterAbsPos;0;45;90;135;180
MasterDeltaPos;0;90;180;270;360
MachineSpeed;50;100;150;200;250
MachineState;0;0;1;1;2
```

**NOTE**

For your CSV file to be valid, ensure the data are separated with *semicolons* (and not commas).

Usage in a FFLD program where column 3 is selected
Column 3 corresponds to column E in the Excel sheet because this parameter is 0 based.

<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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>TravelSpeed</td>
<td>400.0000</td>
<td>LREAL</td>
</tr>
<tr>
<td>MasterAbsPos</td>
<td>135.0000</td>
<td>LREAL</td>
</tr>
<tr>
<td>MasterDeltaPos</td>
<td>270.0000</td>
<td>LREAL</td>
</tr>
<tr>
<td>MachineSpeed</td>
<td>200.0000</td>
<td>LREAL</td>
</tr>
<tr>
<td>Ass1Status</td>
<td>447</td>
<td>DINT</td>
</tr>
<tr>
<td>Ass2Status</td>
<td>447</td>
<td>DINT</td>
</tr>
<tr>
<td>MachineState</td>
<td>1</td>
<td>DINT</td>
</tr>
</tbody>
</table>

4.3.2 Outputs

OK : BOOL       TRUE if OK - FALSE if parameters are invalid

4.3.3 Remarks

- The 'FILE' input is a constant string expression specifying the path name of a valid .RCP or .CSV file. If no path is specified, the file is assumed to be located in the project folder. RCP files are created using an external recipe editor. CSV files can be created using EXCEL or NOTEPAD.
- In CSV files, the first line must contain column headers, and is ignored during compiling. There is one variable per line. The first column contains the symbol of the variable. Other columns are values.
- If a cell is empty, it is assumed to be the same value as the previous (left side) cell. If it is the first cell of a row, it is assumed to be null (0 or FALSE or empty string).
- In FFLD language, the operation is executed only if the input rung (EN) is TRUE. The output rung is the result of the function.

**IMPORTANT**

Recipe files are read at compiling time and are embedded into the downloaded application code. This implies that a modification performed in the recipe file after downloading is not taken into account by the application.

4.3.4 ST Language
OK := ApplyRecipeColumn ('MyFile.rcp', COL);
4.3.5 FBD Language

```
File

ApplyRecipeColumn

Col

OK
```

4.3.6 FFLD Language

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

```
Op1:
LD 'MyFile.rcp'
ApplyRecipeColumn COL
ST
OK
```

4.3.7 IL Language
4.4 AS-interface Functions

The following functions enable special operation on AS-i networks:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASiReadPP</td>
<td>read permanent parameters of an AS-i slave</td>
</tr>
<tr>
<td>ASiWritePP</td>
<td>write 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>ASiReadPI</td>
<td>read actual parameters of an AS-i slave</td>
</tr>
<tr>
<td>ASiStorePI</td>
<td>store actual parameters as permanent parameters</td>
</tr>
</tbody>
</table>

**IMPORTANT**

AS-i networking may be not available on some targets. Please refer to OEM instructions for further details about available features.

**Interface**

```c
Params := ASiReadPP (Master, Slave);
bOK := ASiWritePP (Master, Slave, Params);
bOK := ASiSendParam (Master, Slave, Params);
Params := ASiReadPI (Master, Slave);
bOK := ASiStorePI (Master);
```

**Arguments**

- **Master** : DINT  Index of the AS-i master (1..N) such as shown in configuration
- **Slave** : DINT   Address of the AS-i slave (1..32 / 33..63)
- **Params** : DINT  Value of AS-i parameters
- **bOK** : BOOL     TRUE if successful
4.5 AVERAGE / AVERAGEL [PLCopen]

*Function Block* - Calculates the average of signal samples.

4.5.1 Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>BOOL</td>
<td>Enabling command</td>
</tr>
<tr>
<td>XIN</td>
<td>REAL</td>
<td>Input signal</td>
</tr>
<tr>
<td>N</td>
<td>DINT</td>
<td>Number of samples stored for average calculation - Cannot exceed 128</td>
</tr>
</tbody>
</table>

4.5.2 Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOUT</td>
<td>REAL</td>
<td>Average of the stored samples (*)</td>
</tr>
</tbody>
</table>

(*) AVERAGEL has LREAL arguments.

4.5.3 Remarks

The average is calculated according to the number of stored samples, which can be less than N when the block is enabled. By default the number of samples is 128.

The "N" input (or the number of samples) is taken into account only when the RUN input is FALSE.

**tip**

The "RUN" needs to be reset after a change in the number of samples. You should cycle the RUN input when you first call this function, this will clear the default.

**note**

In FFLD language, the input rung is the RUN command. The output rung keeps the state of the input rung.

4.5.4 ST Language

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

4.5.5 FBD Language

```
RUN  XIN  N  AVERAGE  XOUT
```

4.5.6 FFLD Language

```
(* ENO has the same state as RUN *)
RUN [XIN N] [AVERAGE ( )] XOUT
```
4.5.7 IL Language:

(* MyAve is a declared instance of AVERAGE function block *)
Op1: CAL MyAve (RUN, XIN, N)
       FFLD MyAve.XOUT
       ST   XOUT

See also
INTEGRAL   DERIVATE   LIM_ALRM   HYSTER   STACKINT
4.6 **CurveLin**

*Function block*- Linear interpolation on a curve.

### 4.6.1 Inputs

- **X**: REAL  X coordinate of the point to be interpolated.
- **XAxis**: REAL[]  X coordinates of the known points of the X axis.
- **YVal**: REAL[]  Y coordinate of the points defined on the X axis.

### 4.6.2 Outputs

- **Y**: REAL  Interpolated Y value corresponding to the X input
- **OK**: BOOL  TRUE if successful.
- **ERR**: DINT  Error code if failed - 0 if OK.

### 4.6.3 Remarks

This function performs linear interpolation in between a list of points defined in the XAxis single dimension array. The output Y value is an interpolation of the Y values of the two rounding points defined in the X axis. Y values of defined points are passed in the YVal single dimension array.

Values in XAxis must be sorted from the smallest to the biggest. There must be at least two points defined in the X axis. YVal and XAxis input arrays must have the same dimension.

In case the X input is less than the smallest defined X point, the Y output takes the first value defined in YVal and an error is reported. In case the X input is greater than the biggest defined X point, the Y output takes the last value defined in YVal and an error is reported.

The ERR output gives the cause of the error if the function fails:

<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>
4.7 DERIVATE

*Function Block* - Computes the derivative of a signal with respect to time.

The time unit is seconds. The output signal has the units of the input signal divided by seconds. The DERIVATE block samples the input signal at a maximum rate of 1 millisecond.

### 4.7.1 Inputs

- **RUN**: BOOL  Run command: TRUE=derivate / FALSE=hold
- **XIN**: REAL  Input signal
- **CYCLE**: TIME  Sampling period (must not be less than the target cycle timing)

### 4.7.2 Outputs

- **XOUT**: REAL  Output signal

### 4.7.3 Remarks

In FFLD language, the input rung is the RUN command. The output rung keeps the state of the input rung.

### 4.7.4 ST Language

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

```st
MyDerv (RUN, XIN, CYCLE);
XOUT := MyDerv.XOUT;
```

### 4.7.5 FBD Language

```
RUN  XIN  CYCLE
    |
    | DERIVATE
    |
    |    |
    |    |        Q
    |    |        ( )
    |    |        XOUT
```

### 4.7.6 FFLD Language

(* ENO has the same state as RUN *)

```
RUN
  |
  | [ DERIVATE ( ) ]
  |
  |    |
  |    |
  |    | XOUT
  |
  |    |
  |    |
  |    |
  |
XIN
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
4.8 EnableEvents

*Function* - Enable or disable the production of events for binding (runtime to runtime variable exchange)

4.8.1 Inputs

EN : BOOL  TRUE to enable events / FALSE to disable events

4.8.2 Outputs

ENO : BOOL  Echo of EN input

4.8.3 Remarks

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

In FFLD language, the input rung (EN) enables the event production, and the output rung keeps the state of the input rung. In IL language, the input must be loaded before the function call.

4.8.4 ST Language

ENO := EnableEvents (EN);

4.8.5 FBD Language

```
EnableEvents
```

4.8.6 FFLD Language

(* Events are enables if EN is TRUE *)

(* ENO has the same value as EN *)

```
EN [ EnableEvents ] ENO

```

4.8.7 IL Language:

Op1: LD EN
    EnableEvents
    ST ENO
4.9 FIFO  

*Function block* - Manages a "first in / first out" list

### 4.9.1 Inputs

<table>
<thead>
<tr>
<th>Parameter</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>IN</td>
<td>ANY</td>
<td>Value to be pushed</td>
</tr>
<tr>
<td>@Tail</td>
<td>ANY</td>
<td>Value of the oldest pushed value - updated after call!</td>
</tr>
<tr>
<td>Buf[]</td>
<td>ANY</td>
<td>Array for storing values</td>
</tr>
</tbody>
</table>

### 4.9.2 Outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPTY</td>
<td>BOOL</td>
<td>TRUE if the list 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 list</td>
</tr>
<tr>
<td>pRead</td>
<td>DINT</td>
<td>Index in the buffer of the oldest pushed value</td>
</tr>
<tr>
<td>pWrite</td>
<td>DINT</td>
<td>Index in the buffer of the next push position</td>
</tr>
</tbody>
</table>

### 4.9.3 Remarks

IN, @Tail and Buf[] must have the same data type and cannot be STRING.

The @Tail argument specifies a variable which is filled with the oldest push value after the block is called.

Values are stored in the "BUF" array. Data is arranged as a roll over buffer and is never shifted or reset. Only read and write pointers and pushed values are updated. The maximum size of the list is the dimension of the array.

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

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

### 4.9.4 ST Language

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

4.9.5 FBD Language

```
myfifo
  push Empty
  pop OFLO
  rst Count
  in pRead
  @tail pWrite
  buf[]
```

4.9.6 FFLD Language

```
MyFifo
  push Empty
  pop OFLO
  rst Count
  in pRead
  @tail pWrite
  buf[]
```

4.9.7 IL Language

```
(* MyFIFO is a declared instance of FIFO function block *)
Op1: CAL MyFIFO (PUSH, POP, RST, IN, @Tail, BUFF[])
FFLD MyFIFO.EMPTY
ST EMPTY
FFLD MyFIFO.OFLO
ST OFLO
FFLD MyFIFO.COUNT
ST COUNT
FFLD MyFIFO.PREAD
ST PREAD
FFLD MyFIFO.PWRITE
ST PWRITE
```

See also

LIFO

4.10 FilterOrder1

Function block - first order filter

4.10.1 Inputs
XIN : REAL  Input analog value  
GAIN : REAL  Transformation gain

### 4.10.2 Outputs

XOUT : REAL  Output signal

### 4.10.3 Remarks

The operation performed is:

\[
\text{Output} = (\text{Input} \times \text{Gain}) + (\text{OutputPrev} \times (1-\text{Gain}))
\]

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

### 4.10.4 ST Language

Filt1 is a declared instance of FilterOrder1 function block.

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

### 4.10.5 Example
4.11 File Management

File Management functions provide the ability to do the following:

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

*Please note that a remote drive connection is setup through the web server.*

**TIP**

- If needed, functions to parse out information from a file using a string format can be found in "String operations" (p. 190).
- If the file is in .CSV format, the following functions can be used: "LogFileCSV" (p. 268), "ApplyRecipeColumn" (p. 232)
- You can create, store, and retrieve recipes and other data using:
  - the AKI Terminals. For more information see the KVB manual.
  - through an external bus connection to the PxMM with a supported fieldbus, such as UDP or HTTP.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;F_AOPEN&quot; (p. 252)</td>
<td>Create or open a file in append mode</td>
</tr>
<tr>
<td>&quot;F_CLOSE&quot; (p. 252)</td>
<td>Close an open file</td>
</tr>
<tr>
<td>&quot;F_COPY&quot; (p. 252)</td>
<td>Copy a file</td>
</tr>
<tr>
<td>&quot;F_DELETE&quot; (p. 253)</td>
<td>Remove a file</td>
</tr>
<tr>
<td>&quot;F_EOF&quot; (p. 253)</td>
<td>Test if the end of the file is reached in a file that is open for reading</td>
</tr>
<tr>
<td>&quot;F_EXIST&quot; (p. 253)</td>
<td>Test if a file exists</td>
</tr>
<tr>
<td>F_GETSIZE (p. 253)</td>
<td>Get the size of a file</td>
</tr>
<tr>
<td>&quot;F RENAME&quot; (p. 253)</td>
<td>Rename a file</td>
</tr>
<tr>
<td>&quot;F ROPEN&quot; (p. 254)</td>
<td>Open a file for reading</td>
</tr>
<tr>
<td>&quot;F SEEK&quot; (p. 254)</td>
<td>Set the current position of a file</td>
</tr>
<tr>
<td>&quot;F_WOPEN&quot; (p. 254)</td>
<td>Create or reset a file and open it for writing</td>
</tr>
<tr>
<td>&quot;FA_READ&quot; (p. 255)</td>
<td>Read a DINT integer from a binary file</td>
</tr>
<tr>
<td>&quot;FA_WRITE&quot; (p. 255)</td>
<td>Write a DINT integer to a binary file</td>
</tr>
<tr>
<td>&quot;FB_READ&quot; (p. 255)</td>
<td>Read binary data from a file</td>
</tr>
<tr>
<td>&quot;FB_WRITE&quot; (p. 255)</td>
<td>Write binary data to a file</td>
</tr>
<tr>
<td>&quot;FM_READ&quot; (p. 256)</td>
<td>Read a string value from a text file</td>
</tr>
<tr>
<td>&quot;FM_WRITE&quot; (p. 256)</td>
<td>Write a string value to a text file</td>
</tr>
<tr>
<td>&quot;SD_ISREADY&quot; (p. 257)</td>
<td>Check that the SD card is ready for read/write</td>
</tr>
<tr>
<td>&quot;SD_MOUNT&quot; (p. 257)</td>
<td>Mount an SD card</td>
</tr>
<tr>
<td>&quot;SD_UNMOUNT&quot; (p. 257)</td>
<td>Unmount an SD card</td>
</tr>
</tbody>
</table>

Each file is identified in the application by a unique handle manipulated as a DINT value. The file handles are allocated by the target system. Handles are returned by the Open functions and used in all other calls for identifying the file.

**Related function blocks:**

LogFileCSV log values of variables to a CSV file
These functions can have a serious impact on CPU load and the life expectancy of a flash drive. **It is highly recommended that these be used on an event basis, and not at every PLC cycle.** Files are opened and closed directly by the Operating System of the target. Opening some files can be dangerous for system safety and integrity. The number of open files (from "F_AOPEN" (p. 252), "F_ROPEN" (p. 254), and "F_WOPEN" (p. 254)) is limited by the resources available on the target system.

Ensure that each file successfully opened using "F_AOPEN" (p. 252), "F_ROPEN" (p. 254), and "F_WOPEN" (p. 254) has a corresponding "F_CLOSE" (p. 252) to close the file. Closing the file will release the file ID, making it available for operations on other files.

Opening a file with "F_AOPEN" (p. 252), "F_ROPEN" (p. 254), and "F_WOPEN" (p. 254) can be unsuccessful (invalid path or file name, too many open files...) Your application must check the file ID for a NULL value. If the file ID is NULL (zero), then file read or write operations will fail.

File management may be unavailable on some targets.

Memory on the SD card is available in addition to the existing flash memory.

Valid paths for storing files depend on the target implementation.

Error messages are logged in the Controller log section of KAS Runtime where there is a failure in any related function block.

Using the KAS Simulator, all pathnames are ignored, and files are stored in a reserved directory. Only the file name passed to the Open functions is taken into account.

AKD PDMM / PCMM files are **big endian**.

Be sure to review "File Path Conventions" (p. 248) so you understand hardware-based functional differences.
4.11.1 SD Card Access

Files may be written to and read from an SD card. This is typically used for storing a firmware image for Recovery Mode.

To use an SD card on the controller:

1. Ensure that the SD card is inserted
2. Mount the card using "SD_MOUNT" (p. 257)
3. Ensure the card is accessible using "SD_ISREADY" (p. 257) before performing a read or write action
4. Unmount the card, if desired, using "SD_UNMOUNT" (p. 257) after performing read/write actions
4.11.2 File Path Conventions

Depending upon the system used, paths to file locations may be defined as either absolute (C://dir1/file1) or relative paths (/dir1/file1). Not all systems handle all options, and the paths will vary depending upon the system.

<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 or PCMM</td>
<td>❌</td>
<td>✔</td>
<td>There is no support for creating directories on the controller. Any path provided to the function blocks, file1 for example, will be appended with the default userdata folder which is: /mount/flash/userdata</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 with the default userdata folder, which is: &lt;User Directory&gt;/Kollmorgen/Kollmorgen Automation Suite/Sinope Simulator/Application/userdata</td>
</tr>
<tr>
<td>PAC</td>
<td>✔</td>
<td>✔</td>
<td>When a relative path is provided to the function blocks, the path is appended with the default userdata folder, which is: &lt;User Directory&gt;/Kollmorgen/Kollmorgen Automation Suite/Sinope Runtime/Application/userdata</td>
</tr>
</tbody>
</table>

See Also:

- "Shared Directory Path Conventions" (p. 249)
- "SD Card Path Conventions" (p. 250)
- "File Name Warning & Limitations" (p. 251)
4.11.2.1 Shared Directory Path Conventions

The AKD PDMM and PCMM support access to a shared directory on a remote computer. To access files in a shared directory from the AKD PDMM or PCMM use '/mount/shared' at the beginning of the path, before the shared directory's relative path and file name:

'/mount/shared/directory/filename'

<table>
<thead>
<tr>
<th>Valid Paths</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>/mount/shared</td>
<td>The path is not case sensitive, so /MOUNT/SHARED, MOUNT/SHARED/, etc. are also valid.</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>

Example 1: Opening the file example.txt from a shared directory on a remote computer.

```plaintext
fileID := F_AOPEN('/mount/shared/example.txt');
```

Example 2: Opening the file myfiles/example.txt from a shared directory on a remote computer.

```plaintext
fileID := F_AOPEN('/mount/shared/myfiles/example.txt');
```

See Also:

- "File Name Warning & Limitations" (p. 251)
4.11.2.2  SD Card Path Conventions

To access the SD card memory a valid SD card label must be used at the beginning of the path, followed by the relative path to the SD card. *(Valid SD Card Label)/(Relative Path)*

A valid SD card relative path starts with //, /, \, or \. This is immediately followed by SDCard which is followed by \ or /. Please note that this path label is case insensitive.

<table>
<thead>
<tr>
<th>Valid Paths</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>//SDCard/file1</td>
<td></td>
</tr>
<tr>
<td>\Sdcard/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>

<table>
<thead>
<tr>
<th>Invalid Paths</th>
<th>Reason for being invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>///SDCard/file1</td>
<td>Started with more than two forward or 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>

In order to maintain compatibility with the Simulator, the SDCard folder is created inside the userdata folder. File access points to userdata/SDCard when a PDMM SDCard path is used on the Simulator.

See Also:

- "File Name Warning & Limitations" (p. 251)
4.11.2.3 File Name Warning & Limitations

File names in the AKD PDMM or PCMM flash storage are case-sensitive and the SD card (FAT16 or FAT32) are not case-sensitive.

<table>
<thead>
<tr>
<th>Storage</th>
<th>File System</th>
<th>Case-Sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKD PDMM / PCMM embedded flash</td>
<td>FFS3 (POSIX-like)</td>
<td>Yes</td>
</tr>
<tr>
<td>AKD PDMM / PCMM SD card</td>
<td>FAT16 or FAT32</td>
<td>No</td>
</tr>
</tbody>
</table>

For example, two files (MyFile.txt and myfile.txt) can exist in the same directory of the PDMM flash, but cannot exist in the same directory on the PDMM’s SD card. If you copy two files (via backup operation or function) with the same name, but different upper/lower case letters, from the PDMM flash to the SD card, one of the files will be lost. To prevent conflicts and to keep your application compatible across all platforms, use unique filenames and do not rely on case-sensitive filenames.
### 4.11.3 File Management Functions

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;F_AOPEN&quot; (p. 252)</td>
<td>Create or open a file in append mode</td>
</tr>
<tr>
<td>&quot;F_CLOSE&quot; (p. 252)</td>
<td>Close an open file</td>
</tr>
<tr>
<td>&quot;F_COPY&quot; (p. 252)</td>
<td>Copy a file</td>
</tr>
<tr>
<td>&quot;F_DELETE&quot; (p. 253)</td>
<td>Remove a file</td>
</tr>
<tr>
<td>&quot;F_EOF&quot; (p. 253)</td>
<td>Test if the end of the file is reached in a file that is open for reading</td>
</tr>
<tr>
<td>&quot;F_EXIST&quot; (p. 253)</td>
<td>Test if a file exists</td>
</tr>
<tr>
<td>F_GETSIZE (⇒ p. 253)</td>
<td>Get the size of a file</td>
</tr>
<tr>
<td>&quot;F_RENAME&quot; (p. 253)</td>
<td>Rename a file</td>
</tr>
<tr>
<td>&quot;F_ROPEN&quot; (p. 254)</td>
<td>Open a file for reading</td>
</tr>
<tr>
<td>&quot;F_SEEK&quot; (p. 254)</td>
<td>Set the current position of a file</td>
</tr>
<tr>
<td>&quot;F_WOPEN&quot; (p. 254)</td>
<td>Create or reset a file and open it for writing</td>
</tr>
<tr>
<td>&quot;FA_READ&quot; (p. 255)</td>
<td>Read a DINT integer from a binary file</td>
</tr>
<tr>
<td>&quot;FA_WRITE&quot; (p. 255)</td>
<td>Write a DINT integer to a binary file</td>
</tr>
<tr>
<td>&quot;FB_READ&quot; (p. 255)</td>
<td>Read binary data from a file</td>
</tr>
<tr>
<td>&quot;FB_WRITE&quot; (p. 255)</td>
<td>Write binary data to a file</td>
</tr>
<tr>
<td>&quot;FM_READ&quot; (p. 256)</td>
<td>Read a string value from a text file</td>
</tr>
<tr>
<td>&quot;FM_WRITE&quot; (p. 256)</td>
<td>Write a string value to a text file</td>
</tr>
<tr>
<td>&quot;SD_ISREADY&quot; (p. 257)</td>
<td>Check that the SD card is ready for read/write</td>
</tr>
<tr>
<td>&quot;SD_MOUNT&quot; (p. 257)</td>
<td>Mount an SD card</td>
</tr>
<tr>
<td>&quot;SD_UNMOUNT&quot; (p. 257)</td>
<td>Unmount an SD card</td>
</tr>
</tbody>
</table>

#### 4.11.3.1 F_AOPEN

Open a file in "append" mode

ID := F_AOPEN (PATH);

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

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

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

#### 4.11.3.2 F_CLOSE

Close an open file

OK := F_CLOSE (ID);

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

#### 4.11.3.3 F_COPY

Copy a file

ID := F_COPY (ID);

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

OK := F_COPY (SRC, DST);

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

4.11.3.4 F_DELETE
Remove a file

OK := F_DELETE (PATH);

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

4.11.3.5 F_EOF
Test if the end of a file is encountered

OK := F_EOF (ID);

| ID : DINT   | ID of the open file |
| OK : BOOL   | TRUE if the end of the file has been encountered |

F_EOF must be used only for files open in read mode by the F_ROPEN function.

4.11.3.6 F_EXIST
Test if file exists

OK := F_EXIST (PATH);

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

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

SIZE := F_GETSIZE (PATH);

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

4.11.3.8 F_RENAME
Rename a file

OK := F_RENAME (PATH, NEWNAME);
PATH : STRING  Name of the file (must exist). Can include a path name according to target system conventions.

NEWNAME : STRING  New name for the file

OK : BOOL  TRUE if successful

#### 4.11.3.9 F_ROPEN

Open a file for reading

#### 4.11.3.10.1 Example

##### 4.11.3.11.2.1 Structured Text

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

##### 4.11.3.12.3.2 Ladder Diagram

![Ladder Diagram](image)

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

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

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

#### 4.11.3.13 F_SEEK

Set the current position in an open file

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

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

#### 4.11.3.14 F_WOPEN

Open a file for writing

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

**NOTE**
The positive transition on each file operation FB prevents to open the file every time the program runs (each cycle).
ID := F_WOPEN (PATH);

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

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

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

### 4.11.3.15 FA_READ

Read a DINT value from a file

Q := FA_READ (ID);

| ID: DINT | ID of a file open for reading |
| Q: DINT  | read value or 0 in case of error |

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

### 4.11.3.16 FA_WRITE

Write a DINT value to a file

OK := FA_WRITE (ID, IN);

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

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

### 4.11.3.17 FB_READ

Read binary data from a file

OK := FB_READ (ID, V);

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

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

### 4.11.3.18 FB_WRITE

Write binary data to a file

OK := FB_WRITE (ID, V);

| ID: DINT | ID of a file open for writing |
Variables are stored in binary format in the file, using memory conventions of the target system.

### 4.11.3.19 FM_READ

Read a string value from a file

\[ Q := \text{FM} \_ \text{READ} (ID); \]

<table>
<thead>
<tr>
<th>ID</th>
<th>DINT</th>
<th>ID of a file open for reading</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>STRING</td>
<td>read value or empty string in case of error</td>
</tr>
</tbody>
</table>

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

### 4.11.3.20 FM_WRITE

Write a string value to a file

\[ \text{OK} := \text{FM} \_ \text{WRITE} (ID, IN); \]

<table>
<thead>
<tr>
<th>ID</th>
<th>DINT</th>
<th>ID of a file open for writing</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>STRING</td>
<td>string value to be written</td>
</tr>
<tr>
<td>OK</td>
<td>BOOL</td>
<td>return check; TRUE if successful</td>
</tr>
</tbody>
</table>

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

### 4.11.3.21.1 String Escape Sequences

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

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$</td>
<td>$</td>
</tr>
<tr>
<td>$'</td>
<td>'</td>
</tr>
<tr>
<td>$L</td>
<td>linefeed</td>
</tr>
<tr>
<td>$N</td>
<td>newline</td>
</tr>
<tr>
<td>$P</td>
<td>page (form feed)</td>
</tr>
<tr>
<td>$R</td>
<td>return</td>
</tr>
<tr>
<td>$T</td>
<td>tab</td>
</tr>
<tr>
<td>$xx</td>
<td>hex value</td>
</tr>
</tbody>
</table>

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

```plaintext
ID:=F_WOPEN('c:\ myfile.txt');
WOK:=FM_WRITE(ID,'123456$N');
//WOK:=FM_WRITE(ID,'$N');
WOK:=FM_WRITE(ID,'abcd$N');
WOK:=FM_WRITE(ID,'the end');
WOK:=F_Close(ID);
```
The example outputs a file which reads:

```
123456
abcd
the end
```

### 4.11.3.22 SD_MOUNT

Mount the SDCard on the PDMM. This will not perform any action, and always return TRUE with a Simulator.

```
OK := SD_MOUNT();
```

**OK : BOOL**  TRUE if mounting SDCard is successful

**NOTE**

Before performing, make sure the SDCard is inserted.

**TIP**

It is recommended that SD_MOUNT be used only when motion is not started.

### 4.11.3.23 SD_UNMOUNT

Un-mount the SDCard from the PDMM. This will not perform any action, and always return TRUE with a Simulator.

```
OK := SD_UNMOUNT();
```

**OK : BOOL**  TRUE if un-mounting SDCard is successful

**TIP**

It is recommended that SD_UNMOUNT be used only when motion is not started.

### 4.11.3.24 SD_ISREADY

Verify if the SDCard is mounted on the PDMM. This will verify if the SDCard folder is available inside the userdata folder when using a Simulator.

```
OK := SD_ISREADY();
```

**OK : BOOL**  TRUE if the SDCard is mounted (PDMM / PCMM)
4.11.4 File Management Function Examples

Following are several examples of how File Management functions may be used. The functions used include "F_AOPEN" (p. 252), "F_CLOSE" (p. 252), "F_WOPEN" (p. 254), and "FM_WRITE" (p. 256).

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

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

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

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

*Function* - Returns system information.

### 4.12.1 Inputs
INFO : DINT  Identifier of the requested information

### 4.12.2 Outputs
Q : DINT  Value of the requested information or 0 if error

### 4.12.3 Remarks
The INFO parameter can be one of the following predefined values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<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_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_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_CYCLESTAMP_MS</td>
<td>time stamp of the current cycle in milliseconds (OEM dependent)</td>
</tr>
<tr>
<td>_SYSINFO_CYCLEOVERFLOWS</td>
<td>number of detected cycle time overflows</td>
</tr>
<tr>
<td>_SYSINFO_CYCLECOUNT</td>
<td>counter of cycles</td>
</tr>
<tr>
<td>_SYSINFO_APPSTAMP</td>
<td>compiling date stamp of the application</td>
</tr>
<tr>
<td>_SYSINFO_CODECRC</td>
<td>CRC of the application code.</td>
</tr>
<tr>
<td>_SYSINFO_DATACRC</td>
<td>CRC of the application symbols.</td>
</tr>
<tr>
<td>_SYSINFO_FREEHEAP</td>
<td>Available space in memory heap (bytes)</td>
</tr>
<tr>
<td>_SYSINFO_DBSIZE</td>
<td>Space used in RAM (bytes)</td>
</tr>
<tr>
<td>_SYSINFO_ELAPSED</td>
<td>Seconds elapsed since startup</td>
</tr>
<tr>
<td>_SYSINFO_CHANGE_CYCLE</td>
<td>Indicates a cycle just after an On Line Change</td>
</tr>
<tr>
<td>_SYSINFO_WARMSTART</td>
<td>Non zero if RETAIN variables were loaded at the last start</td>
</tr>
<tr>
<td>_SYSINFO_NBLOCKED</td>
<td>Number of locked variables</td>
</tr>
<tr>
<td>_SYSINFO_NBBREAKPOINTS</td>
<td>Number of installed breakpoints</td>
</tr>
<tr>
<td>_SYSINFO_BIGENDIAN</td>
<td>Non zero if the runtime processor is big endian</td>
</tr>
<tr>
<td>_SYSINFO_DEMOAPP</td>
<td>Non zero if the application was compiled in DEMO mode</td>
</tr>
</tbody>
</table>

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

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

### 4.12.4 ST Language

Q := GETSYSINFO (INFO);

### 4.12.5 FBD Language
4.12.6 FFLD Language

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

4.12.7 IL Language:

Op1: LD INFO
     GETSYSINFO
     ST Q
4.13 HYSTER

*Function Block* - Hysteresis detection.

### 4.13.1 Inputs

- **XIN1**: REAL  
  First input signal
- **XIN2**: REAL  
  Second input signal
- **EPS**: REAL  
  Hysteresis

### 4.13.2 Outputs

- **Q**: BOOL  
  Detected hysteresis: TRUE if XIN1 becomes greater than XIN2+EPS and is not yet below XIN2-EPS

### 4.13.3 Remarks

The hysteresis is detected on the difference of XIN1 and XIN2 signals. In FFLD language, the input rung (EN) is used for enabling the block. The output rung is the Q output.

### 4.13.4 ST Language

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

```plaintext
MyHyst(XIN1, XIN2, EPS);
Q := MyHyst.Q;
```

### 4.13.5 FBD Language

![FBD Diagram](image)

### 4.13.6 FFLD Language

(* The block is not called if EN is FALSE *)

```plaintext
EN  Q  [
  /)
  [
  Hyster)
  XIN1
  XIN2
  EPS
```

### 4.13.7 IL Language:

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

```plaintext
Op1: CAL MyHyst (XIN1, XIN2, EPS)
FFLD MyHyst.Q
ST Q
```

See also

- AVERAGE
- INTEGRAL
- DERIVATE
- LIM_ALRM
- STACKINT
4.14 INTEGRAL

*Function Block* - Calculates the integral of a signal with respect to time.

The time unit is seconds. The output signal has the units of the input signal multiplied by seconds. The INTEGRAL block samples the input signal at a maximum rate of 1 millisecond.

### 4.14.1 Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>BOOL</td>
<td>Run command: TRUE=integrate / FALSE=hold</td>
</tr>
<tr>
<td>R1</td>
<td>BOOL</td>
<td>Overriding reset</td>
</tr>
<tr>
<td>XIN</td>
<td>REAL</td>
<td>Input signal</td>
</tr>
<tr>
<td>X0</td>
<td>REAL</td>
<td>Initial value</td>
</tr>
<tr>
<td>CYCLE</td>
<td>TIME</td>
<td>Sampling period (must not be less than the target cycle timing)</td>
</tr>
</tbody>
</table>

### 4.14.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>DINT</td>
<td>Running mode report: NOT (R1)</td>
</tr>
<tr>
<td>XOUT</td>
<td>REAL</td>
<td>Output signal</td>
</tr>
</tbody>
</table>

### 4.14.3 Remarks

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

### 4.14.4 ST Language

```st
(* MyIntg is a declared instance of INTEGRAL function block *)
MyIntg (RUN, R1, XIN, X0, CYCLE);
Q := MyIntg.Q;
XOUT := MyIntg.XOUT;
```

### 4.14.5 FBD Language

![INTEGRAL FBD Diagram]

### 4.14.6 FFLD Language

![INTEGRAL FFLD Diagram]

### 4.14.7 IL Language:
(* MyIntg is a declared instance of INTEGRAL function block *)
Op1: CAL MyIntg (RUN, R1, XIN, X0, CYCLE)
    FFLD  MyIntg.Q
    ST   Q
    FFLD  MyIntg.XOUT
    ST   XOUT

See also
AVERAGE  DERIVATE  LIM_ALRM  HYSTER  STACKINT
4.15 LIFO

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

4.15.1 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>

4.15.2 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>

4.15.3 Remarks

NEXTIN, NEXTOUT and BUFFER must have the same data type and cannot be STRING. The NEXTOUT argument specifies a variable which is filled with the value at the top of the stack after the block is called.

Values are stored in the "BUFFER" array. Data is never shifted or reset. Only read and write pointers and pushed values are updated. The maximum size of the stack is the dimension of the array.

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

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

4.15.4 ST Language

```plaintext
(* MyLIFO is a declared instance of LIFO function block *)
MyLIFO (PUSH, POP, RST, NEXTIN, NEXTOUT, BUFFER);
EMPTY := MyLIFO.EMPTY;
OFLO := MyLIFO.OFLO;
COUNT := MyLIFO.COUNT;
PREAD := MyLIFO.PREAD;
PWRITE := MyLIFO.PWRITE;
```
4.15.5 FBD Language

```
  LIFO
  PUSH  EMPTY
  POP   OFLO
  RST   COUNT
  NEXTIN PREAD
  NEXTOUT PWRITE
  BUFFER
```

4.15.6 FFLD Language

```
  PUSH ] [ LIFO EMPTY ( ) [ ]
  POP   OFLO
  RST   COUNT
  NEXTIN PREAD
  NEXTOUT PWRITE
  BUFFER
```

4.15.7 IL Language

```
(* MyLIFO is a declared instance of LIFO function block *)
Op1: CAL MyLIFO (PUSH, POP, RST, NEXTIN, NEXTOUT, BUFFER)
FFLD MyLIFO.EMPTY
ST EMPTY
FFLD MyLIFO.OFLO
ST OFLO
FFLD MyLIFO.COUNT
ST COUNT
FFLD MyLIFO.PREAD
ST PREAD
FFLD MyLIFO.PWRITE
ST PWRITE
```

See also

FIFO
4.16 LIM_ALRM

Function Block - Detects High and Low limits of a signal with hysteresis.

4.16.1 Inputs
H : REAL Value of the High limit
X : REAL Input signal
L : REAL Value of the Low limit
EPS : REAL Value of the hysteresis

4.16.2 Outputs
QH : BOOL TRUE if the signal exceeds the High limit
Q : BOOL TRUE if the signal exceeds one of the limits (equals to QH OR QL)
QL : BOOL TRUE if the signal exceeds the Low limit

4.16.3 Remarks
In FFLD language, the input rung (EN) is used for enabling the block. The output rung is the QH output.

4.16.4 ST Language

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

4.16.5 FBD Language

```
LIM_ALRM
<table>
<thead>
<tr>
<th>H</th>
<th>X</th>
<th>L</th>
<th>EPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH</td>
<td>Q</td>
<td>QL</td>
<td></td>
</tr>
</tbody>
</table>
```

4.16.6 FFLD Language

```
(* The block is not called if EN is FALSE *)
EN [ H X L EPS ] LIM_ALRM { } QH
```

4.16.7 IL Language:
(* MyAlarm is a declared instance of LIM_ALRM function block *)

Op1: CAL MyAlarm (H, X, L, EPS)
    FFLD MyAlarm.QH
    ST QH
    FFLD MyAlarm.Q
    ST Q
    FFLD MyAlarm.QL
    ST QL

See also

ALARM_A  ALARM_M
4.17 LogFileCSV

Function block - Generate a log file in CSV format for a list of variables

4.17.1 Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG</td>
<td>BOOL</td>
</tr>
<tr>
<td>RST</td>
<td>BOOL</td>
</tr>
<tr>
<td>LIST</td>
<td>DINT</td>
</tr>
<tr>
<td>PATH</td>
<td>STRING</td>
</tr>
</tbody>
</table>

4.17.2 Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>BOOL</td>
</tr>
<tr>
<td>ERR</td>
<td>DINT</td>
</tr>
</tbody>
</table>

**IMPORTANT**
Calling this function can lead to missing several PLC cycles. Files are opened and closed directly by the target’s Operating System. Opening some files may be dangerous for system safety and integrity. The number of open files may be limited by the target system.

**NOTE**
- Opening a file may be unsuccessful (invalid path or file name, too many open files...) Your application has to process such error cases in a safe way.
- File management may be not available on some targets. Please refer to OEM instructions for further details about available features.
- Valid paths for storing files depend on the target implementation. Please refer to OEM instructions for further details about available paths.

4.17.3 Remarks

This function enables to log values of a list of variables in a CSV file. On each rising edge of the LOG input, one more line of values is added to the file. There is one column for each variable, as they are defined in the list.

The list of variables is prepared using the KAS IDE or a text editor. Use the VLID function to get the identifier of the list.

On a rising edge of the RST command, the file is emptied.

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

In case of error, a report is given in the ERR output. Possible error values are:

1 = Cannot reset file on a RST command
2 = Cannot open file for data storing on a LOG command
3 = Embedded lists are not supported by the runtime
4 = Invalid list ID
5 = Error while writing to file

Combined with real time clock management functions, this block provides a very easy way to generate a periodical log file. The following example shows a list and a program that log values everyday at 14h23m (2:23 pm) (see call out 1)
4.17.4 ST Language

(* MyLOG is a declared instance ofLogFileCSV function block *)
MyLOG (b_LOG, RST, LIST, PATH);
Q := MyLOG.Q;
ERR := MyLog.ERR;

4.17.5 FBD Language

4.17.6 FFLD Language
4.17.7 IL Language

(* MyLOG is a declared instance of LogFileCSV function block *)
Op1: CAL MyLOG (b_LOG, RST, LIST, PATH);
FFLD MyLOG.Q
ST Q
FFLD MyLog.ERR
ST ERR

See also
VLID
4.18 PID

*Function Block - PID loop*

### 4.18.1 Inputs

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

### 4.18.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xout</td>
<td>REAL</td>
<td>Output command value.</td>
</tr>
<tr>
<td>ER</td>
<td>REAL</td>
<td>Last calculated error.</td>
</tr>
<tr>
<td>Xout_P</td>
<td>REAL</td>
<td>Last calculated proportional value.</td>
</tr>
<tr>
<td>Xout_I</td>
<td>REAL</td>
<td>Last calculated integrated value.</td>
</tr>
<tr>
<td>Xout_D</td>
<td>REAL</td>
<td>Last calculated derivated value.</td>
</tr>
<tr>
<td>Xout_HLM</td>
<td>BOOL</td>
<td>TRUE if the output value is saturated to XMIN.</td>
</tr>
<tr>
<td>Xout_LLM</td>
<td>BOOL</td>
<td>TRUE if the output value is saturated to XMAX.</td>
</tr>
</tbody>
</table>
4.18.3 Diagram

4.18.4 Remarks

- It is important for the stability of the control that the TS sampling period is much bigger than the cycle time.
- Output of the PID block always starts with zero. The value will vary upon the inputs provided upon further cycle executions.
- The output rung has the same value as the AUTO input, corresponding to the input rung, in the FFLD language.

4.18.5 ST Language

(* MyPID is a declared instance of PID function block *)

MyPID (AUTO, PV, SP, XOUT_MANU, KP, TI, TD, TS, XMIN, XMAX, I_SEL, I_ITL_ON, I_ITLVAL, DEADB_ERR, FFD);
XOUT := MyPID.XOUT;
ER := MyPID.ER;
XOUT_P := MyPID.XOUT_P;
XOUT_I := MyPID.XOUT_I;
XOUT_D := MyPID.XOUT_D;
XOUT_HLM := MyPID.XOUT_HLM;
XOUT_LLM := MyPID.XOUT_LLM;

4.18.6 FBD Language
4.18.7 FFLD Language

(* ENO has the same state as the input rung *)

4.18.8 IL Language

(* MyPID is a declared instance of PID function block *)
Op1: CAL MyPID (AUTO, PV, SP, XOUT_MANU, KP, TI, TD, TS, XMIN, XMAX, I_SEL, I_ITL_ON, I_ITLVAL, DEADB_ERR, FFD)
      FFLD MyPID.XOUT
4.19  **PWM**  

Function block - generate a PWM signal.

### 4.19.1 Inputs

<table>
<thead>
<tr>
<th>XIN</th>
<th>REAL</th>
<th>Input analog value</th>
</tr>
</thead>
<tbody>
<tr>
<td>XinMin</td>
<td>REAL</td>
<td>Minimum input value</td>
</tr>
<tr>
<td>XinMax</td>
<td>REAL</td>
<td>Maximum input value</td>
</tr>
<tr>
<td>MinPulse</td>
<td>TIME</td>
<td>Minimum pulse time on output</td>
</tr>
<tr>
<td>Period</td>
<td>TIME</td>
<td>Period of the output signal</td>
</tr>
</tbody>
</table>

### 4.19.2 Outputs

| Q   | BOOL | Blinking PWM signal |

### 4.19.3 Remarks

The input value is truncated to \([\text{XinMin} .. \text{XinMax}]\) interval. XinMax must be greater than XinMin.

The signal is TRUE during:

\[
\frac{(Xin - XinMin) \times \text{Period}}{XinMax - XinMin}
\]

### 4.19.4 ST Language

PWM1 is a declared instance of PWM function block.

```plaintext
PWM1 (rIn, rInMin, rInMax, tMinPulse, tPeriod);
Signal := PWM1.Q;
```

### 4.19.5 Example
### 4.20 RAMP

*Function block* - Limit the ascendance or descendance of a signal

#### 4.20.1 Inputs

- **IN**: REAL Input signal
- **ASC**: REAL Maximum ascendance during time base
- **DSC**: REAL Maximum descendance during time base
- **TM**: TIME Time base
- **RST**: BOOL Reset

#### 4.20.2 Outputs

- **OUT**: REAL Ramp signal

#### 4.20.3 Time diagram

![Time diagram of RAMP function block](image)

#### 4.20.4 Remarks

Parameters are not updated constantly. They are taken into account when only:
- the first time the block is called
- when the reset input (RST) is TRUE

In these two situations, the output is set to the value of IN input.

ASC and DSC give the maximum ascendant and descendant growth during the TB time base. Both must be expressed as positive numbers.

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

#### 4.20.5 ST Language

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

```
MyRamp(IN, ASC, DSC, TM, RST);
OUT := MyBlinker.OUT;
```

#### 4.20.6 FBD Language
4.20.7 FFLD Language

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

(* ENO keeps the same value as EN *)

4.20.8 IL Language

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

Op1: CAL MyRamp (IN, ASC, DSC, TM, RST)

FFLD MyBlinker.OUT

ST OUT
### 4.21 Real Time Clock Management Functions

The following functions read the real time clock of the target system:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTCurDate</td>
<td>Get current date stamp</td>
</tr>
<tr>
<td>DTCurTime</td>
<td>Get current time stamp</td>
</tr>
<tr>
<td>DTDay</td>
<td>Get day from date stamp</td>
</tr>
<tr>
<td>DTMonth</td>
<td>Get month from date stamp</td>
</tr>
<tr>
<td>DTYear</td>
<td>Get year from date stamp</td>
</tr>
<tr>
<td>DTSec</td>
<td>Get seconds from time stamp</td>
</tr>
<tr>
<td>DTMin</td>
<td>Get minutes from time stamp</td>
</tr>
<tr>
<td>DTHour</td>
<td>Get hours from time stamp</td>
</tr>
<tr>
<td>DTMs</td>
<td>Get milliseconds from time stamp</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAY_TIME</td>
<td>With predefined format</td>
</tr>
<tr>
<td>DTFORMAT</td>
<td>With custom format</td>
</tr>
</tbody>
</table>

The following functions are used for triggering operations:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTAt</td>
<td>Pulse signal at the given date/time</td>
</tr>
<tr>
<td>DTEvery</td>
<td>Pulse signal with long period</td>
</tr>
</tbody>
</table>

**Important**

The real-time clock may not be available on all controller hardware models. Please consult the controller hardware specifications for real-time clock availability.

**DAY_TIME**: get current date or time

```plaintext
Q := DAY_TIME (SEL);
SEL : DINT specifies the wished information (see below)
Q : STRING wished information formatted on a string
```

Possible values of SEL input

1. current time - format: 'HH:MM:SS'
2. day of the week
3. (default) current date - format: 'YYYY/MM/DD'

**DTCURDATE**: get current date stamp

```plaintext
Q := DTCurDate ();
Q : DINT numerical stamp representing the current date
```

**DTCURTIME**: get current time stamp

```plaintext
Q := DTCurTime ();
Q : DINT numerical stamp representing the current time of the day
```

**DTYEAR**: extract the year from a date stamp

```plaintext
Q := DTYear (iDate);
IDATE : DINT numerical stamp representing a date. This is output of DTCURDATE.
Q : DINT year of the date (ex: 2004)
```

**DTMONTH**: extract the month from a date stamp

```plaintext
Q := DTMonth (iDate);
```
IDATE : DINT  numerical stamp representing a date. This is output of DTCURDATE.
Q  : DINT  month of the date (1..12)

**DTDAY:** extract the day of the month from a date stamp
Q := DTDay (iDate);

IDATE : DINT  numerical stamp representing a date. This is output of DTCURDATE.
Q  : DINT  day of the month of the date (1..31)

**DTHOUR:** extract the hours from a time stamp
Q := DHour (iTime);

ITIME : DINT  numerical stamp representing a time. This is output of DTCURDATE.
Q  : DINT  Hours of the time (0..23)

**DTMIN:** extract the minutes from a time stamp
Q := DMin (iTime);

ITIME : DINT  numerical stamp representing a time. This is output of DTCURDATE.
Q  : DINT  Minutes of the time (0..59)

**DTSEC:** extract the seconds from a time stamp
Q := DTSec (iTime);

ITIME : DINT  numerical stamp representing a time. This is output of DTCURDATE.
Q  : DINT  Seconds of the time (0..59)

**DTMS:** extract the milliseconds from a time stamp
Q := DTMs (iTime);

ITIME : DINT  numerical stamp representing a time. This is output of DTCURDATE.
Q  : DINT  Milliseconds of the time (0..999)
4.21.1 DAY_TIME

Function - Format the current date/time to a string.

4.21.1.1 Inputs
SEL : DINT Format selector

4.21.1.2 Outputs
Q : STRING String containing formatted date or time

**IMPORTANT**
The real-time clock may not be available on all controller hardware models. Please consult the controller hardware specifications for real-time clock availability.

4.21.1.3 Remarks
Possible values of the SEL input are:

1. current time - format: 'HH:MM:SS'
2. day of the week
0 (default) current date - format: 'YYYY/MM/DD'

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

4.21.1.4 ST Language
Q := DAY_TIME (SEL);

4.21.1.5 FBD Language

```
SEL
```

4.21.1.6 FFLD Language

```
EN [ SEL ]
```

4.21.1.7 IL Language
Op1: LD SEL
DAY_TIME
ST Q

See also
DTFORMAT
4.21.2 DTFORMAT

Function - Format the current date/time to a string with a custom format.

4.21.2.1 Inputs
FMT: STRING  Format string

4.21.2.2 Outputs
Q : STRING  String containing formatted date or time

⚠️ IMPORTANT
The real-time clock may not be available on all controller hardware models. Please consult the controller hardware specifications for real-time clock availability.

4.21.2.3 Remarks
The format string may contain any character. Some special markers beginning with the '% character indicates a date/time information:

- %Y Year including century (e.g. 2006)
- %y Year without century (e.g. 06)
- %m Month (1..12)
- %d Day of the month (1..31)
- %H Hours (0..23)
- %M Minutes (0..59)
- %S Seconds (0..59)

Example
(* let's say we are at July 04th 2006, 18:45:20 *)

Q := DTFORMAT ('Today is %Y/%m/%d - %H:%M:%S');
(* Q is 'Today is 2006/07/04 - 18:45:20 *)

4.21.2.4 ST Language

Q := DTFORMAT (FMT);

4.21.2.5 FBD Language

```
FMT
```

4.21.2.6 FFLD Language

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

```
EN  DTFORMAT  ENO
FMT  Q
```
4.21.2.7 IL Language

Op1: LD FMT
DFORMAT
ST Q

See also
DAY_TIME
4.21.3 DTAT

*Function Block* - Generate a pulse at given date and time

### 4.21.3.1 Inputs

- **YEAR** : DINT  
  Wished year (e.g. 2006)
- **MONTH** : DINT  
  Wished month (1 = January)
- **DAY** : DINT  
  Wished day (1 to 31)
- **TMOFDAY** : TIME  
  Wished time
- **RST** : BOOL  
  Reset command

### 4.21.3.2 Outputs

- **QAT** : BOOL  
  Pulse signal
- **QPAST** : BOOL  
  True if elapsed

#### Important

The real-time clock may not be available on all controller hardware models. Please consult the controller hardware specifications for real-time clock availability.

### 4.21.3.3 Remarks

Parameters are not updated constantly. They are taken into account when only:

- the first time the block is called
- when the reset input (RST) is TRUE

In these two situations, the outputs are reset to FALSE.

The first time the block is called with RST=FALSE and the specified date/stamp is passed, the output QPAST is set to TRUE, and the output QAT is set to TRUE for one cycle only (pulse signal).

Highest units are ignored if set to 0. For instance, if arguments are "year=0, month=0, day = 3, tmofday=#10h" then the block will trigger on the next 3rd day of the month at 10h.

In FFLD language, the block is activated only if the input rung is TRUE.

### 4.21.3.4 ST Language

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

```plaintext
MyDTAT (YEAR, MONTH, DAY, TMOFDAY, RST);
QAT := MyDTAT.QAT;
QPAST := MyDTAT.QPAST;
```

### 4.21.3.5 FBD Language

![FBD Diagram](image)
4.21.3.6 FFLD Language

(* Called only if EN is TRUE *)

\[
\begin{array}{c|c|c}
\text{EN} & \text{DTAT} & \text{Qat} \\
\hline
\text{YEAR} & \text{QPast} \\
\text{MONTH} & \\
\text{DAY} & \\
\text{TMOFDAY} & \\
\text{RST} & \\
\end{array}
\]

4.21.3.7 IL Language:

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

Op1: CAL MyDTAT (YEAR, MONTH, DAY, TMOFDAY, RST)
FFLD MyDTAT.QAT
ST QAT
FFLD MyDTAT.A.QPAST
ST QPAST

See also

DTEVERY  Real time clock functions
4.21.4 DTEVERY

*Function Block* - Generate a pulse signal with long period

4.21.4.1 Inputs

<table>
<thead>
<tr>
<th>RUN</th>
<th>DINT</th>
<th>Enabling command</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAYS</td>
<td>DINT</td>
<td>Period: number of days</td>
</tr>
<tr>
<td>TM</td>
<td>TIME</td>
<td>Rest of the period (if not a multiple of 24h)</td>
</tr>
</tbody>
</table>

4.21.4.2 Outputs

| Q       | BOOL  | Pulse signal |

4.21.4.3 Remarks

This block provides a pulse signal with a period of more than 24h. The period is expressed as:

\[ \text{DAYS} \times 24\text{h} + \text{TM} \]

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

4.21.4.4 ST Language

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

\[
\text{MyDTEVERY (RUN DAYS, TM);}
\]

\[
\text{Q := MyDTEVERY.Q;}
\]

4.21.4.5 FBD Language

![FBD Diagram]

4.21.4.6 FFLD Language

![FFLD Diagram]

4.21.4.7 IL Language:

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

Opt1: CAL MyDTEVERY (RUN DAYS, TM)

    FFLD MyDTEVERY.Q
    ST Q

See also

DTAT  Real time clock functions
4.22 SerializeIn

4.22.1 Description
Extract the value of a variable from a binary frame. This function is commonly used for extracting data from a communication frame in binary format.

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

_tip_
This function is not available in IL language.

The FRAME input must fit the input position and data size. If the value cannot be safely extracted, the function returns 0. The DATA input must be directly connected to a variable, and cannot be a constant or complex expression. This variable will be forced with the extracted value.

The function extracts the following number of bytes from the source frame:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BOOL, SINT, USINT and BYTE variables</td>
</tr>
<tr>
<td>2</td>
<td>INT, UINT and WORD variables</td>
</tr>
<tr>
<td>4</td>
<td>DINT, UDINT, DWORD and REAL variables</td>
</tr>
<tr>
<td>8</td>
<td>LINT and LREAL variables</td>
</tr>
</tbody>
</table>

_important_
The function cannot be used to serialize STRING variables.

The function returns the position in the source frame, after the extracted data. Thus the return value can be used as a position for the next serialization.

4.22.2 Arguments

4.22.2.1 Input

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

- **any except STRING**

<table>
<thead>
<tr>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>n/a</td>
<td>—</td>
</tr>
</tbody>
</table>

#### Pos

**Description**: Position in the source buffer

<table>
<thead>
<tr>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>DINT</td>
<td>[0,+65535]</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

#### BigEndian

**Description**: TRUE if the frame is encoded with Big Endian format.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Range</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

### 4.22.2.2 Output

#### OK

**Description**: Returns true when the function successfully executes. See [Function - General rules](#).

<table>
<thead>
<tr>
<th>Data type</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>n/a</td>
<td>?</td>
</tr>
</tbody>
</table>

#### NextPos

**Description**: Position in the source buffer after the extracted data. 0 in case of error (invalid position / buffer size).

<table>
<thead>
<tr>
<th>Data type</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>DINT</td>
<td>n/a</td>
</tr>
</tbody>
</table>

### 4.22.3 Examples

#### 4.22.3.1 Structured Text

```plaintext
NextPos := SerializeIn(Frame[] (*USINT*), @Data (*ANY*), Pos (*DINT*), BigEndian (*BOOL*)); //Read the position
```
4.23 SerializeOut

4.23.1 Description
This function copies the value of a variable to a binary frame. This function is commonly used for building a communication frame in binary format.

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

**NOTE**
This function is not available in IL language.

The FRAME input must be an array large enough to receive the data. If the data cannot be safely copied to the destination buffer, the function returns 0.

The function copies the following number of bytes to the destination frame:

| 1 byte | BOOL, SINT, USINT and BYTE variables |
| 2 bytes | INT, UINT and WORD variables |
| 4 bytes | DINT, UDINT, DWORD and REAL variables |
| 8 bytes | LINT and LREAL variables |

**IMPORTANT**
The function cannot be used to serialize STRING variables.

The function returns the position in the destination frame, after the copied data. Thus the return value can be used as a position for the next serialization.

4.23.2 Arguments

4.23.2.1 Input

<table>
<thead>
<tr>
<th>En</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>En</td>
<td>Execute the function</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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frame[]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame[]</td>
<td>Destination 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>_</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Source variable to be copied</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Data type</td>
<td>any except STRING</td>
</tr>
<tr>
<td>Range</td>
<td>—</td>
</tr>
<tr>
<td>Unit</td>
<td>n/a</td>
</tr>
<tr>
<td>Default</td>
<td>—</td>
</tr>
</tbody>
</table>

**Pos**

<table>
<thead>
<tr>
<th>Description</th>
<th>Position in the destination buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data type</td>
<td>DINT</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>—</td>
</tr>
</tbody>
</table>

**BigEndian**

<table>
<thead>
<tr>
<th>Description</th>
<th>TRUE if the frame is encoded with Big Endian format.</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

### 4.23.2.2 Output

**OK**

<table>
<thead>
<tr>
<th>Description</th>
<th>Returns true when the function successfully executes. See Function - General rules.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data type</td>
<td>BOOL</td>
</tr>
<tr>
<td>Unit</td>
<td>n/a</td>
</tr>
</tbody>
</table>

**NextPos**

<table>
<thead>
<tr>
<th>Description</th>
<th>Position in the destination buffer after the copied data. 0 in case of error (invalid position / buffer size).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data type</td>
<td>DINT</td>
</tr>
<tr>
<td>Unit</td>
<td>n/a</td>
</tr>
</tbody>
</table>

### 4.23.3 Examples

#### 4.23.3.1 Structured Text

```plaintext
NextPos := SerializeOut(Frame[](*USINT*), Data(*ANY*), Pos (*DINT*), BigEndian(*BOOL*)); //Read the position
```
4.24 SigID

*Function* - Get the identifier of a "Signal" resource

**4.24.1 Inputs**

- **SIGNAL**: STRING - Name of the signal resource - *must be a constant value!*
- **COL**: STRING - Name of the column within the signal resource - *must be a constant value!*

**4.24.2 Outputs**

- **ID**: DINT - ID of the signal - to be passed to other blocks

**4.24.3 Remarks**

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

**4.24.4 ST Language**

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

**4.24.5 FBD Language**

![FBD Diagram]

**4.24.6 FFLD Language**

![FFLD Diagram]

**4.24.7 IL Language**

Op1: LD 'MySignal'
    SigId 'FirstColumn'
    ST ID

**See also**

SigPlay  SigScale
4.25 SigPlay

Function block - Generate a signal defined in a resource

4.25.1 Inputs

- **IN**: BOOL  Triggering command
- **ID**: DINT  ID of the signal resource, provided by SigID function
- **RST**: BOOL  Reset command
- **TM**: TIME  Minimum time in between two changes of the output

4.25.2 Outputs

- **Q**: BOOL  TRUE when the signal is finished
- **OUT**: REAL  Generated signal
- **ET**: TIME  Elapsed time

4.25.3 Remarks

The "ID" argument is the identifier of the "signal" resource. Use the SigID function to get this value.

The "IN" argument is used as a "Play / Pause" command to play the signal. The signal is not reset to the beginning when IN becomes FALSE. Instead, use the "RST" input that resets the signal and forces the OUT output to 0.

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

This function block includes its own timer. Alternatively, you can use the SigScale function if you want to trigger the signal using a specific timer.

4.25.4 ST Language

```plaintext
Q := SigScale (ID, IN);
```

4.25.5 FBD Language

![FBD Diagram]

4.25.6 FFLD Language

![FFLD Diagram]

4.25.7 IL Language
Op1: FFLD IN
SigScale ID
ST Q

See also
SigScale SigID
4.26 SigScale

*Function* - Get a point from a "Signal" resource

### 4.26.1 Inputs

<table>
<thead>
<tr>
<th>ID</th>
<th>DINT</th>
<th>ID of the signal resource, provided by SigID function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>TIME</td>
<td>Time (X) coordinate of the wished point within the signal resource</td>
</tr>
</tbody>
</table>

### 4.26.2 Outputs

| Q         | REAL     | Value (Y) coordinate of the point in the signal |

### 4.26.3 Remarks

The "ID" argument is the identifier of the "signal" resource. Use the SigID function to get this value.

This function converts a time value to a analog value such as defined in the signal resource. This function can be used instead of SigPlay function block if you want to trigger the signal using a specific timer.

### 4.26.4 ST Language

Q := SigScale (ID, IN);

### 4.26.5 FBD Language

```
  ID
  IN
  Q
  SigScale
```

### 4.26.6 FFLD Language

```
  EN [ ]
  ID
  IN
  SigScale
  Q
  ENO ( )
```

### 4.26.7 IL Language

```
Op1: LD
IN
SigScale ID
ST Q
```

**See also**

SigPlay SigID
4.27 STACKINT

*Function Block* - Manages a stack of DINT integers.

### 4.27.1 Inputs

- **PUSH**: BOOL  Command: when changing from FALSE to TRUE, the value of IN is pushed on the stack
- **POP**: BOOL  Pop command: when changing from FALSE to TRUE, deletes the top of the stack
- **R1**: BOOL  Reset command: if TRUE, the stack is emptied and its size is set to N
- **IN**: DINT  Value to be pushed on a rising pulse of PUSH
- **N**: DINT  maximum stack size - cannot exceed 128

### 4.27.2 Outputs

- **EMPTY**: BOOL  TRUE if the stack is empty
- **OFLO**: BOOL  TRUE if the stack is full
- **OUT**: DINT  value at the top of the stack

### 4.27.3 Remarks

Push and pop operations are performed on rising pulse of PUSH and POP inputs. In FFLD language, the input rung is the PUSH command. The output rung is the EMPTY output.

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

### 4.27.4 ST Language

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

```
MyStack (PUSH, POP, R1, IN, N);
EMPTY := MyStack.EMPTY;
OFLO := MyStack.OFLO;
OUT := MyStack.OUT;
```

### 4.27.5 FBD Language

```
STACKINT

PUSH
POP
R1
IN
N

EMPTY
OFLO
OUT
```

### 4.27.6 FFLD Language

```
PUSH [ ]
POP
R1
IN
N

STACKINT ( )
EMPTY
OFLO
OUT
```

### 4.27.7 IL Language
(* MyStack is a declared instance of STACKINT function block *)

Op1: CAL MyStack (PUSH, POP, R1, IN, N)
  FFLD MyStack.EMPTY
  ST EMPTY
  FFLD MyStack.OFLO
  ST OFLO
  FFLD MyStack.OUT
  ST OUT

See also

AVERAGE  INTEGRAL  DERIVATE  LIM_ALRM  HYSTER
4.28 SurfLin

*Function block*- Linear interpolation on a surface.

### 4.28.1 Inputs
- **X** : REAL  X coordinate of the point to be interpolated.
- **Y** : REAL  Y coordinate of the point to be interpolated.
- **XAxis** : REAL[]  X coordinates of the known points of the X axis.
- **YAxis** : REAL[]  Y coordinates of the known points of the Y axis.
- **ZVal** : REAL[,]  Z coordinate of the points defined by the axis.

### 4.28.2 Outputs
- **Z** : REAL  Interpolated Z value corresponding to the X,Y input point
- **OK** : BOOL  TRUE if successful.
- **ERR** : DINT  Error code if failed - 0 if OK.

### 4.28.3 Remarks
This function performs linear surface interpolation in between a list of points defined in XAxis and YAxis single dimension arrays. The output Z value is an interpolation of the Z values of the four rounding points defined in the axis. Z values of defined points are passed in the ZVal matrix (two dimension array).

ZVal dimensions must be understood as: ZVal [ iX , iY ]

Values in X and Y axis must be sorted from the smallest to the biggest. There must be at least two points defined in each axis. ZVal must fit the dimension of XAxis and YAxis arrays. For instance:

- **XAxis** : ARRAY [0..2] of REAL;
- **YAxis** : ARRAY [0,3] of REAL;
- **ZVal** : ARRAY [0..2,0..3] of REAL;

In case the input point is outside the rectangle defined by XAxis and YAxis limits, the Z output is bound to the corresponding value and an error is reported.

The ERR output gives the cause of the error if the function fails:

<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>3</td>
<td>Invalid points for the Y axis</td>
</tr>
<tr>
<td>4</td>
<td>X,Y point is out of the defined axis</td>
</tr>
</tbody>
</table>
4.29 **VLID**

*Function* - Get the identifier of an embedded list of variables

### 4.29.1 Inputs

FILE : STRING  
Path name of the .TXT list file - *must be a constant value!*

### 4.29.2 Outputs

ID : DINT  
ID of the list - to be passed to other blocks

### 4.29.3 Remarks

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

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

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

![Important](image)

List files are read at compiling time and are embedded into the downloaded application code. This implies that a modification performed in the list file after downloading will not be taken into account by the application.

### 4.29.4 ST Language

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

### 4.29.5 FBD Language

![FBD Diagram](image)

### 4.29.6 FFLD Language

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

![FFLD Diagram](image)

### 4.29.7 IL Language

Opt1: LD 'MyFile.txt'
VLID COL
ST ID
About KOLLMORGEN

Kollmorgen is a leading provider of motion systems and components for machine builders. Through world-class knowledge in motion, industry-leading quality and deep expertise in linking and integrating standard and custom products, Kollmorgen delivers breakthrough solutions that are unmatched in performance, reliability and ease-of-use, giving machine builders an irrefutable marketplace advantage.

Join the Kollmorgen Developer Network for product support. Ask the community questions, search the knowledge base for answers, get downloads, and suggest improvements.

<table>
<thead>
<tr>
<th>Region</th>
<th>Address</th>
<th>Web</th>
<th>Mail</th>
<th>Tel.</th>
<th>Fax</th>
</tr>
</thead>
<tbody>
<tr>
<td>North America</td>
<td>KOLLMORGEN</td>
<td><a href="http://www.kollmorgen.com">www.kollmorgen.com</a></td>
<td><a href="mailto:support@kollmorgen.com">support@kollmorgen.com</a></td>
<td>+1 - 540 - 633 - 3545</td>
<td>+1 - 540 - 639 - 4162</td>
</tr>
<tr>
<td>Europe</td>
<td>KOLLMORGEN Europe GmbH</td>
<td><a href="http://www.kollmorgen.com">www.kollmorgen.com</a></td>
<td><a href="mailto:technik@kollmorgen.com">technik@kollmorgen.com</a></td>
<td>+49 - 2102 - 9394 - 0</td>
<td>+49 - 2102 - 9394 - 3155</td>
</tr>
<tr>
<td>South America</td>
<td>KOLLMORGEN</td>
<td><a href="http://www.kollmorgen.com">www.kollmorgen.com</a></td>
<td><a href="mailto:contato@kollmorgen.com">contato@kollmorgen.com</a></td>
<td>+55 11 4615-6300</td>
<td></td>
</tr>
<tr>
<td>China and SEA</td>
<td>KOLLMORGEN</td>
<td><a href="http://www.kollmorgen.cn">www.kollmorgen.cn</a></td>
<td><a href="mailto:sales.china@kollmorgen.com">sales.china@kollmorgen.com</a></td>
<td>+86 - 400 668 2802</td>
<td>+86 - 21 6248 5387</td>
</tr>
</tbody>
</table>