Function Blocks of REXYGEN
Reference manual

REX Controls s.r.o.

Version 2.50.10
2020-09-03
Plzeň (Pilsen), Czech Republic
Contents

1 Introduction ........................................... 13
   1.1 How to use this manual ........................... 13
   1.2 The function block description format ........... 15
   1.3 Conventions for variables, blocks and subsystems naming ........................... 16
   1.4 The signal quality corresponding with OPC ................... 17

2 EXEC – Real-time executive configuration ................. 19
   ARC – The REXYGEN system archive ..................... 20
   EXEC – Real-time executive ........................... 22
   HMI – Human-Machine Interface Configuration .......... 24
   INFO – Description of Algorithm ........................ 26
   IODRV – The REXYGEN system input/output driver ....... 27
   IOTASK – Driver-triggered task of the REXYGEN system ........................... 29
   LPBRK – Loop break .................................... 30
   MODULE – Extension module of the REXYGEN system .... 31
   PROJECT – Additional Project Settings ................. 32
   QTASK – Quick task of the REXYGEN system ............. 33
   SLEEP – Timing in Simulink ........................... 34
   SRTF – Set run-time flags ........................... 35
   OSCALL – Operating system calls ........................ 37
   TASK – Standard task of the REXYGEN system ......... 38
   TIODRV – The REXYGEN system input/output driver with tasks ....... 40
   WWW – Internal Web Server Content ..................... 42

3 INOUT – Input and output blocks ......................... 43
   Display – Numeric display of input values ............. 44
   From, INSTD – Signal connection or input ............. 45
   Goto, OUTSTD – Signal source or output ................. 47
   GotoTagVisibility – Visibility of the signal source .... 49
   Inport, Outport – Input and output port ................. 50
   SubSystem – Subsystem block ........................... 52
   INQUAD, INQCT, INHEXD – Multi-input blocks ........... 54
   OUTQUAD, OUTQCT, OUTHEXD – Multi-output blocks .... 55
4 MATH – Math blocks

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>Absolute value</td>
<td>65</td>
</tr>
<tr>
<td>ADD</td>
<td>Addition of two signals</td>
<td>66</td>
</tr>
<tr>
<td>ADDQAD, ADDQCT, ADDHED</td>
<td>Multi-input addition</td>
<td>67</td>
</tr>
<tr>
<td>CNB</td>
<td>Boolean (logic) constant</td>
<td>68</td>
</tr>
<tr>
<td>CNE</td>
<td>Enumeration constant</td>
<td>69</td>
</tr>
<tr>
<td>CNI</td>
<td>Integer constant</td>
<td>70</td>
</tr>
<tr>
<td>CNR</td>
<td>Real constant</td>
<td>71</td>
</tr>
<tr>
<td>DIF</td>
<td>Difference</td>
<td>72</td>
</tr>
<tr>
<td>DIV</td>
<td>Division of two signals</td>
<td>73</td>
</tr>
<tr>
<td>EAS</td>
<td>Extended addition and subtraction</td>
<td>74</td>
</tr>
<tr>
<td>EMD</td>
<td>Extended multiplication and division</td>
<td>75</td>
</tr>
<tr>
<td>FNX</td>
<td>Evaluation of single-variable function</td>
<td>76</td>
</tr>
<tr>
<td>FXY</td>
<td>Evaluation of two-variables function</td>
<td>78</td>
</tr>
<tr>
<td>GAIN</td>
<td>Multiplication by a constant</td>
<td>80</td>
</tr>
<tr>
<td>GRADS</td>
<td>Gradient search optimization</td>
<td>81</td>
</tr>
<tr>
<td>IADD</td>
<td>Integer addition</td>
<td>83</td>
</tr>
<tr>
<td>ISUB</td>
<td>Integer subtraction</td>
<td>85</td>
</tr>
<tr>
<td>DMUL</td>
<td>Integer multiplication</td>
<td>86</td>
</tr>
<tr>
<td>DIV</td>
<td>Integer division</td>
<td>88</td>
</tr>
<tr>
<td>DMD</td>
<td>Remainder after integer division</td>
<td>89</td>
</tr>
<tr>
<td>LIN</td>
<td>Linear interpolation</td>
<td>90</td>
</tr>
<tr>
<td>MUL</td>
<td>Multiplication of two signals</td>
<td>91</td>
</tr>
<tr>
<td>POL</td>
<td>Polynomial evaluation</td>
<td>92</td>
</tr>
<tr>
<td>REC</td>
<td>Reciprocal value</td>
<td>93</td>
</tr>
<tr>
<td>REL</td>
<td>Relational operator</td>
<td>94</td>
</tr>
<tr>
<td>RT0I</td>
<td>Real to integer number conversion</td>
<td>95</td>
</tr>
<tr>
<td>SQR</td>
<td>Square value</td>
<td>96</td>
</tr>
<tr>
<td>SQRT</td>
<td>Square root</td>
<td>97</td>
</tr>
<tr>
<td>SUB</td>
<td>Subtraction of two signals</td>
<td>98</td>
</tr>
</tbody>
</table>

5 ANALOG – Analog signal processing

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABRSOT</td>
<td>Processing data from absolute position sensor</td>
<td>101</td>
</tr>
<tr>
<td>ASW</td>
<td>Switch with automatic selection of input</td>
<td>103</td>
</tr>
<tr>
<td>AVG</td>
<td>Moving average filter</td>
<td>105</td>
</tr>
<tr>
<td>AVS</td>
<td>Motion control unit</td>
<td>106</td>
</tr>
<tr>
<td>EPF</td>
<td>Band-pass filter</td>
<td>107</td>
</tr>
</tbody>
</table>
CONTENTS

6 GEN – Signal generators

6.1 ANLS – Controlled generator of piecewise linear function... 149
6.2 BINS – Controlled binary sequence generator... 150
6.3 BIS – Binary sequence generator... 152
6.4 MP – Manual pulse generator... 154
6.5 PRBS – Pseudo-random binary sequence generator... 156

7 REG – Function blocks for control

7.1 ARLY – Advance relay... 161
7.2 FLCU – Fuzzy logic controller unit... 163
7.3 FRID – Frequency response identification... 164
7.4 ISPM – Identification of a three parameter model... 166
7.5 LC – Lead compensator... 168
7.6 LLC – Lead-lag compensator... 170
7.7 MCU – Manual control unit... 171
7.8 PIDAT – PID controller with relay autotuner... 172
7.9 PIDC – PID controller with defined static error... 174
8 LOGIC – Logic control

ANDL – Logical product of two signals ........................................... 232
ANDQUAD, ANDOT, ANDHEXD – Logical product of multiple signals ........ 233
ATMT – Finite-state automaton ...................................................... 234
BDQCT, BDHEXD – Bitwise demultiplexers ..................................... 237
BITOP – Bitwise operation ........................................................... 238
BMQCT, BMHEXD – Bitwise multiplexers ....................................... 239
COUNT – Controlled counter ......................................................... 240
EATMT – Extended finite-state automaton ....................................... 242
EDGE_ – Falling/rising edge detection in a binary signal ................. 245
EQ – Equivalence of two signals ................................................... 246
INTSM – Integer number bit shift and mask .................................... 247
ISSW – Simple switch for integer signals ...................................... 248
INTSM – Integer number bit shift and mask .................................... 249
ITOI – Transformation of integer and binary numbers ..................... 250
NOT_ – Boolean complementation ................................................. 251
ORL – Logical sum of two signals ............................................... 252
ORQUAD, ORDOT, ORHEXD – Logical sum of multiple signals ........ 253
RS – Reset-set flip-flop circuit .................................................... 254
SR – Set-reset flip-flop circuit ................................................... 255
TIMER_ – Multipurpose timer ...................................................... 256
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>TIME – Blocks for handling time</td>
<td>259</td>
</tr>
<tr>
<td></td>
<td>DATE, – Current date</td>
<td>260</td>
</tr>
<tr>
<td></td>
<td>DATETIME – Get, set and convert time</td>
<td>261</td>
</tr>
<tr>
<td></td>
<td>TIME – Current time</td>
<td>263</td>
</tr>
<tr>
<td></td>
<td>WSCH – Weekly schedule</td>
<td>264</td>
</tr>
<tr>
<td>10</td>
<td>ARC – Data archiving</td>
<td>267</td>
</tr>
<tr>
<td>10.1</td>
<td>Functionality of the archiving subsystem</td>
<td>268</td>
</tr>
<tr>
<td>10.2</td>
<td>Generating alarms and events</td>
<td>269</td>
</tr>
<tr>
<td></td>
<td>ALB, ALBI – Alarms for Boolean value</td>
<td>269</td>
</tr>
<tr>
<td></td>
<td>ALN, ALNI – Alarms for numerical value</td>
<td>271</td>
</tr>
<tr>
<td></td>
<td>ARS – Archive store value</td>
<td>274</td>
</tr>
<tr>
<td>10.3</td>
<td>Trends recording</td>
<td>276</td>
</tr>
<tr>
<td></td>
<td>ACD – Archive compression using Delta criterion</td>
<td>276</td>
</tr>
<tr>
<td></td>
<td>TRND – Real-time trend recording</td>
<td>278</td>
</tr>
<tr>
<td></td>
<td>TRNDV – Real-time trend recording with vector input</td>
<td>281</td>
</tr>
<tr>
<td>10.4</td>
<td>Archive management</td>
<td>282</td>
</tr>
<tr>
<td></td>
<td>AFLUSH – Forced archive flushing</td>
<td>282</td>
</tr>
<tr>
<td>11</td>
<td>STRING – Blocks for string operations</td>
<td>285</td>
</tr>
<tr>
<td></td>
<td>CNS – String constant</td>
<td>286</td>
</tr>
<tr>
<td></td>
<td>CONCAT – Concat string by pattern</td>
<td>287</td>
</tr>
<tr>
<td></td>
<td>FIND – Find a Substring</td>
<td>288</td>
</tr>
<tr>
<td></td>
<td>ITOS – Integer number to string conversion</td>
<td>289</td>
</tr>
<tr>
<td></td>
<td>LEN – String length</td>
<td>290</td>
</tr>
<tr>
<td></td>
<td>MID – Substring Extraction</td>
<td>291</td>
</tr>
<tr>
<td></td>
<td>PJROCT – Parse JSON string (real output)</td>
<td>292</td>
</tr>
<tr>
<td></td>
<td>PJSOCT – Parse JSON string (string output)</td>
<td>293</td>
</tr>
<tr>
<td></td>
<td>REGEXP – Regular expression parser</td>
<td>294</td>
</tr>
<tr>
<td></td>
<td>REPLACE – Replace substring</td>
<td>297</td>
</tr>
<tr>
<td></td>
<td>RTOS – Real Number to String Conversion</td>
<td>298</td>
</tr>
<tr>
<td></td>
<td>SELSOCT – Selector switch for string signals</td>
<td>299</td>
</tr>
<tr>
<td></td>
<td>STOR – String to real number conversion</td>
<td>300</td>
</tr>
<tr>
<td>12</td>
<td>PARAM – Blocks for parameter handling</td>
<td>301</td>
</tr>
<tr>
<td></td>
<td>GETPA – Block for remote array parameter acquisition</td>
<td>302</td>
</tr>
<tr>
<td></td>
<td>GETPR, GETPI, GETPB – Blocks for remote parameter acquisition</td>
<td>304</td>
</tr>
<tr>
<td></td>
<td>GETPS – * Block for remote string parameter acquisition</td>
<td>306</td>
</tr>
<tr>
<td></td>
<td>PARA – Block with input-defined array parameter</td>
<td>307</td>
</tr>
<tr>
<td></td>
<td>PARE – Block with input-defined enumeration parameter</td>
<td>308</td>
</tr>
<tr>
<td></td>
<td>PARR, PARI, PARB – Blocks with input-defined parameter</td>
<td>309</td>
</tr>
<tr>
<td></td>
<td>PARs – * Block with input-defined string parameter</td>
<td>311</td>
</tr>
<tr>
<td></td>
<td>SETPA – Block for remote array parameter setting</td>
<td>312</td>
</tr>
<tr>
<td></td>
<td>SETPR, SETPI, SETPB – Blocks for remote parameter setting</td>
<td>314</td>
</tr>
</tbody>
</table>
13 MODEL – Dynamic systems simulation

COELESMS – Continuous state space model of a linear system with time delay 326
CSSM – Continuous state space model of a linear system 329
DOELESMS – Discrete state space model of a linear system with time delay 331
DSSM – Discrete state space model of a linear system 333
FOPDT – First order plus dead-time model 335
MDL – Process model 336
MDLI – Process model with input-defined parameters 337
MV – Motorized valve drive 338
SOPDT – Second order plus dead-time model 339

14 MATRIX – Blocks for matrix and vector operations

CWA – Array (vector/matrix) constant 344
MB_DASUM – Sum of the absolute values 345
MB_DAXPY – Performs $y := a \cdot x + y$ for vectors $x,y$ 346
MB_DCOPY – Copies vector $x$ to vector $y$ 348
MB-DDOT – Dot product of two vectors 350
MB_DGEMM – Performs $C := \alpha \cdot \text{op}(A) \cdot \text{op}(B) + \beta \cdot C$, where $\text{op}(X) = X$ or $\text{op}(X) = X^\top$ 352
MB_DGEMV – Performs $y := \alpha \cdot A \cdot x + \beta \cdot y$ or $y := \alpha \cdot A^\top \cdot x + \beta \cdot y$ 354
MB_DGER – Performs $A := \alpha \cdot x \cdot y^\top + A$ 356
MB_DNRM2 – Euclidean norm of a vector 358
MB_DROT – Plain rotation of a vector 359
MB_DSCAL – Scales a vector by a constant 361
MB_DSWAP – Interchanges two vectors 362
MB_DTRMM – Performs $B := \text{op}(A) \cdot B$ or $B := \text{op}(A) \cdot B$, where $\text{op}(X) = X$ or $\text{op}(X) = X^\top$ for triangular matrix $A$ 364
MB_DTRMV – Performs $x := A \cdot x$ or $x := A^\top \cdot x$ for triangular matrix $A$ 366
MB_DTRSV – Solves one of the system of equations $A \cdot x = b$ or $A^\top \cdot x = b$ for triangular matrix $A$ 369
ML_DGEBAK – Backward transformation to ML_DGEBAL of left or right eigenvectors 372
ML_DGEBAL – Balancing of a general real matrix 374
ML_DGEBRD – Reduces a general real matrix to bidiagonal form by an orthogonal transformation 376
ML_DGECT – Estimates the reciprocal of the condition number of a general real matrix 378
CONTENTS

ML_DGEES – Computes the eigenvalues, the Schur form, and, optionally, the matrix of Schur vectors ........................................... 380
ML_DGEEV – Computes the eigenvalues and, optionally, the left and/or right eigenvectors ......................................................... 382
ML_DGEHRD – Reduces a real general matrix A to upper Hessenberg form .................................................................................... 384
ML_DGELQF – Computes an LQ factorization of a real M-by-N matrix A ................................................................. 386
ML_DGELSD – Computes the minimum-norm solution to a real linear least squares problem .................................................... 388
ML_DGELRF – Computes an QR factorization of a real M-by-N matrix A ................................................................. 390
ML_DGESDD – Computes the singular value decomposition (SVD) of a real M-by-N matrix A .................................................. 392
ML_DLAOPY – Copies all or part of one matrix to another matrix .............................................................................................. 394
ML_DLANGE – Computes one of the matrix norms of a general matrix ...................................................................................... 396
ML_DLASET – Initializes the off-diagonal elements and the diagonal elements of a matrix to given values ....................... 398
ML_DTRSYL – Solves the real Sylvester matrix equation for quasi-triangular matrices A and B ................................................. 400
MX_AT – Get Matrix/Vector element ..................................................................................................................................... 402
MX_ATSET – Set Matrix/Vector element ................................................................................................................................. 403
MX_CNADD – Add scalar to each Matrix/Vector element ......................................................................................................... 404
MX_CNMLM – Multiply a Matrix/Vector by a scalar .................................................................................................................. 405
MX_CTQPPA – Discretizes continuous model given by (A,B) to (Ad,Bd) using Padé approximations .............................................. 406
MX_DIM – Matrix/Vector dimensions ..................................................................................................................................... 408
MX_DIMSET – Set Matrix/Vector dimensions ............................................................................................................................ 409
MX_DSAGET – Set subarray of A into B .................................................................................................................................. 411
MX_DSAREF – Set reference to subarray of A into B .................................................................................................................... 413
MX_DSASET – Set A into subarray of B ...................................................................................................................................... 415
MX_DTRNSP – General matrix transposition: B := alpha*A^T .................................................................................................. 417
MX_DTRNSQ – Square matrix in-place transposition: A := alpha*A^T ....................................................................................... 419
MX.Fill – Fill real matrix or vector ....................................................................................................................................... 420
MX_MAT – Matrix data storage block ....................................................................................................................................... 421
MX_RAND – Randomly generated matrix or vector ..................................................................................................................... 422
MX_REFCOPY – Copies input references of matrices A and B to their output references ......................................................... 424
MX_SLFS – Save or load a Matrix/Vector into file or string ......................................................................................................... 425
MX_VEC – Vector data storage block ....................................................................................................................................... 427
MX_WRITE – Write a Matrix/Vector to the console/system log ............................................................................................... 428
RTUV – Vector multiplexer ......................................................................................................................................................... 430
SWVMR – Vector/matrix/reference signal switch ...................................................................................................................... 431
VTOR – Vector demultiplexer ..................................................................................................................................................... 432
15 SPEC – Special blocks

EPC – External program call ............................................. 433
HTTP – HTTP GET or POST request (obsolete) ..................... 434
HTTP2 – Block for generating HTTP GET or POST requests ....... 437
SMTP – Send e-mail message via SMTP ................................. 441
RDC – Remote data connection ........................................... 443
REXLANG – User programmable block ................................. 448

16 LANG – Special blocks

PYTHON – User programmable block in Python ....................... 468

17 MQTT – Communication via MQTT protocol ....................... 475

MqttPublish – Publish MQTT message ................................ 476
MqttSubscribe – Subscribe to MQTT topic ............................ 478

18 MC_SINGLE – Motion control - single axis blocks ............... 481

MC_Axis – Motion control axis ........................................ 484
MC_AccelerationProfile, MCP_AccelerationProfile – Acceleration profile ........................................ 489
MC_Halt, MCP_Halt – Stopping a movement (interruptible) ........ 493
MC_HaltSuperimposed, MCP_HaltSuperimposed – Stopping a movement (superimposed and interruptible) ................. 494
MC_Home, MCP_Home – Homing ........................................ 495
MC_MoveAbsolute, MCP_MoveAbsolute – Move to position (absolute coordinate) ........................................ 497
MC_MoveAdditive, MCP_MoveAdditive – Move to position (relative to previous motion) ....................................... 501
MC_MoveRelative, MCP_MoveRelative – Move to position (relative to execution point) ........................................ 504
MC_MoveSuperimposed, MCP_MoveSuperimposed – Superimposed move ........................................ 507
MC_MoveContinuousAbsolute, MCP_MoveContinuousAbsolute – Move to position (absolute coordinate) ....................... 510
MC_MoveContinuousRelative, MCP_MoveContinuousRelative – Move to position (relative to previous motion) ............... 513
MC_MoveVelocity, MCP_MoveVelocity – Move with constant velocity ........................................ 517
MC_PositionProfile, MCP_PositionProfile – Position profile .... 521
MC_Power – Axis activation (power on/off) ......................... 525
MC_ReadActualPosition – Read actual position ...................... 526
MC_ReadAxisError – Read axis error .................................. 527
MC_ReadBoolParameter – Read axis parameter (bool) .............. 528
MC_ReadParameter – Read axis parameter ............................ 529
MC_ReadStatus – Read axis status .................................... 531
MC_Reset – Reset axis errors .......................................... 533
MC_SetOverride, MCP_SetOverride – Set override factors .......... 534
CONTENTS

MC_Stop, MGP_Stop – Stopping a movement ........................................... 536
MC_TorqueControl, MGP_TorqueControl – Torque/force control .......... 538
MC_VelocityProfile, MGP_VelocityProfile – Velocity profile ............. 541
MC_WriteBoolParameter – Write axis parameter (bool) .................... 545
MC_WriteParameter – Write axis parameter ................................. 546
RM_AxisOut – Axis output ....................................................... 547
RM_AxisSpline – Commanded values interpolation .......................... 548
RM_Track – Tracking and inching .............................................. 549

19 MC_MULTI – Motion control - multi axis blocks .......................... 551
MC_CamIn, MGP_CamIn – Engage the cam .................................... 552
MC_CamOut – Disengage the cam ............................................. 556
MGP_CamTableSelect – Cam definition ....................................... 558
MC_CombineAxes, MGP_CombineAxes – Combine the motion of 2 axes into a third axis ..................................................... 560
MC_GearIn, MGP_GearIn – Engage the master/slave velocity ratio ....... 560
MC_GearInPos, MGP_GearInPos – Engage the master/slave velocity ratio in defined position ........................................... 566
MC_GearOut – Disengage the master/slave velocity ratio .................. 571
MC_PhasingAbsolute, MGP_PhasingAbsolute – Phase shift in synchro-
nized motion (absolute coordinates) ....................................... 573
MC_PhasingRelative, MGP_PhasingRelative – Phase shift in synchro-
nized motion (relative coordinates) ...................................... 576

20 MC_COORD – Motion control - coordinated movement blocks ............ 579
RM_AxesGroup – Axes group for coordinated motion control ............. 582
RM_Feed – * MC Feeder ....................................................... 585
RM_Gcode – * CNC motion control .......................................... 586
MC_AddAxisToGroup – Adds one axis to a group ............................ 588
MC_UnGroupAllAxes – Removes all axes from the group ................ 589
MC_GroupEnable – Changes the state of a group to GroupEnable ...... 590
MC_GroupDisable – Changes the state of a group to GroupDisabled ... 591
MC_SetCartesianTransform – Sets Cartesian transformation ............ 592
MC_ReadCartesianTransform – Reads the parameter of the cartesimal tran-
sformation ........................................................................ 594
MC_GroupSetPosition, MGP_GroupSetPosition – Sets the position of all axes in a group .................................................. 595
MC_GroupReadActualPosition – Read actual position in the selected co-
ordinate system .................................................................... 597
MC_GroupReadActualVelocity – Read actual velocity in the selected co-
ordinate system .................................................................. 598
MC_GroupReadActualAcceleration – Read actual acceleration in the se-
lected coordinate system ...................................................... 599
MC_GroupStop – Stopping a group movement .................................. 600
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC_GroupHalt</td>
<td>Stopping a group movement (interruptible)</td>
<td>603</td>
</tr>
<tr>
<td>MC_GroupInterrupt, MCP_GroupInterrupt</td>
<td>Read a group interrupt</td>
<td>608</td>
</tr>
<tr>
<td>MC_GroupContinue</td>
<td>Continuation of interrupted movement</td>
<td>609</td>
</tr>
<tr>
<td>MC_GroupReadStatus</td>
<td>Read a group status</td>
<td>610</td>
</tr>
<tr>
<td>MC_GroupReadError</td>
<td>Read a group error</td>
<td>611</td>
</tr>
<tr>
<td>MC_GroupReset</td>
<td>Reset axes errors</td>
<td>612</td>
</tr>
<tr>
<td>MC_MoveLinearAbsolute</td>
<td>Linear move to position (absolute coordinates)</td>
<td>613</td>
</tr>
<tr>
<td>MC_MoveLinearRelative</td>
<td>Linear move to position (relative to execution point)</td>
<td>617</td>
</tr>
<tr>
<td>MC_MoveCircularAbsolute</td>
<td>Circular move to position (absolute coordinates)</td>
<td>621</td>
</tr>
<tr>
<td>MC_MoveCircularRelative</td>
<td>Circular move to position (relative to execution point)</td>
<td>625</td>
</tr>
<tr>
<td>MC_MoveDirectAbsolute</td>
<td>Direct move to position (absolute coordinates)</td>
<td>629</td>
</tr>
<tr>
<td>MC_MoveDirectRelative</td>
<td>Direct move to position (relative to execution point)</td>
<td>632</td>
</tr>
<tr>
<td>MC_MovePath</td>
<td>General spatial trajectory generation</td>
<td>635</td>
</tr>
<tr>
<td>MC_GroupSetOverride</td>
<td>Set group override factors</td>
<td>637</td>
</tr>
<tr>
<td>21 CanDrv</td>
<td>Communication via CAN bus</td>
<td>639</td>
</tr>
<tr>
<td>CanItem</td>
<td>Secondary received CAN message</td>
<td>640</td>
</tr>
<tr>
<td>CanRecv</td>
<td>Receive CAN message</td>
<td>641</td>
</tr>
<tr>
<td>CanSend</td>
<td>Send CAN message</td>
<td>643</td>
</tr>
<tr>
<td>22 OpcUaDrv</td>
<td>Communication using OPC UA</td>
<td>645</td>
</tr>
<tr>
<td>OpcUaReadValue</td>
<td>Read value from OPC UA Server</td>
<td>646</td>
</tr>
<tr>
<td>OpcUaServerValue</td>
<td>Expose value as an OPC UA Node</td>
<td>648</td>
</tr>
<tr>
<td>OpcUaWriteValue</td>
<td>Write value to OPC UA Server</td>
<td>650</td>
</tr>
</tbody>
</table>

A Licensing options | 653 |
B Licensing of individual function blocks | 655 |
C Error codes of the RXXYGEN system | 667 |

Bibliography | 673 |
Index | 675 |

Note: Only a partial documentation is available in blocks marked by *.
Chapter 1

Introduction

The manual “REXYGEN system function blocks” is a reference manual for the REXYGEN system function block library RexLib. It includes description and detailed information about all function blocks RexLib consists of.

1.1 How to use this manual

The extensive function block library RexLib, which is a standard part of the REXYGEN system, is divided into smaller sets of logically related blocks, the so-called categories (sublibraries). A separate chapter is devoted to each category, introducing the general properties of the whole category and its blocks followed by a detailed description of individual function blocks.

The content of individual chapters of this manual is following:

1 Introduction
   This introductory chapter familiarizing the readers with the content and ordering of the manual. A convention used for individual function blocks description is presented.

2 EXEC – Runtime executive configuration
   Blocks used mainly for configuration of the structure, priorities and timing of individual objects linked to the real-time subsystem of the REXYGEN system (the RexCore program) are described in this chapter.

3 INOUT – Input and output blocks
   This sublibrary consists of the blocks used mainly for the REXYGEN system. These blocks provide the connection between the control tasks and input/output drivers.

4 MATH – Mathematic blocks
   The blocks for arithmetic operations and basic math functions.
5 ANALOG – Analog signal processing
The integrator, derivator, time delay, moving average, various filters, comparators and selectors can be found among the blocks for analog signal processing. The starting unit block (AVS) is also very interesting.

6 GEN – Signal generators
This chapter deals with analog and logic signal generators.

7 REG – Function blocks for control
The control function blocks form the most extensive sublibrary of the RexLib library. Blocks ranging from simple dynamic compensators to several modifications of PID (P, I, PI, PD a PID) controller and some advanced controllers are included. The blocks for control schemes switching and conversion of output signals for various types of actuators can be found in this sublibrary. The involved controllers include the PIDG block, enabling online switching of parameter sets (the so-called gain scheduling), the PIDMA block with built-in moment autotuner, the PIDAT block with built in relay autotuner, the FLCU fuzzy controller or the PSMPC predictive controller, etc.

8 LOGIC – Logic control
This chapter describes blocks for combinational and sequential logic control including the simplest Boolean operations (not, and, or) and also more complex blocks like the sequential logic automat ATMT implementing the SFC standard (Sequential Function Charts, formerly Grafcet).

10 ARC – Data archiving
This sublibrary contains blocks for alarms generation and blocks for storing trend data directly on the target device.

12 PARAM – Parameter handling
This sublibrary contains blocks for parameter handling, namely saving, loading and remote manipulation with parameters.

13 MODEL – Dynamic systems modeling
The REXGEN system can also be used for creating real-time mathematical models of dynamic systems. The function blocks of this sublibrary were developed for such cases.

14 MATRIX – Working with matrix and vector data
Function blocks for handling vector and matrix data in REXGEN are included in this sublibrary.

18 MC_SINGLE – Single-axis motion control
Function blocks of this sublibrary were developed according to the PLCopen Motion Control standard for single axis motion control.
1.2. THE FUNCTION BLOCK DESCRIPTION FORMAT

19 MC_MULTI – Multi-axes motion control
Function blocks of this sublibrary were developed according to the PLCopen Motion Control standard for motion control in multiple axes.

20 MC_COORD – Coordinated motion control
Function blocks of this sublibrary were developed according to the PLCopen Motion Control standard for coordinated motion control.

15 SPEC – Special blocks
The most interesting blocks of this sublibrary are the REXLANG and RDC blocks. It is possible to compile and interpret user algorithms using the REXLANG block, whose programming language is very similar to the C language (the syntax of the REXLANG commands is mostly the same as in the C language). The RDC block can be used for real-time communication between two REXYGEN-enabled target devices.

The individual chapters of this reference guide are not much interconnected, which means they can be read in almost any order or even only the necessary information for specific block can be read for understanding the function of that block. The electronic version of this manual (in the .pdf format) is well-suited for such case as it is equipped with hypertext bookmarks and contents, which makes the look-up of individual blocks very easy.

Despite of that it is recommended to read the following subchapter, which describes the conventions used for description of individual blocks in the rest of this manual.

1.2 The function block description format

The description of each function block consists of several sections (in the following order):

Block Symbol – displays the graphical symbol of the block

Function Description – brief description of the block function, omitting too detailed information.

Inputs – detailed description of all inputs of the block

Outputs – detailed description of all outputs of the block

Parameters – detailed description of all parameters of the block

Examples – a simple example of the use of the block in the context of other blocks and optional graph with input and output signals for better understanding of the block function.

If the block function is obvious, the section Examples is omitted. In case of block with no input or no output the corresponding section is omitted as well.

The inputs, outputs and parameters description has a tabular form:
CHAPTER 1. INTRODUCTION

\(<\text{name}>\)  \([\text{nam}]\) Detailed description of the input (output, parameter) \(<\text{name}>\). Mathematical symbol \(\text{nam}\) on the right side of the first column is used in the equations in the Function Description section. It is listed only if it differs from the name more than typographically. If the variable value is limited to only enumerated values, the meaning of these values is explained in this column.  \([\circ\text{<def>}]|\downarrow\text{<min>}|\uparrow\text{<max>}\)

The meaning of the three columns is quite obvious. The third column contains the item \(<\text{type}>\). The REXGEN control system supports the types listed in table 1.1. But the most frequently used types are \textbf{Bool} for Boolean variables, \textbf{Long} (I32) for integer variables and \textbf{Double} (F64) for real variables (in floating point arithmetics).

Each described variable (input, output or parameter) has a default value \(<\text{def}>\) in the REXGEN system, which is preceded by the \(\circ\) symbol. Also it has upper and lower limits, preceded by the symbols \(\downarrow\) and \(\uparrow\) respectively. All these three values are optional (marked by \([\ ]\)). If the value \(\circ\text{<def>}\) is not listed in the second column, it is equal to zero. If the values of \(\downarrow\text{<min>}\) and/or \(\uparrow\text{<max>}\) are missing, the limits are given by the the minimum and/or maximum of the corresponding type, see table 1.1\(^1\).

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bool</td>
<td>Boolean value 0 or 1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Byte (U8)</td>
<td>8-bit integer number without the sign</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>Short (I16)</td>
<td>16-bit integer number with the sign</td>
<td>-32768</td>
<td>32767</td>
</tr>
<tr>
<td>Long (I32)</td>
<td>32-bit integer number with the sign</td>
<td>-2147483648</td>
<td>2147483647</td>
</tr>
<tr>
<td>Large (I64)</td>
<td>64-bit integer number with the sign</td>
<td>-9.2234 \times 10^{18}</td>
<td>9.2234 \times 10^{18}</td>
</tr>
<tr>
<td>Word (U16)</td>
<td>16-bit integer number without the sign</td>
<td>0</td>
<td>65535</td>
</tr>
<tr>
<td>DWord (U32)</td>
<td>32-bit integer number without the sign</td>
<td>0</td>
<td>4294967295</td>
</tr>
<tr>
<td>Float (F32)</td>
<td>32-bit real number in floating point arithmetics</td>
<td>-3.4 \times 10^{38}</td>
<td>3.4 \times 10^{38}</td>
</tr>
<tr>
<td>Double (F64)</td>
<td>64-bit real number in floating point arithmetics</td>
<td>-1.7 \times 10^{308}</td>
<td>1.7 \times 10^{308}</td>
</tr>
<tr>
<td>String</td>
<td>character string</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1.1: Types of variables in the REXGEN system.

1.3 Conventions for variables, blocks and subsystems naming

Several conventions are used to simplify the use of the REXGEN control system. All used variable types were defined in the preceding chapter. The term variable refers to function block inputs, outputs and parameters in this chapter. The majority of the blocks uses only the following three types:

\(^1\) Precise range of the Large data type is -9223372036854775808 to 9223372036854775807.
1.4. THE SIGNAL QUALITY CORRESPONDING WITH OPC

**Bool** – for two-state logic variables, e.g. on/off, yes/no or true/false. The logic one (yes, true, on, 1) is referred to as on in this manual. Similarly the logic zero (no, false, off, 0) is represented by off. This holds also for REXYGEN Studio. Other tools and 3rd party software may display these values as 1 for on and 0 for off. The names of logic variables consist of uppercase letters, e.g. **RUN**, **YCN**, **R1**, **UP**, etc.

**Long** (I32) – for integer values, e.g. set of parameters ID, length of trend buffer, type of generated signal, error code, counter output, etc. The names of integer variables use usually lowercase letters and the initial character (always lowercase) is in most cases {i, k, l, m, n, or o}, e.g. **ips**, **l**, **isig**, **iE**, etc. But several exceptions to this rule exist, e.g. **cnt** in the **COUNT** block, **btype**, **ptype1**, **pfac** and **afac** in the **TRND** block, etc.

**Double** (F64) – for floating point values (real numbers), e.g. gain, saturation limits, results of the majority of math functions, PID controller parameters, time interval lengths in seconds, etc. The names of floating point variables use only lowercase letters, e.g. **hilim**, **y**, **ti**, **tt**.

The function block names in the REXYGEN system use uppercase letters, numbers and the ‘_’ (underscore) character. It is recommended to append a lowercase user-defined string to the standard block name when creating user instances of function blocks.

It is explicitly not recommended to use diacritic and special characters like spaces, CR (end of line), punctuation, operators, etc. in the user-defined names. The use of such characters limits the transferability to various platforms and it can lead to incomprehension. The names are checked by the REXYGEN Compiler compiler which generates warnings if inappropriate characters are found.

1.4 The signal quality corresponding with OPC

Every signal (input, output, parameter) in the REXYGEN system has the so-called *quality flags* in addition to its own value of corresponding type (table 1.1). The quality flags in the REXYGEN system correspond with the OPC (OLE for Process Control) specification [1]. They can be represented by one byte, whose structure is explained in the table 1.2.

<table>
<thead>
<tr>
<th>Bit number</th>
<th>7 6</th>
<th>5 4 3 2</th>
<th>1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit weight</td>
<td>128</td>
<td>64</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>8 4</td>
<td>2 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit field</th>
<th>Quality</th>
<th>Substatus</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q</td>
<td>S S S S</td>
<td>L L</td>
</tr>
<tr>
<td>BAD</td>
<td>0 0</td>
<td>S S S S</td>
<td>L L</td>
</tr>
<tr>
<td>UNCERTAIN</td>
<td>0 1</td>
<td>S S S S</td>
<td>L L</td>
</tr>
<tr>
<td>not used in OPC</td>
<td>1 0</td>
<td>S S S S</td>
<td>L L</td>
</tr>
<tr>
<td>GOOD</td>
<td>1 1</td>
<td>S S S S</td>
<td>L L</td>
</tr>
</tbody>
</table>

**Table 1.2: The quality flags structure**
CHAPTER 1. INTRODUCTION

The basic quality type is determined by the QQ flags in the two most important bits. Based on these, the quality is distinguished between GOOD, UNCERTAIN, and BAD. The four SSSS bits provide more detailed information about the signal. They have different meaning for each basic quality. The two least significant bits LL inform whether the value exceeded its limits or if it is constant. Additional details and the meaning of all bits can be found in [1], chapter 6.8.
Chapter 2

EXEC – Real-time executive configuration

Contents

<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARC – The REXYGEN system archive</td>
<td>20</td>
</tr>
<tr>
<td>EXEC – Real-time executive</td>
<td>22</td>
</tr>
<tr>
<td>HMI – Human-Machine Interface Configuration</td>
<td>24</td>
</tr>
<tr>
<td>INFO – Description of Algorithm</td>
<td>26</td>
</tr>
<tr>
<td>IODRV – The REXYGEN system input/output driver</td>
<td>27</td>
</tr>
<tr>
<td>IOTASK – Driver-triggered task of the REXYGEN system</td>
<td>29</td>
</tr>
<tr>
<td>LPBK – Loop break</td>
<td>30</td>
</tr>
<tr>
<td>MODULE – Extension module of the REXYGEN system</td>
<td>31</td>
</tr>
<tr>
<td>PROJECT – Additional Project Settings</td>
<td>32</td>
</tr>
<tr>
<td>QTASK – Quick task of the REXYGEN system</td>
<td>33</td>
</tr>
<tr>
<td>SLEEP – Timing in Simulink</td>
<td>34</td>
</tr>
<tr>
<td>SRTF – Set run-time flags</td>
<td>35</td>
</tr>
<tr>
<td>OS CALL – Operating system calls</td>
<td>37</td>
</tr>
<tr>
<td>TASK – Standard task of the REXYGEN system</td>
<td>38</td>
</tr>
<tr>
<td>TIODRV – The REXYGEN system input/output driver with tasks</td>
<td>40</td>
</tr>
<tr>
<td>WWW – Internal Web Server Content</td>
<td>42</td>
</tr>
</tbody>
</table>
ARC – The REXYGEN system archive

Block Symbol

 Licence: STANDARD

Function Description

The ARC block is intended for archives configuration in the REXYGEN control system. The archives can be used for continuous recording of alarms, events and history trends directly on the target platform. The output Archives of the EXEC block must be connected to the prev input of the first archive. The following archives can be added by connecting the input prev with the preceding archive’s output next. Only one archive block can be connected to each next output, the output of the last archive remains unconnected. The resulting archives sequence determines the order of allocation and initialization of individual archives in the REXYGEN system and also the index of the archive, which is used in the arc parameter of the archiving blocks (see chapter 10). The archives are numbered from 1 and the maximum number of archives is limited to 15 (archive no. 0 is the internal system log).

The atype parameter determines the type of archive from the data-available-after-restarting point of view. The admissible types depend on the target platform properties, which can be inspected in the Target tab in the REXYGEN Diagnostics program after successful connecting to the target device.

Archive consists of sequenced variable-length items (memory and disk space optimization) with a timestamp. Therefore the other parameters are the total archive size in bytes asize and maximum number of timestamps nmarks for speeding-up the sequential seeking in the archive.

Input

prev Input for connecting with the next output of the preceding Long (I32) archive or with the Archives output of the EXEC block in the case of the first archive

Output

next Output for creating sequences of archives by connecting to the Long (I32) prev input of the following archive
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>atype</strong></td>
<td>Archive type</td>
</tr>
<tr>
<td>1 . . . .</td>
<td>archive is allocated in the RAM memory (data is irreversibly lost after restarting the target device)</td>
</tr>
<tr>
<td>2 . . . .</td>
<td>archive is allocated in backed-up memory, e.g. CMOS (data remains available after restarting the target device)</td>
</tr>
<tr>
<td>3 . . . .</td>
<td>archive is allocated on a drive (data remains available in the file after restarting)</td>
</tr>
<tr>
<td><strong>asize</strong></td>
<td>Size of the archive in bytes</td>
</tr>
<tr>
<td><strong>nmarks</strong></td>
<td>Number of time stamps for speeding-up sequential seeking in the archive</td>
</tr>
<tr>
<td><strong>ldamax</strong></td>
<td>Maximum size of archive per day [bytes]</td>
</tr>
<tr>
<td><strong>period</strong></td>
<td>Period of writing data to disk [s]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>01266</td>
<td>Long (132)</td>
</tr>
<tr>
<td>0102400</td>
<td>Long (132)</td>
</tr>
<tr>
<td>02720</td>
<td>Long (132)</td>
</tr>
<tr>
<td>1000</td>
<td>Large (164)</td>
</tr>
<tr>
<td>2147480000</td>
<td>Double (64)</td>
</tr>
<tr>
<td>60.0</td>
<td>Double (64)</td>
</tr>
</tbody>
</table>
EXEC - Real-time executive

Block Symbol

Licence: STANDARD

Function Description

The EXEC block is a cornerstone of the so-called project main file in the .mdl format, which configures individual subsystems of the REXYGEN system. No similar block can be found in the Matlab-Simulink system. The EXEC block and all connected configuration blocks do not implement any mathematic algorithm. Such configuration structure is used by the REXYGEN Compiler compiler during building of the overall REXYGEN control system application.

The REXYGEN system configuration consists of modules (Modules), input/output drivers (Drivers), archive subsystem (Archives) and real-time subsystem, which includes quick computation tasks (see the QTask function block description for details) and four priority levels (Level0 to Level3) for inserting computation tasks (see the TASK function block description for details).

The base (shortest) period of the application is determined by the tick parameter. This value is checked by the REXYGEN Compiler compiler as its limits vary by selected target platform. Generally speaking, the lower period is used, the higher computational requirements of the REXYGEN system runtime core (RexCore) are.

The periods of individual computation levels (Level0 to Level3) are determined by multiplying the base period tick by the parameters ntick0 to ntick3. Parameters pri0 to pri3 are the logical priorities of corresponding computation levels in the REXYGEN system. The REXYGEN system uses 32 logical priorities, which are internally mapped to the target platform operating system dependent priorities. The highest logical priority of the REXYGEN system is 0, the value 31 means the lowest. Should two tasks with different priorities run at the same time, the lower priority (higher value) task would be
interrupted by the higher priority (lower value) task.

The default priorities pri0 to pri3 reflect the commonly accepted idea that the "fast" tasks (short sampling period) should have higher priority than the "slow" ones (the so-called Rate monotonic scheduling). This means that the default priorities need not to be changed in most cases. Impetuous changes can lead to unpredictable effects!

## Outputs

<table>
<thead>
<tr>
<th>Modules</th>
<th>Output for connecting the REXYGEN system expansion modules, see the MODULE function block description for details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drivers</td>
<td>Output for connecting the REXYGEN system input/output drivers, see the IODRV and TIODRV function block descriptions for details</td>
</tr>
<tr>
<td>Archives</td>
<td>Output for archives configuration, see the ABC block</td>
</tr>
<tr>
<td>QTASK</td>
<td>Output for connecting quick tasks with the highest priority and the shortest period, see the QTASK block</td>
</tr>
<tr>
<td>Level0</td>
<td>Computation level for inserting tasks (see the TASK block) with high priority pri0 and short period determined by the ntick0 parameter</td>
</tr>
<tr>
<td>Level1</td>
<td>Computation level for inserting tasks with medium priority pri1 and medium-length period determined by the ntick1 parameter</td>
</tr>
<tr>
<td>Level2</td>
<td>Computation level for inserting tasks with low priority pri2 and long period determined by the ntick2 parameter</td>
</tr>
<tr>
<td>Level3</td>
<td>Computation level for inserting tasks with the lowest priority pri3 and the longest period determined by the ntick3 parameter</td>
</tr>
</tbody>
</table>

## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>Target Device</td>
<td>Target device for PC - Windows (String)</td>
</tr>
<tr>
<td>tick</td>
<td>Double</td>
<td>The base period (tick) of the REXYGEN system core and also the quick task (QTASK) period (in seconds)</td>
</tr>
<tr>
<td>ntick0</td>
<td>Long</td>
<td>The multiplication tick*ntick0 determines the period of tasks connected to Level0</td>
</tr>
<tr>
<td>ntick1</td>
<td>Long</td>
<td>The multiplication tick*ntick1 determines the period of tasks connected to Level1</td>
</tr>
<tr>
<td>ntick2</td>
<td>Long</td>
<td>The multiplication tick*ntick2 determines the period of tasks connected to Level2</td>
</tr>
<tr>
<td>ntick3</td>
<td>Long</td>
<td>The multiplication tick*ntick3 determines the period of tasks connected to Level3</td>
</tr>
<tr>
<td>pri0</td>
<td>Long</td>
<td>Priority of all Level0 tasks</td>
</tr>
<tr>
<td>pri1</td>
<td>Long</td>
<td>Priority of all Level1 tasks</td>
</tr>
<tr>
<td>pri2</td>
<td>Long</td>
<td>Priority of all Level2 tasks</td>
</tr>
<tr>
<td>pri3</td>
<td>Long</td>
<td>Priority of all Level3 tasks</td>
</tr>
</tbody>
</table>
CHAPTER 2. EXEC – REAL-TIME EXECUTIVE CONFIGURATION

HMI – Human-Machine Interface Configuration

Block Symbol

Licence: STANDARD

Function Description

The HMI block is a so-called "pseudo-block" which stores additional settings and parameters related to the Human-Machine Interface (HMI) and the contents of the internal web server. The only file where the block can be placed is the main project file with a single EXEC block.

The REXYGEN system currently provides three straightforward methods of how to create Human-Machine Interface:

- **WebWatch** is an auto-generated HMI from the REXYGEN Studio development tool during project compilation. It has similar look, attributes and functions as the online mode of the REXYGEN Studio development tool. The main difference is that WebWatch is stored on the target device, is available from the integrated web server and may be viewed with any modern web browser or any application that is compatible with HTML, SVG and JavaScript. The WebWatch is a perfect tool for instant creation of HMI that is suitable for system developers or integrators. It provides a graphical interaction with almost all signals in the control algorithm.

- **WebBuDi**, which is an acronym for Web Buttons and Displays, is a simple JavaScript file with several declarative blocks that describe data points which the HMI is connected to and assemble a table in which all the data is presented. It provides a textual interaction with selected signals and is suitable for system developers and integrators or may serve as a fall-back mode HMI for non-standard situations.

- **RexHMI** is a standard SVG file that is edited using REXYGEN HMI Designer. The REXYGEN HMI Designer is a great tool for creating graphical HMI that is suitable for operators and other end users.

The IncludeHMI parameter includes or excludes the HMI files from the final binary form of the project. The HmiDir specifies a path to a directory where the final HMI is located and from where it is inserted into the binary file during project compilation. The path may be absolute or relative to the project. The GenerateWebWatch specifies whether a WebWatch HMI should be generated into HmiDir during compilation. The GenerateRexHMI specifies whether a RexHMI and WebBuDi should be generated into HmiDir during compilation.

The logic of generating and including HMI during project compilation is as follows:
1. Delete all contents from HmiDir when GenerateWebWatch or GenerateRexHMI is specified.

2. Generate RexHMI and WebBuDi from SourceDir into HmiDir if GenerateRexHMI is enabled. All WebBuDi source files should be named in a *.hmi.js format and all RexHMI source files should be named in a *.hmi.svg format. The generated files are then named *.html.

3. Copy all contents from SourceDir except WebBuDi or RexHMI source files into HmiDir if IncludeHMI is enabled.

4. Insert HMI from HmiDir into binary configuration if IncludeHMI is enabled.

The block does not have any inputs or outputs. The HMI block itself does not become a part of the final binary configuration, only the files it points to do. Be careful when inserting big files or directories as the integrated web server is not designed for massive data transfers. It is possible to shrink the data by enabling gzip compression. The compression also reduces amount of data transferred to the client, but decompression must be performed by the server when a client does not support gzip compression, which brings additional load on the target device.

For a proper operation of the HMI block the compilation must be launched from the REXYGEN Studio development tool and the REXYGEN HMI Designer must be installed.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IncludeHMI</td>
<td>Include HMI files in the project</td>
<td>on</td>
</tr>
<tr>
<td>HmiDir</td>
<td>Output folder for HMI files</td>
<td>hmi</td>
</tr>
<tr>
<td>SourceDir</td>
<td>Source directory</td>
<td>hmi</td>
</tr>
<tr>
<td>GenerateWebWatch</td>
<td>Generate WebWatch HMI files</td>
<td>on</td>
</tr>
<tr>
<td>GenerateRexHMI</td>
<td>Generate HMI from SVG and JS files</td>
<td>on</td>
</tr>
<tr>
<td>RedirectToHMI</td>
<td>Web server will automatically redirect to HMI webpage if enabled</td>
<td>on</td>
</tr>
<tr>
<td></td>
<td>otherwise it will serve a standard home page as a starting page.</td>
<td></td>
</tr>
<tr>
<td>Compression</td>
<td>Enables data compression in gzip format.</td>
<td>on</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


```
INFO – Description of Algorithm

Block Symbol

License: STANDARD

Function Description

The INFO block is a so-called "pseudo-block" which stores textual information about a real-time executive. The only file where the block can be placed is a main project file with a single EXEC block and so it belongs to the EXEC category. The block does not have any inputs or outputs. The information specified with this block becomes a part of the final configuration, is stored on the target device and may be seen on different diagnostics screens but does not have any impact on execution of the control algorithm or target’s behavior.

Parameters

<table>
<thead>
<tr>
<th>Title</th>
<th>Project title</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author</td>
<td>Project author</td>
<td>String</td>
</tr>
<tr>
<td>Description</td>
<td>Brief description of the project</td>
<td>String</td>
</tr>
<tr>
<td>Customer</td>
<td>Information about a customer</td>
<td>String</td>
</tr>
</tbody>
</table>
IODRV – The REXYGEN system input/output driver

Block Symbol

Licence: STANDARD

Function Description

The input/output drivers of the REXYGEN system are implemented as extension modules (see the MODULE block). A module can contain several drivers, which are added to the REXYGEN system configuration by using the IODRV blocks. The prev input of the block must be connected with the Drivers output of the EXEC block or with the next output of a IODRV block which is already included in the configuration. There can be only one driver connected to the next output of the IODRV block. The next output of the last driver in the configuration remains unconnected. This means that the drivers create a unidirectional chain which defines the order of initialization and execution of the individual drivers.

Each driver of the REXYGEN system is identified by its name, which is defined by the classname parameter (beware, the name is case-sensitive!). If the name of the driver differs from the name of the module containing the given driver, the module name must be specified by the module parameter, it is left blank otherwise. Details about these two parameters can be found in the documentation of the corresponding REXYGEN system driver.

The majority of drivers stores its own configuration data in files with .rio extension (REXYGEN Input/Output), whose name is specified by the cfgname parameter. The .rio files are created in the same directory where the project main file is located (.mdl file with the EXEC block). Driver is configured (e.g. names of the input/output signals, connection to physical inputs/outputs, parameters of communication with the input/output device, etc.) in an embedded editor provided by the driver itself. The editor is opened when the Configure button is pressed in the parameter dialog of the IODRV block in the REXYGEN Studio program of the REXYGEN control system. In Matlab/Simulink the editor is opened upon ticking the "Tick this checkbox to call Iodrv EDIT dialog" checkbox.

The remaining parameters are useful only when the driver implements its own computational task (see the corresponding driver documentation). The factor parameter defines the driver’s task execution period by multiplying the EXEC block’s tick parameter factor times (factor*tick). The stack parameter defines the stack size in bytes. It is recommended to keep the default setting unless stated otherwise in the driver documentation. The last parameter pri defines the logical priority of the driver’s task. Inappropriate priority can influence the overall performance of the control system critically so it is highly recommended to check the driver documentation and the load of the control system (drivers, levels and tasks) in the REXYGEN Diagnostics diagnostic program.
CHAPTER 2. EXEC—REAL-TIME EXECUTIVE CONFIGURATION

Input

prev
Input for connecting the driver with the Drivers output of the EXEC block or with the next output of the preceding driver
Long (I32)

Output

next
Output for connecting to the prev input of the succeeding driver
Long (I32)

Parameters

module
Name of the module, which includes the input/output driver (mandatory only if module name differs from classname)
String

classname
I/O driver class name; case sensitive!
DrvClass
String

cfgname
Name of the driver configuration file
idrv.rio
String

factor
Multiple of the EXEC block’s tick parameter defining the driver’s task execution period
\(1 \times 10\)

stack
Stack size of the driver’s task in bytes
\(1024 \times 10240\)

pri
Logical priority of the driver’s task
\(1 \times 3 \times 3\)

timer
Driver is a source of time
Bool
IOTASK – Driver-triggered task of the REXYGEN system

Block Symbol: IOTASK

Licence: STANDARD

Function Description

Standard tasks of the REXYGEN system are integrated into the configuration using the TASK or QTASK blocks. Such tasks are executed by the system timer, whose tick is configured by the EXEC block.

But the system timer can be unsuitable in some cases, e.g. when the shortest execution period is too long or when the task should be executed by an external event (input signal interrupt) etc. In such a case the IOTASK can be executed directly by the I/O driver configured by the TI0DRV block. The user manual of the given driver provides more details about the possibility and conditions of using the above mentioned approach.

Input

<table>
<thead>
<tr>
<th>prev</th>
<th>Input for connecting the first task to the Tasks output of the TI0DRV block or for connecting to the previous task's next output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Long (i32)</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>next</th>
<th>Output for sequencing the tasks by connecting to the prev input of the following task</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Long (i32)</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>factor</th>
<th>Execution factor which can be used to determine the task execution period, see the user guide of the corresponding I/O driver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Long (i32)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>stack</th>
<th>Stack size [bytes]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10240</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>filename</th>
<th>Name of the file with the .md1 extension which contains the task algorithm; in the case filename is not specified, the filename is given by the name of the IOTASK block in the project main file (the .md1 extension is attached automatically)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>String</td>
</tr>
</tbody>
</table>
LPBRK – Loop break

Block Symbol

Licence: STANDARD

Function Description

The LPBRK block is an auxiliary block often used in the control schemes consisting of the REXYGEN system function blocks. The block is usually placed in all feedback loops in the scheme. Its behavior differs in the REXYGEN system and the Simulink system.

The LPBRK block creates a one-sample delay in the Simulink system. If there exists a feedback loop without the LPBRK block, the Simulink system detects an algebraic loop and issues a warning (Matlab version 6.1 and above). The simulation fails after some time.

The REXYGEN Compiler compiler omits the LPBRK block, the only effect of this block is the breaking of the feedback loop at the block’s position. If there exists a loop without the LPBRK block, the REXYGEN Compiler compiler issues a warning and breaks the loop at an automatically determined position. It is recommended to use the LPBRK block in all loops to achieve the maximum compatibility between the REXYGEN system and the Simulink system.

Input

\[ u \quad \text{Input signal} \quad \text{Double (F64)} \]

Output

\[ y \quad \text{Output signal} \quad \text{Double (F64)} \]
MODULE – Extension module of the REXYGEN system

Block Symbol

Licence: STANDARD

Function Description

The REXYGEN system has an open architecture thus its functionality can be extended. Such extension is provided by modules. Each module is identified by its name placed below the block symbol. The individual modules are added to the project main file by connecting the prev input with the Modules output of the EXEC block or with the next output of a MODULE which is already included in the project. There can be only one module connected to the next output of the MODULE block. The next output of the last module in the project remains unconnected. This means that the modules create a unidirectional chain which defines the order of initialization of individual modules.

Input

<table>
<thead>
<tr>
<th>prev</th>
<th>Input for connecting the module with the Modules output of the EXEC block or with the next output of the preceding module</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Long (132)</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>next</th>
<th>Output for connecting to the prev input of the succeeding module</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Long (132)</td>
</tr>
</tbody>
</table>
CHAPTER 2. EXEC – REAL-TIME EXECUTIVE CONFIGURATION

PROJECT – Additional Project Settings

Block Symbol

Licence: STANDARD

Function Description

The PROJECT block is a so-called "pseudo-block" which stores additional settings and parameters related to a project and a real-time executive. The only file where the block can be placed is a main project file with a single EXEC block and so it belongs to the EXEC category.

The block does not have any inputs or outputs. The block does not become a part of the final binary configuration.

Parameters

- CompileParams: Command-line options which are passed to REXYGEN. String
- Compiler: during project compilation.
- SourcesOnTarget: Store source files on target device. on: Bool
- TargetURL: URL address of a target on which the configuration should be run. The address is inserted into all connection dialogs automatically. String
- LibraryPath: Path to libraries referenced in the project. Can be absolute or relative to project folder. String
QTASK – Quick task of the REXYGEN system

Block Symbol  

Function Description

The QTASK block is used for including the so-called quick task with high priority into the executive of the REXYGEN system. This task is used where the fastest processing of the input signals is necessary, e.g. digital filtering of input signals corrupted with noise or immediate processing of switches connected via digital inputs. The quick task is added into the configuration by connecting the prev input with the EXEC block’s QTASK output. The quick task is initialized before the initialization of the Level0 computation level (see the TASK block).

There can be only one QTASK block in the REXYGEN control system. It runs with the logical priority no. 2. The algorithm of the quick task is configured the same way as the standard TASK, it is a separate .md1 file.

The execution period of the task is given by a multiple of the factor parameter and the tick of the EXEC block. The task is executed with the shortest period of tick seconds for factor=1. In that case the system load is the highest. Under all circumstances the QTASK must be executed within tick seconds, otherwise a real-time executive fatal error occurs and no other tasks are executed. Therefore the QTASK block must be used with consideration. The execution time of the block is displayed in the REXYGEN Diagnostics diagnostic program.

Input

prev  Input for connecting the task with the QTask output of the EXEC block  Long (132)

Parameters

factor  Multiple of the EXEC block’s tick parameter defining the quick task execution period  Long (132)

stack  Stack size [bytes]  01 10240  Long (132)

filename  Name of the file with the .md1 extension which contains the quick task algorithm; in the case filename is not specified, the filename is given by the name of the QTASK block in the project main file (the .md1 extension is attached automatically)  String
SLEEP – Timing in Simulink

Block Symbol

Licence: STANDARD

Function Description

The Matlab/Simulink system works natively in simulation time, which can run faster or slower than real time, depending on the complexity of the algorithm and the computing power available. Therefore the SLEEP block must be used when accurate timing and execution of the algorithm in the Matlab/Simulink system is required. In the REXYGEN system, timing and execution is provided by system resources (see the EXEC block) and the SLEEP block is ignored.

In order to perform real-time simulation of the algorithm, the SLEEP block must be included. It guarantees that the algorithm is executed with the period given by the ts parameter unless the execution time is longer than the requested period.

The SLEEP block is implemented for Matlab/Simulink running in Microsoft Windows operating system. It is recommended to use periods of 100 ms and above. For the proper functionality the 'Solver type' must be set to fixed-step and discrete (no continuous states) in the 'Solver' tab of the 'Simulation parameters' dialog. Further the Fixed step size parameter must be equal to the ts parameter of the SLEEP block. There should be at most one SLEEP block in the whole simulation scheme (including all subsystems).

Parameter

\( ts \)  Simulation scheme execution period (in seconds)  \( 0.1 \)  Double (F64)
SRTF – Set run-time flags

Function Description

The SRTF block (Set Run-Time Flags) can be used to influence the execution of tasks, subsystems (sequences) and blocks of the REXYGEN system. This block is not meant for use in Matlab-Simulink. When describing this block, the term object refers to a REXYGEN system object running in real-time, i.e. input/output driver, one of the tasks, subsystem or a simple function block of the REXYGEN system.

All the operations described below affect the object, whose full path is given by the name parameter. Should the parameter be left blank (empty string), the operation applies to the nearest owner of the SRTF object, i.e. the subsystem in which the block is directly included or the task containing the block.

The run-time flags allow the following operations:

- **Disable execution** of the object by setting the EXDIS input to on. The execution can be enabled again by using the input signal EXDIS = off. The EXDIS input sets the same run-time flag as the Halt/Run button in the upper right corner of the Workspace tab in the REXYGEN Diagnostics diagnostic program.

- **One-shot execution** of the object. If the object execution is disabled by the EXDIS = on input or by the REXYGEN Diagnostics program, it is possible to trigger one-shot execution by EXOSH = on.

- **Enable diagnostics** for the given object by DGEN = on. The result is equivalent to ticking the Enable checkbox in the diagnostic pane of the corresponding tab (I/O Driver, Level, Quick Task, Task, I/O Task, Sequence) in the REXYGEN Diagnostics program.

- **Reset diagnostic data** of the given object by DGRES = on. The same flag can be set by the Reset button in the diagnostic pane of the corresponding tab in the REXYGEN Diagnostics program. The flag is automatically set back to 0 when the data reset is performed.

The following table shows the flags available for various objects in the REXYGEN system.
CHAPTER 2. EXEC – REAL-TIME EXECUTIVE CONFIGURATION

<table>
<thead>
<tr>
<th>Object type</th>
<th>EXDIS</th>
<th>EXOSH</th>
<th>DGEN</th>
<th>DGRES</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O Driver</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Level</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Task</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Quick Task</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>I/O Task</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sequence, subsystem</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Block</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

**Inputs**

- **EXDIS**: Disable execution  
  - **Value**: Bool
- **EXOSH**: One-shot execution  
  - **Value**: Bool
- **DGEN**: Enable diagnostics  
  - **Value**: Bool
- **DGRES**: Reset diagnostic data  
  - **Value**: Bool
- **DLOG**: Enable more verbose logging  
  - **Value**: Bool

**Outputs**

- **E**: Error flag  
  - **Value**: Bool
  - **off**: No error  
  - **on**: An error occurred
- **iE**: Error code (for E = on)  
  - **Value**: Long (I32)
  - **0**: No error
  - **1**: The object specified by the *bname* parameter was not found
  - **2**: REXGEN system internal error (invalid pointers)
  - **3**: Flag could not be set (timeout)

**Parameter**

- **bname**: Full path to the block/object. Case sensitive. Individual layers are separated by dots, the object names excluding tasks (TASK, QTASK) start with the following special characters:
  - **^**: Computational level, e.g. ^0 for Level10
  - **&**: Input/Output Driver, e.g. &WcmDrv

Name of the task triggered by input/output driver (IOTASK) has the form &<driver_name>.<task_name>. 
OSCALL – Operating system calls

Block Symbol

Function Description

The OSCALL block is intended for executing operating system functions from within the REXYGEN system. The chosen action is performed upon a rising edge (off→on) at the TRG input. However, not all actions are supported on individual platforms. The result of the operation and the possible error code are displayed by the E and iE outputs.

Note that there is also the EPC block available, which allows execution of external programs.

Input

| TRG | Trigger of the selected action | Bool |

Outputs

| E   | Error flag   | Bool |
| iE  | Error code  | Long (I32) |
|     | i .... REXYGEN general error |

Parameter

<table>
<thead>
<tr>
<th>action</th>
<th>System function to perform</th>
<th>⊙1</th>
<th>Long (I32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reboot system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>System shutdown</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>System halt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Flush disc caches</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Lock system partition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Unlock system partition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Disable internal webserver</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Enable internal webserver</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TASK – Standard task of the REXYGEN system

Block Symbol

Licence: STANDARD

Function Description

The overall control algorithm of the REXYGEN system consists of individual tasks. These are included by using the TASK block. There can be one or more tasks in the control algorithm. The REXYGEN system contains four main computational levels represented by the Level10 to Level13 outputs of the EXEC block. The individual tasks are added to the given computational level \(<i>\) by connecting the prev input with the corresponding Level\(<i>\) output or with the next output of a TASK, which is already included in the given level \(<i>\). There can be only one task connected to the next output of the TASK block. The next output of the last task in the given level remains unconnected. This means that the tasks in one level create a unidirectional chain which defines the order of initialization and execution of the individual tasks of the given level in the REXYGEN system. The individual levels are ordered from Level10 to Level13 (the QTASK block precedes Level10).

All the tasks of the given level \(<i>\) are executed with the same priority given by the pri\(<i>\) parameter of the EXEC block. The execution period of the task is given by a multiple of the factor parameter and the base tick of the given level \(<i>\) \(ntick< i > \ast tick\) in the EXEC block. The time allocated for the task to execute starts at the start tick and ends at the stop tick, where the inequality \(0 \leq start < stop \leq ntick< i >\) must hold for the start and stop parameters. The REXYGEN Compiler compiler further checks whether the stop parameter of the preceding task is less or equal to the stop parameter of the succeeding task, i.e. the allocated time intervals for individual tasks cannot overlap. In the case the timing of individual levels is inappropriate, the tasks are interrupted by tasks and other events with higher priority and might not execute in the allocated time. In such a case the execution is not aborted but delayed (in contrary to the QTASK block).

The REXYGEN Diagnostics program diagnoses whether the execution delay is occasional or permanent (the Level and Task tabs).

Input

prev Input for connecting the task with the corresponding Level\(<i>\) Long (I32)

output of the EXEC block or with the next output of the preceding task of the given level

Output

next Output for connecting to the prev input of the succeeding task Long (I32)
in the given level
Parameters

factor  Execution factor; multiple of the execution period of the i-th level of the EXEC block defining the execution period of the task: \( \text{factor} \times \text{tick} \times \text{tick} <i> \)  
\( \text{factor} \times \text{tick} \times \text{tick} <i> \)  
\( \text{Long} \ (I32) \)

start  Number of tick of the given computational level which should trigger the task execution  
\( \downarrow 0 \times \text{tick} <i> \)  
\( \text{Long} \ (I32) \)

stop  Number of tick of the given computational level by which the task execution should finish  
\( \downarrow \text{start} + 1 \times \text{tick} <i> \)  
\( \text{Long} \ (I32) \)

stack  Stack size [bytes]  
\( \text{0} \times 10240 \)  
\( \text{Long} \ (I32) \)

filename  Name of the file with the \text{.md1} extension which contains the task algorithm. In the case \text{filename} is not specified, the filename is given by the name of the TASK block in the project main file (the \text{.md1} extension is attached automatically)  
\( \text{String} \)
TIODRV — The REXYGEN system input/output driver with tasks

Block Symbol

Function Description

The TIODRV block is used for configuration of special drivers of the REXYGEN system which are able to execute tasks defined by the IOTASK blocks. See the corresponding driver documentation.

The \texttt{prev} input of the IOTASK block must be connected with the Tasks output of the TIODRV block. If the driver allows so, the \texttt{next} output of a TIODRV block which is already included in the configuration can be used to add more tasks. The \texttt{next} output of the last task remains unconnected. On the contrary to standard tasks, the number and order of the driver's tasks are not checked by the REXYGEN Compiler compiler but by the input-output driver itself.

If the driver cannot guarantee periodic execution of some task (e.g. task is triggered by an external event), a corresponding flag is set for the given task. Such a task cannot contain blocks which require constant sampling period (e.g. the majority of controllers). If some of these restricted blocks are used, the executive issues a task execution error, which can be traced using the REXYGEN Diagnostics program.

Input

\begin{tabular}{ll}
\texttt{prev} & Input for connecting the driver with the Drivers output of the \texttt{EXEC} block or with the \texttt{next} output of the preceding driver \hfill \texttt{Long (I32)} \\
\end{tabular}

Outputs

\begin{tabular}{ll}
\texttt{next} & Output for connecting to the \texttt{prev} input of the succeeding driver \hfill \texttt{Long (I32)} \\
\texttt{Tasks} & The IOTASK blocks executed by the driver are connected to this output using the \texttt{prev} input \hfill \texttt{Long (I32)} \\
\end{tabular}

Parameters

\begin{tabular}{ll}
\texttt{module} & Name of the module, which includes the input/output driver \hfill \texttt{String} \\
& (mandatory only if module name differs from \texttt{classname}) \\
\texttt{classname} & Name of the driver class; case sensitive! \hfill \texttt{DrvClass \ String} \\
\texttt{cfgname} & Name of the driver configuration file \hfill \texttt{iodrv.rio \ String} \\
\end{tabular}
| **factor** | Multiple of the **EXEC** block’s tick parameter defining the driver’s task execution period | Long (I32) \( \downarrow 1 \odot 10 \) |
| **stack** | Stack size of the driver’s task in bytes | Long (I32) \( \downarrow 1024 \odot 10240 \) |
| **pri** | Logical priority of the driver’s task | Long (I32) \( \downarrow 1 \uparrow 31 \odot 3 \) |
| **timer** | Driver is a source of time | Bool |
WWW – Internal Web Server Content

Block Symbol

Licence: STANDARD

Function Description

The WWW block is a so-called "pseudo-block" which stores additional information about a contents of an internal web server. The only file where the block can be placed is a main project file with a single EXEC block an so it belongs to the EXEC category.

The block does not have any inputs or outputs. The block itself does not become a part of a final binary configuration but the data it points to does. Be careful when inserting big files or directories as the integrated web server is not optimized for a large data. It is possible to shrink the data by enabling gzip compression. The compression also reduces amount of data transferred to the client, but decompression must be performed on the server side when a client does not support gzip compression which brings additional load on the target device.

Parameters

Source  Specifies a source directory or a file name that should be placed on the target and should be available via integrated web server using standard HTTP and/or HTTPS protocol. The path may be absolute or relative to path of a main project file.  String

Target  Specifies a target directory or a file name on the integrated web server.  String

Compression  Enables data compression in gzip format.  Bool
Chapter 3

INOUT – Input and output blocks

Contents

Display – Numeric display of input values .................. 44
From, INSTD – Signal connection or input .................. 45
Goto, OUTSTD – Signal source or output .................. 47
GotoTagVisibility – Visibility of the signal source .......... 49
Inport, Outport – Input and output port .................. 50
SubSystem – Subsystem block ................................. 52
INQUAD, INOCT, INHEXD – Multi-input blocks ............. 54
OUTQUAD, OUTOCT, OUTHEXD – Multi-output blocks ...... 55
OUTRQUAD, OUTROCT, OUTRHEXD – Multi-output blocks with verification 57
OUTRSTD – Output block with verification .................. 58
QFC – Quality flags coding ................................. 59
QFD – Quality flags decoding ............................... 60
VIN – Validation of the input signal ....................... 61
VOUT – Validation of the output signal .................... 62
Display – Numeric display of input values

Block Symbol Licence: STANDARD

Function Description
The DISPLAY block shows input value in a selected format. A suffix may be appended to the value. An actual value is shown immediately in REXYGEN Studio even without turning on Watch mode for the block, and the same in WebWatch. Actual conversion of input into its textual representation is performed on the target device in each Decimation period so the value displayed may be also read via the REST interface or used in visualization.

Input
u Input signal Unknown

Parameters

<table>
<thead>
<tr>
<th>Format</th>
<th>Format of displayed value</th>
<th>Value is evaluated in each Decimation period</th>
<th>A string to be appended to the value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best fit</td>
<td>Long (I32)</td>
<td>[1, 100000]</td>
<td>String</td>
</tr>
<tr>
<td>short</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>long ..</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>short_e</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>long_e</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bank ..</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hex ...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bin ...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>det ...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DispValue
From, INSTD – Signal connection or input

Block Symbols

![Diagram of block symbols with DRV_signal]

Licence: STANDARD

Function Description

The two blocks From (signal connection) and INSTD (standard input) share the same symbol. They are used for referring to another signal, either internal or external.

In the function block library, you can only find the From block. It is converted to the INSTD block at compile time if necessary. The following rules define how the REXYGEN Compiler compiler distinguishes between the two block types:

- If the parameter GotoTag contains the __ delimiter (two successive '_' characters), then the block is of the INSTD type. The part (substring) of the parameter before the delimiter (DRV in the block symbol above) is considered to be the name of an IODRV type block contained in the main file of the project. The REXYGEN Compiler compiler returns an error when such block does not exist. If the driver exists in the project, the other part of the GotoTag parameter (following the delimiter, signal in this case) is considered to be the name of a signal within the corresponding driver. This name is validated by the driver and in the case of success, an instance of the INSTD block is created. This instance collects real-time data from the driver and feeds the data into the control algorithm at each execution of the task it is included in.

- If there is no __ delimiter in the GotoTag parameter, the block is of type From. A matching Goto block with the same GotoTag parameter and required visibility given by the TagVisibility parameter (see the Goto block description) is searched. In case it is not found, the REXYGEN Compiler compiler issues a warning and deletes the From block. Otherwise an "invisible" connection is created between the corresponding blocks. The From block is removed also in this case and thus it is not contained in the resulting control system configuration.

In the case of INSTD block, the GotoTag parameter includes the symbol of the driver <DRV> and the name of the signal <signal> of the given driver:

<DRV>_<signal>

E.g. the first digital input of a Modbus I/O device might be referenced by MBM_DI1. Detailed information about signal naming can be found in the user manual of the corresponding I/O driver.
CHAPTER 3. INOUT - INPUT AND OUTPUT BLOCKS

Since version 2.50.5 it is possible to use placeholders in names of I/O driver signals. This is useful inside subsystems where this placeholder is replaced by the value of subsystem parameter. E.g. the flag `MBM_DI{id}` will refer to digital input 1, 2, 3 etc. depending on the parameter `id` of the subsystem the block is contained in. See the `Subsystem` function block for information on defining subsystem parameters.

Output

`value` Signal coming from I/O driver or `Goto` block. The type of output is determined by the type of the signal which is being referred by the `GotoTag` parameter.

Parameter

`GotoTag` Reference to a `Goto` block with the same `GotoTag` parameter, which should be connected with the `From` block or a reference to input signal of the `REXYGEN` I/O driver, which should provide data through the block’s output.
Goto, OUTSTD – Signal source or output

Block Symbols

Function Description

The two blocks Goto (signal source) and OUTSTD (standard output) share the same symbol. They are used for providing signals, either internal or external.

In the function block library, you can only find the Goto block. It is converted to the OUTSTD block at compile time if necessary. The following rules define how the REXYGEN Compiler compiler distinguishes between the two block types:

- If the parameter GotoTag contains the ___ delimiter (two successive '_ ' characters), then the block is of the OUTSTD type. The part (substring) of the parameter before the delimiter (DRV in the block symbol above) is considered to be the name of an IDRV type block contained in the main file of the project. The REXYGEN Compiler returns an error when such a block does not exist. If the driver exists in the project, the other part of the GotoTag parameter (following the delimiter, signal in this case) is considered to be the name of a signal within the appropriate driver. This name is validated by the driver and in the case of success, an instance of the OUTSTD block is created. This instance collects real-time data from the driver and feeds the data into the control algorithm at each execution of the task it is included in.

- If there is no ___ delimiter in the GotoTag parameter, the block is of type Goto. A matching From block with the same GotoTag parameter for which the Goto block is visible is searched. In case it is not found, the REXYGEN Compiler issues a warning and deletes the Goto block. Otherwise an "invisible" connection is created between the corresponding blocks. The Goto block is removed also in this case thus it is not contained in the resulting control system configuration.

The other parameter of the Goto block defines the visibility of the block within the given .mdl file. The TagVisibility parameter can be local, global or scoped, whose meaning is explained in the table below. This parameter is ignored if the block is compiled as the OUTSTD block.

In the case of OUTSTD block, the GotoTag parameter includes the symbol of the driver <DRV> and the name of the signal <signal> of the given driver:

<DRV>__<signal>
CHAPTER 3. INOUT – INPUT AND OUTPUT BLOCKS

E.g. the first digital output of a Modbus I/O device might be referenced by MBM_DO1.
Detailed information about signal naming can be found in the user manual of the corresponding I/O driver.

Since version 2.50.5 it is possible to use placeholders in names of I/O driver signals. This is useful inside subsystems where this placeholder is replaced by the value of subsystem parameter. E.g. the flag MBM_DO<i> will refer to digital output 1, 2, 3 etc. depending on the parameter id of the subsystem the block is contained in. See the SubSystem function block for information on defining subsystem parameters.

**Input**

<table>
<thead>
<tr>
<th>value</th>
<th>Signal going to I/O driver or From block. In case of connection to an I/O driver, the type of input is determined by the I/O driver from the GotoTag parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unknown</td>
<td></td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>GotoTag</th>
<th>Reference to a From block with the same GotoTag parameter, which should be connected with the Goto block or a reference to output signal of the REXYGEN control system driver, which should send the data from block input to the process.</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TagVisibility</th>
<th>Visibility (availability) of the block within the .md1 file. Defines conditions under which the two corresponding Goto and From blocks are reciprocally available:</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>the two blocks must be in the same subsystem</td>
</tr>
<tr>
<td>global</td>
<td>blocks can be anywhere in the given task (.md1 file)</td>
</tr>
<tr>
<td>scoped</td>
<td>the From block must be placed in the same subsystem or in any lower hierarchical level below the GotoTagVisibility block with the same GotoTag parameter</td>
</tr>
<tr>
<td>String</td>
<td></td>
</tr>
<tr>
<td>○local</td>
<td></td>
</tr>
</tbody>
</table>
GotoTagVisibility – Visibility of the signal source

Block Symbol

Licence: STANDARD

Function Description

The GotoTagVisibility blocks specify the visibility of the Goto blocks with scoped visibility. The symbol (tag) defined in the Goto block by the GotoTag parameter is available for all From blocks in the subsystem which contains the appropriate GotoTagVisibility block and also in all subsystems below in the hierarchy.

The GotoTagVisibility block is required only for Goto blocks whose TagVisibility parameter is set to scoped. There is no need for the GotoTagVisibility block for local or global visibility.

The GotoTagVisibility block is used only during project compilation by the REXY-GEN Compiler compiler. It is not included in the binary configuration file for real-time execution.

Parameter

GotoTag Reference to a Goto block with the GotoTag parameter, String whose visibility is defined by the position of this block (GotoTagVisibility)
Inport, Outport – Input and output port

Block Symbols

Function Description

The Inport and Outport blocks are used for connecting signals over individual hierarchical levels. There are two possible ways to use these blocks in the REXYGEN system:

1. To connect inputs and outputs of the subsystem. The blocks create an interface between the symbol of the subsystem and its inner algorithm (sequence of blocks contained in the subsystem). The Inport or Outport blocks are located inside the subsystem, the name of the given port is displayed in the subsystem symbol in the upper hierarchy level.

2. To provide connection between various tasks. The port blocks are located in the highest hierarchy level of the given task (.mil file) in this case. The connection of Inport and Outport blocks in various tasks is checked and created by the REXYGEN Compiler compiler.

The ordering of the blocks to be connected is based on the Port parameter of the given block. The numberings of the input and output ports are independent on each other. The numbering is automatic in REXYGEN Studio and it starts at 1. The numbers of ports must be unique in the given hierarchy level, in case of manual modification of the port number the other ports are re-numbered automatically. Be aware that after re-numbering in an already connected subsystem the inputs (or outputs) in the upper hierarchy level are re-ordered, which results in probably unintended change in signal mapping!

Input

| value | Value going to the output pin or Inport | Unknown |

Output

| value | Value coming from the input pin or Outport | Unknown |

Parameters

| Port | Ordering of the Inport or Outport pins | Long (I32) |
GetDataTypeStr  Data type of item
    Inherit: auto
double
single
uint8
int16
uint16
int32
uint32
bool
float
int64
string
array

String
SubSystem – Subsystem block

Block Symbol

Licence: STANDARD

Function Description

The SubSystem block is a cornerstone of hierarchical organization of block diagrams in REXYGEN. A subsystem is a container for a group of function blocks and their connections, which then appear as a single block. Nesting of subsystems is allowed, i.e. a subsystem can include additional subsystems.

The runtime core or REXYGEN executes the subsystem as an ordered sequence of blocks. Therefore the subsystem is sometimes referred to as sequence. All blocks from the surroundings of the subsystem are executed strictly before or strictly after the whole subsystem is executed.

Subsystems are also used for creating user-defined reusable components, which are then placed in user libraries.

A library reference can be distinguished from a standard subsystem by the style of the upper border.

Please refer to [2] for details on using subsystems and creating reusable components in REXYGEN.

Also see examples 0101-02 and 0101-03 demonstrating the use of subsystems. The examples are included in REXYGEN Studio.

Inputs

The ordering and names of the inputs are given by the numbers and names of the Import blocks contained within the subsystem. See REXYGEN Studio manual [2] for details.

Outputs
The ordering and names of the outputs are given by the numbers and names of the Output blocks contained within the subsystem. See REXYGEN Studio manual [2] for details.

Parameters

The parameters of the subsystem are defined by the so-called subsystem mask. See REXYGEN Studio manual [2] for details.
INQUAD, INOCT, INHEXD – Multi-input blocks

Block Symbols

Licence: STANDARD

Function Description

The REXYGEN system allows not only reading of a single input signal but also simultaneous reading of multiple signals through just one block (for example all signals from one module or plug-in board). The blocks INQUAD, INOCT and INHEXD are designed for these purposes. They differ only in the maximum number of signals (4, 8 and 16, respectively).

The name of the block instance includes the symbol of the driver <DRV> and the name of the signal <signal> of the given driver:

<DRV>_<signal>

It is created the same way as the GotoTag parameter of the INSTD and OUTSTD blocks. E.g. the digital inputs of a Modbus I/O device might be referenced by MEM__DI. Detailed information about signal naming can be found in the user manual of the corresponding I/O driver.

The overhead necessary for data acquisition through input/output drivers is minimized when using these blocks, which is important mainly for very fast control algorithms with sampling period of 1 ms and lower. Moreover, all the inputs are read simultaneously or as successively as possible. Detailed information about using these blocks for particular driver can be found in the user manual for the given driver.

Since version 2.50.5 it is possible to use placeholders in names of I/O driver signals. This is useful inside subsystems where this placeholder is replaced by the value of subsystem parameter. E.g. the name MEM_module<id> will refer to module 1, 2, 3 etc. depending on the parameter id of the subsystem the block is contained in. See the SubSystem function block for information on defining subsystem parameters.

Outputs

vali: Input signals fed into the control algorithm through Unknown input/output drivers. The type and location of individual signals is described in the user manual for the given driver.
OUTQUAD, OUTOCT, OUTHEXD – Multi-output blocks

Block Symbols

Licence: STANDARD

Function Description

The REXYGEN system allows not only writing of a single output signal but also simultaneous writing of multiple signals through just one block (for example all signals of one module or plug-in board). The blocks OUTQUAD, OUTOCT and OUTHEXD are designed for these purposes. They differ only in the maximum number of signals (4, 8 and 16, respectively). These blocks are not included in the RexLib function block library for Matlab-Simulink.

The name of the block instance includes the symbol of the driver <DRV> and the name of the signal <signal> of the given driver:

<DRV>_<signal>

It is created the same way as the GotoTag parameter of the INSTD and OUTSTD blocks. E.g. the digital outputs of a Modbus I/O device might be referenced by MBM__DO. Detailed information about signal naming can be found in the user manual of the corresponding I/O driver.

The overhead necessary for setting the outputs through input/output drivers is minimized when using these blocks, which is important mainly for very fast control algorithms with sampling period of 1 ms and lower. Moreover, all the inputs are written simultaneously or as successively as possible. Detailed information about using these blocks for particular driver can be found in the user manual for the given driver.

Since version 2.50.5 it is possible to use placeholders in names of I/O driver signals. This is useful inside subsystems where this placeholder is replaced by the value of subsystem parameter. E.g. the name MBM__module<id> will refer to signals of module 1, 2, 3 etc. depending on the parameter id of the subsystem the block is contained in. See the SubSystem function block for information on defining subsystem parameters.
CHAPTER 3. INOUT — INPUT AND OUTPUT BLOCKS

Inputs

val:  Signals to be sent to the process via the input/output driver.  Unknown
The type and location of individual signals is described in the user manual for the given driver.
OUTRQUAD, OUTROCT, OUTRHEXD – Multi-output blocks with verification

Block Symbols

License: ADVANCED

Function Description

The OUTRQUAD, OUTROCT, and OUTRHEXD blocks allow simultaneous writing of multiple signals, they are similar to the OUTQUAD, OUTOCT, and OUTHEXD blocks. Additionally they provide feedback information about the result of write operation for the given output.

There are two ways to inform the control algorithm about the result of write operation through the raw output:

- Through the value of the output, which can e.g. contain the real bit value in case of exceeding the limits of D/A converter (thus the raw notation).

- Through reading the quality flags of the signal. This information can be separated from the signal by the VIN and QFD blocks.

The raw outputs are not always refreshed right at the moment of block execution, there is some delay given by the properties of the driver, communication line and/or target platform.

Inputs

vali Output signals defined by the control algorithm through the Unknown input/output driver. The type and location of individual signals is described in the user manual for the given driver.

Outputs

rawi Feedback information about the write operation result. The type and meaning of individual signals is described in the user manual for the given driver.
OUTRSTD – Output block with verification

Block Symbol

Function Description

The OUTRSTD block is similar to the OUTSTD block. Additionally, it provides feedback information about the result of write operation for the output signal.

There are two ways to inform the control algorithm about the result of write operation through the raw output:

- Through the value of the output, which can e.g. contain the real bit value in case of exceeding the limits of D/A converter (thus the raw notation).
- Through reading the quality flags of the signal. This information can be separated from the signal by the VIN and QFD blocks.

The raw outputs is not refreshed right at the moment of block execution, there is some delay given by the properties of the driver, communication line and/or target platform.

Input

value  Output signal defined by the control algorithm through the input/output driver. The type and naming of the signal is described in the user manual for the given driver.

Output

raw  Feedback information about the write operation result. The type and meaning of the signal is described in the user manual for the given driver.
QFC – Quality flags coding

Block Symbol

Function Description

The QFC block creates the resulting signal iqf representing the quality flags by combining three components iq, is and il. The quality flags are part of each input or output signal in the REXYGEN system. Further details about quality flags can be found in chapter 1.4 of this manual. The RexLib function block library for Matlab-Simulink does not use any quality flags.

It is possible to use the QFC block together with the VOUT block to force arbitrary quality flags for a given signal. Reversed function to the QFC block is performed by the QFD block.

Inputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>iq</td>
<td>Basic quality type flags, see table 1.2, page 17</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>is</td>
<td>Substatus flags, see [1]</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>il</td>
<td>Limits flags, see [1]</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>iqf</td>
<td>Bit combination of the iq, is and il input signals</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>
QFD – Quality flags decoding

Block Symbol

Function Description

The QFD decomposes quality flags to individual components iq, is and il. The quality flags are part of each input or output signal in the REXYGEN system. Further details about quality flags can be found in chapter 1.4 of this manual. The RexLib function block library for Matlab-Simulink does not use any quality flags.

It is possible to use the QFD block together with the VIN block for detailed processing of quality flags of a given signal. Reversed function to the QFD block is performed by the QFC block.

Input

iqf  Quality flags to be decomposed to iq, is and il components  Long (I32)

Outputs

iq  Basic quality type flags, see table 1.2, page 17  Long (I32)
is  Substatus flags, see [1]  Long (I32)
il  Limits flags, see [1]  Long (I32)
VIN – Validation of the input signal

Block Symbol

Function Description

The VIN block can be used for verification of the input signal quality in the REXYGEN system. Further details about quality flags can be found in chapter 1.4 of this manual.

The block continuously separates the quality flags from the input u and feeds them to the iqf output. Based on these quality flags and the GU parameter (Good if Uncertain), the input signals are processed in the following manner:

- For GU = off the output QG is set to on if the quality is GOOD. It is set to QG = off in case of BAD or UNCERTAIN quality.
- For GU = on the output QG is set to on if the quality is GOOD or UNCERTAIN. It is set to QG = off only in case of BAD quality.

The output yg is equal to the u input if QG = on. Otherwise it is set to yg = sv (substitution variable).

Inputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>Input signal whose quality is assessed. The type of the signal is determined upon the connected signal.</td>
</tr>
<tr>
<td>sv</td>
<td>Substitute value for an error case</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yg</td>
<td>Validated output signal (yg = u for QG = on or yg = sv for QG = off)</td>
</tr>
<tr>
<td>QG</td>
<td>Indicator of input signal acceptability</td>
</tr>
<tr>
<td>iqf</td>
<td>Complete quality flag separated from the u input signal</td>
</tr>
</tbody>
</table>

Parameter

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GU</td>
<td>Acceptability of UNCERTAIN quality</td>
</tr>
<tr>
<td>off</td>
<td>Uncertain quality unacceptable</td>
</tr>
<tr>
<td>on</td>
<td>Uncertain quality acceptable</td>
</tr>
</tbody>
</table>

Licence: ADVANCED
VOUT – Validation of the output signal

Block Symbol

Licence: ADVANCED

Function Description

It is possible to use the VOUT block to force arbitrary quality flags for a given signal. The desired quality flags are given by the input signal iqf. Further details about quality flags can be found in chapter 1.4 of this manual.

Inputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>Input signal whose quality flags are being replaced. The type of the signal is determined upon the connected signal.</td>
</tr>
<tr>
<td>iqf</td>
<td>Desired quality flags</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yq</td>
<td>Resulting signal composed from input u and quality flags given by the iqf input</td>
</tr>
</tbody>
</table>

Unknown

Long (I32)
Chapter 4

MATH – Math blocks

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS, – Absolute value</td>
<td>65</td>
</tr>
<tr>
<td>ADD – Addition of two signals</td>
<td>66</td>
</tr>
<tr>
<td>ADDQUAD, ADDOCT, ADDHEXD – Multi-input addition</td>
<td>67</td>
</tr>
<tr>
<td>CIN – Boolean (logic) constant</td>
<td>68</td>
</tr>
<tr>
<td>CNE – Enumeration constant</td>
<td>69</td>
</tr>
<tr>
<td>CNI – Integer constant</td>
<td>70</td>
</tr>
<tr>
<td>CINR – Real constant</td>
<td>71</td>
</tr>
<tr>
<td>DIF, – Difference</td>
<td>72</td>
</tr>
<tr>
<td>DIV – Division of two signals</td>
<td>73</td>
</tr>
<tr>
<td>EAS – Extended addition and subtraction</td>
<td>74</td>
</tr>
<tr>
<td>EMD – Extended multiplication and division</td>
<td>75</td>
</tr>
<tr>
<td>FNX – Evaluation of single-variable function</td>
<td>76</td>
</tr>
<tr>
<td>FNXY – Evaluation of two-variables function</td>
<td>78</td>
</tr>
<tr>
<td>GAIN – Multiplication by a constant</td>
<td>80</td>
</tr>
<tr>
<td>GRADS – Gradient search optimization</td>
<td>81</td>
</tr>
<tr>
<td>IADD – Integer addition</td>
<td>83</td>
</tr>
<tr>
<td>ISUB – Integer subtraction</td>
<td>85</td>
</tr>
<tr>
<td>IMUL – Integer multiplication</td>
<td>86</td>
</tr>
<tr>
<td>IDIV – Integer division</td>
<td>88</td>
</tr>
<tr>
<td>IMOD – Remainder after integer division</td>
<td>89</td>
</tr>
<tr>
<td>LIN – Linear interpolation</td>
<td>90</td>
</tr>
<tr>
<td>MUL – Multiplication of two signals</td>
<td>91</td>
</tr>
<tr>
<td>POL – Polynomial evaluation</td>
<td>92</td>
</tr>
<tr>
<td>REC – Reciprocal value</td>
<td>93</td>
</tr>
<tr>
<td>REL – Relational operator</td>
<td>94</td>
</tr>
<tr>
<td>RTOI – Real to integer number conversion</td>
<td>95</td>
</tr>
</tbody>
</table>
SQR – Square value ................................................. 96
SQR\_ – Square root ............................................... 97
SUB – Subtraction of two signals .............................. 98
**ABS** – Absolute value

Block Symbol

License: STANDARD

Function Description

The **ABS** block computes the absolute value of the analog input signal $u$. The output $y$ is equal to the absolute value of the input and the $\text{sgn}$ output denotes the sign of the input signal.

\[
\text{sgn} = \begin{cases} 
-1, & \text{for } u < 0, \\
0, & \text{for } u = 0, \\
1, & \text{for } u > 0.
\end{cases}
\]

Input

$u$ 
Analog input of the block 
Double (F64)

Outputs

$y$ 
Absolute value of the input signal 
Double (F64)

$\text{sgn}$ 
Indication of the input signal sign 
Long (I32)
ADD – Addition of two signals

Block Symbol

Licence: STANDARD

Function Description
The ADD block sums two analog input signals. The output is given by
\[ y = u_1 + u_2. \]

Consider using the ADDICT block for addition or subtraction of multiple signals.

Inputs

<table>
<thead>
<tr>
<th>u1</th>
<th>First analog input of the block</th>
<th>Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>u2</td>
<td>Second analog input of the block</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

Output

| y    | Sum of the input signals        | Double (F64) |
ADDQUAD, ADDOCT, ADDHEXD – Multi-input addition

Block Symbols

 Licence: STANDARD

Function Description

The ADDQUAD, ADDOCT and ADDHEXD blocks sum (or subtract) up to 16 input signals. The
n1 parameter defines the inputs which are subtracted instead of adding. For an empty n1
parameter the block output is given by y = u1 + u2 + u3 + u4 + u5 + u6 + u7 + ... + u16.
For e.g. n1=2,5,7, the block implements the function y = u1 − u2 + u3 + u4 − u5 + u6 −
u7 + ... + u16.

Note that the ADD and SUB blocks are available for simple addition and subtraction
operations.

Inputs

u1..u16 Analog input signals Double (F64)

Output

y Resulting value Double (F64)

Parameter

n1 List of signals to subtract instead of adding. The format of the list Long (I32)
is e.g. 1,3..5,8. Third-party programs (Simulink, OPC clients
etc.) work with an integer number, which is a binary mask, i.e.
157 (binary 10011101) in the mentioned case.
CNB – Boolean (logic) constant

Block Symbol

\[ \text{CNB} \]

Licence: STANDARD

Function Description

The CNB block stands for a Boolean (logic) constant.

Output

\( Y \)  Logical output of the block  \( \text{Bool} \)

Parameter

\( YCN \)  Boolean constant  \( \circ \text{on} \)  \( \text{Bool} \)

- off  ...  Disabled
- on  ...  Enabled
CNE – Enumeration constant

Block Symbol Licence: STANDARD

Function Description

The CNE block allows selection of a constant from a predefined popup list. The popup list of constants is defined by the pupstr string, whose syntax is obvious from the default value shown below. The output value corresponds to the number at the beginning of the selected item. In case the pupstr string format is invalid, the output is set to 0.

There is a library called CNEs in Simulink, which contains CNE blocks with the most common lists of constants.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>yenum</td>
<td>Enumeration constant</td>
<td>Option A String</td>
</tr>
<tr>
<td>pupstr</td>
<td>Popup list definition</td>
<td>Option A String</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iy</td>
<td>Integer output of the block</td>
</tr>
</tbody>
</table>
CNI – Integer constant

Block Symbol

Licence: STANDARD

Function Description
The CNI block stands for an integer constant.

Output
\[ iy \quad \text{Integer output of the block} \quad \text{Long (I32)} \]

Parameter
\[ icn \quad \text{Integer constant} \quad \text{#1 Long (I32)} \]
CNR – Real constant

Block Symbol

Function Description
The CNR block stands for a real constant.

Output

\[ y \]  Analog output of the block  Double (F64)

Parameter

\[ \text{ycn} \]  Real constant  \( \circ 1.0 \)  Double (F64)
DIF_ — Difference

Block Symbol

Function Description
The DIF_ block differentiates the input signal u according to the following formula

\[ y_k = u_k - u_{k-1}, \]

where \( u_k = u \), \( y_k = y \) and \( u_{k-1} \) is the value of input u in the previous cycle (delay \( T_s \), which is the execution period of the block).

Input
\[ u \quad \text{Analog input of the block} \quad \text{Double (F64)} \]

Output
\[ y \quad \text{Difference of the input signal} \quad \text{Double (F64)} \]

Parameters
\[ \text{ISSF} \quad \text{Zero output at start-up} \quad \text{Bool} \]

off . . . In the first cycle the output will be \( y = u \).
on . . . Zero output in the first cycle, \( y = 0 \).
DIV – Division of two signals

Block Symbol

Function Description

The DIV block divides two analog input signals $y = \frac{u_1}{u_2}$. In case $u_2 = 0$, the output $E$ is set to on and the output $y$ is substituted by $y = y_{err}$.

Inputs

- $u_1$: First analog input of the block
- $u_2$: Second analog input of the block

Outputs

- $y$: Quotient of the inputs
- $E$: Error flag – division by zero

Parameter

- $y_{err}$: Substitute value for an error case
  - $1.0$: Double (F64)
EAS – Extended addition and subtraction

Block Symbol

Licence: STANDARD

Function Description

The EAS block sums input analog signals $u_1$, $u_2$, $u_3$ and $u_4$ with corresponding weights $a$, $b$, $c$ and $d$. The output $y$ is then given by

$$ y = a \cdot u_1 + b \cdot u_2 + c \cdot u_3 + d \cdot u_4 + y_0. $$

Inputs

<table>
<thead>
<tr>
<th>$u_1$</th>
<th>First analog input of the block</th>
<th>Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_2$</td>
<td>Second analog input of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$u_3$</td>
<td>Third analog input of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$u_4$</td>
<td>Fourth analog input of the block</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

Output

| $y$ | Analog output of the block | Double (F64) |

Parameters

<table>
<thead>
<tr>
<th>$a$</th>
<th>Weighting coefficient of the $u_1$ input</th>
<th>$\odot 1.0$ Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>Weighting coefficient of the $u_2$ input</td>
<td>$\odot 1.0$ Double (F64)</td>
</tr>
<tr>
<td>$c$</td>
<td>Weighting coefficient of the $u_3$ input</td>
<td>$\odot 1.0$ Double (F64)</td>
</tr>
<tr>
<td>$d$</td>
<td>Weighting coefficient of the $u_4$ input</td>
<td>$\odot 1.0$ Double (F64)</td>
</tr>
<tr>
<td>$y_0$</td>
<td>Additive constant (bias)</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
EMD – Extended multiplication and division

Block Symbol

Licence: STANDARD

Function Description

The EMD block multiplies and divides analog input signals u1, u2, u3 and u4 with corresponding weights a, b, c and d. The output y is then given by

\[ y = \frac{(a \times u1 + a0)(b \times u2 + b0)}{(c \times u3 + c0)(d \times u4 + d0)}. \]  

(4.1)

The output E is set to on in the case that the denominator in the equation (4.1) is equal to 0 and the output y is substituted by \( y = yerr \).

Inputs

- **u1**: First analog input of the block
- **u2**: Second analog input of the block
- **u3**: Third analog input of the block
- **u4**: Fourth analog input of the block

Outputs

- **y**: Analog output of the block
- **E**: Error flag – division by zero

Parameters

- **a**: Weighting coefficient of the u1 input
- **a0**: Additive constant for u1 input
- **b**: Weighting coefficient of the u2 input
- **b0**: Additive constant for u2 input
- **c**: Weighting coefficient of the u3 input
- **c0**: Additive constant for u3 input
- **d**: Weighting coefficient of the u4 input
- **d0**: Additive constant for u4 input
- **yerr**: Substitute value for an error case
**FNX – Evaluation of single-variable function**

Block Symbol

 LICENCE: STANDARD

**Function Description**

The FNX block evaluates basic math functions of single variable. The table below shows the list of supported functions with corresponding constraints. The fn parameter determines the active function.

**List of functions:**

<table>
<thead>
<tr>
<th>fn: shortcut</th>
<th>function</th>
<th>constraints on u</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: acos</td>
<td>arccosine</td>
<td>( u \in (-1,1) )</td>
</tr>
<tr>
<td>2: asin</td>
<td>arcsine</td>
<td>( u \in (-1,1) )</td>
</tr>
<tr>
<td>3: atan</td>
<td>arctangent</td>
<td>-</td>
</tr>
<tr>
<td>4: ceil</td>
<td>rounding towards the nearest higher integer</td>
<td>-</td>
</tr>
<tr>
<td>5: cos</td>
<td>cosine</td>
<td>-</td>
</tr>
<tr>
<td>6: cosh</td>
<td>hyperbolic cosine</td>
<td>-</td>
</tr>
<tr>
<td>7: exp</td>
<td>exponential function ( e^u )</td>
<td>-</td>
</tr>
<tr>
<td>8: exp10</td>
<td>exponential function ( 10^u )</td>
<td>-</td>
</tr>
<tr>
<td>9: fabs</td>
<td>absolute value</td>
<td>-</td>
</tr>
<tr>
<td>10: floor</td>
<td>rounding towards the nearest lower integer</td>
<td>-</td>
</tr>
<tr>
<td>11: log</td>
<td>logarithm</td>
<td>( u &gt; 0 )</td>
</tr>
<tr>
<td>12: log10</td>
<td>decimal logarithm</td>
<td>( u &gt; 0 )</td>
</tr>
<tr>
<td>13: random</td>
<td>arbitrary number ( z \in (0,1) ) (u independent)</td>
<td>-</td>
</tr>
<tr>
<td>14: sin</td>
<td>sine</td>
<td>-</td>
</tr>
<tr>
<td>15: sinh</td>
<td>hyperbolic sine</td>
<td>-</td>
</tr>
<tr>
<td>16: sqrt</td>
<td>square function</td>
<td>-</td>
</tr>
<tr>
<td>17: sqrt</td>
<td>square root</td>
<td>( u &gt; 0 )</td>
</tr>
<tr>
<td>18: srand</td>
<td>changes the seed for the random function to u</td>
<td>( u \in \mathbb{N} )</td>
</tr>
<tr>
<td>19: tan</td>
<td>tangent</td>
<td>-</td>
</tr>
<tr>
<td>20: tanh</td>
<td>hyperbolic tangent</td>
<td>-</td>
</tr>
</tbody>
</table>

**Note:** All trigonometric functions process data in radians.

The error output is activated (E = on) in the case when the input value u falls out of its bounds or an error occurs during evaluation of the selected function (implementation dependent), e.g., square root of negative number. The output is set to substitute value in such case (y = yerr).
Input

u  Analog input of the block  Double (F64)

Outputs

y  Result of the selected function  Double (F64)

E  Error flag  Bool

Parameters

ifn  Function type (see table above)  01 Long (I32)

yerr  Substitute value for an error case  Double (F64)
FNXY – Evaluation of two-variables function

Block Symbol Licence: STANDARD

Function Description

The FNXY block evaluates basic math functions of two variables. The table below shows the list of supported functions with corresponding constraints. The fn parameter determines the active function.

List of functions:

<table>
<thead>
<tr>
<th>fn: shortcut</th>
<th>function</th>
<th>constraints on u1, u2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: atan2</td>
<td>arctangent u1/u2</td>
<td>-</td>
</tr>
<tr>
<td>2: fmod</td>
<td>remainder after division u1/u2</td>
<td>u2 ≠ 0.0</td>
</tr>
<tr>
<td>3: pow</td>
<td>exponentiation of the inputs y = u1^u2</td>
<td>-</td>
</tr>
</tbody>
</table>

The atan2 function result belongs to the interval (−π, π). The signs of both inputs u1 and u2 are used to determine the appropriate quadrant.

The fmod function computes the remainder after division u1/u2 such that u1 = i · u2 + y, where i is an integer, the signs of the y output and the u1 input are the same and the following holds for the absolute value of the y output: |y| < |u2|.

The error output is activated (E = on) in the case when the input value u2 does not meet the constraints or an error occurs during evaluation of the selected function (implementation dependent), e.g. division by zero. The output is set to substitute value in such case (y = yerr).

Inputs

- u1: First analog input of the block
- u2: Second analog input of the block

Outputs

- y: Result of the selected function
- E: Error flag
  - off ... No error
  - on ... An error occurred

Inputs

- u1: First analog input of the block
  - Double (F64)
- u2: Second analog input of the block
  - Double (F64)

Outputs

- y: Result of the selected function
  - Double (F64)
- E: Error flag
  - off ... No error
  - on ... An error occurred
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ifn</code></td>
<td>Function type (see the table above)</td>
<td>○1 Long (I32)</td>
</tr>
<tr>
<td></td>
<td>1 .... atan2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 .... fmod</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 .... pow</td>
<td></td>
</tr>
<tr>
<td><code>yerr</code></td>
<td>Substitute value for an error case</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
GAIN – Multiplication by a constant

Block Symbol

Function Description
The GAIN block multiplies the analog input \( u \) by a real constant \( k \). The output is then
\[
y = ku.
\]

Input

\( u \)
Analog input of the block
Double (F64)

Output

\( y \)
Analog output of the block
Double (F64)

Parameter

\( k \)
Gain
\( \circ1.0 \) Double (F64)
GRADS – Gradient search optimization

Function Description

The GRADS block performs one-dimensional optimization of the \( f(x, v) \) function by gradient method, where \( x \in (x_{\text{min}}, x_{\text{max}}) \) is the optimized variable and \( v \) is an arbitrary vector variable. It is assumed that the value of the function \( f(x, v) \) for given \( x \) at time \( k \) is enumerated and fed to the \( f \) input at time \( k + n \times T_S \), where \( T_S \) is the execution period of the GRADS block. This means that the individual optimization iterations have a period of \( n \times T_S \). The length of step of the gradient method is given by

\[
\begin{align*}
  \text{grad} &= (f_i - f_{i-1}) \times (dx)_{i-1} \\
  (dx)_i &= -\gamma \times \text{grad},
\end{align*}
\]

where \( i \) stands for \( i \)-th iteration. The step size is restricted to lie within the interval \( (d_{\text{min}}, d_{\text{max}}) \). The value of the optimized variable for the next iteration is given by

\[
x_{i+1} = x_i + (dx)_i
\]

Inputs

- \( f \) Value of the optimized \( f(.) \) for given variable \( x \) Double (F64)
- \( x0 \) Optimization starting point Double (F64)
- START Starting signal (rising edge) Bool
- BRK Termination signal Bool

Outputs

- \( x \) Current value of the optimized variable Double (F64)
- \( x_{\text{opt}} \) Resulting optimal value of the \( x \) variable Double (F64)
- \( f_{\text{opt}} \) Resulting optimal value of the function \( f(x, v) \) Double (F64)
- BSY Indicator of running optimization Bool
- \( \text{iter} \) Number of current iteration Long (I32)
- E Error flag Bool
### iE

Error code

1. \( x \not\in \langle \text{xmin}, \text{xmax} \rangle \)
2. \( x = \text{xmin} \) or \( x = \text{xmax} \)

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmin</td>
<td>Lower limit for the x variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>xmax</td>
<td>Upper limit for the x variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>gamma</td>
<td>Coefficient for determining the step size of the gradient optimization method</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>d0</td>
<td>Initial step size</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>dmin</td>
<td>Minimum step size</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>dmax</td>
<td>Maximum step size</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>n</td>
<td>Iteration period (in sampling periods ( T_s )</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>itermax</td>
<td>Maximum number of iterations</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>
IADD – Integer addition

Block Symbol

Function Description
The IADD block sums two integer input signals \( n = i1 + i2 \). The range of integer numbers in a computer is always restricted by the variable type. This block uses the \( \text{vtype} \) parameter to specify the type. If the sum fits in the range of the given type, the result is the ordinary sum. In the other cases the result depends on the SAT parameter.

The overflow is not checked for SAT = off, i.e. the output \( E = \text{off} \) and the output value \( n \) corresponds with the arithmetics of the processor. E.g. for the Short type, which has the range of \(-32768..+32767\), we obtain \( 30000 + 2770 = -32766 \).

For SAT = on the overflow results in setting the error output to \( E = \text{on} \) and the \( n \) output to the nearest displayable value. For the above mentioned example we get \( 30000 + 2770 = 32767 \).

Inputs
\[
\begin{array}{ll}
i1 & \text{First integer input of the block} \\
i2 & \text{Second integer input of the block}
\end{array}
\]
\( \downarrow -9.22E+18 \uparrow 9.22E+18 \) \( \text{Long (I32)} \)

Outputs
\[
\begin{array}{ll}
n & \text{Integer sum of the input signals} \\
E & \text{Error flag}
\end{array}
\]
\( \downarrow -9.22E+18 \uparrow 9.22E+18 \) \( \text{Long (I32)} \)

Error flag
- off ... No error
- on .... An error occurred

Parameters
\[
\begin{array}{ll}
\text{vtype} & \text{Numeric type} \\
\hline
\text{--} & \text{--} \\
2 & \text{Byte (U8)} \\
3 & \text{Short (I16)} \\
4 & \text{Long (I32)} \\
5 & \text{Word (U16)} \\
6 & \text{DWord (U32)} \\
\text{--} & \text{--} \\
\text{--} & \text{--} \\
10 & \text{Large (I64)}
\end{array}
\]

Licence: STANDARD
SAT  Saturation (overflow) checking  Bool
    off  . . . Overflow is not checked
    on  . . . Overflow is checked
**ISUB – Integer subtraction**

Block Symbol

<table>
<thead>
<tr>
<th>Licence: STANDARD</th>
</tr>
</thead>
<tbody>
<tr>
<td>![ISUB symbol]</td>
</tr>
</tbody>
</table>

**Function Description**

The ISUB block subtracts two integer input signals $n = i1 - i2$. The range of integer numbers in a computer is always restricted by the variable type. This block uses the $v$ type parameter to specify the type. If the difference fits in the range of the given type, the result is the ordinary sum. In the other cases the result depends on the SAT parameter.

The overflow is not checked for SAT = off, i.e. the output $E = off$ and the output value $n$ corresponds with the arithmetics of the processor. E.g. for the Short type, which has the range of $-32768...+32767$, we obtain $30000 - 2770 = 32767$.

For SAT = on the overflow results in setting the error output to $E = on$ and the $n$ output to the nearest displayable value. For the above mentioned example we get $30000 - 2770 = 32767$.

**Inputs**

| $i1$ | First integer input of the block | ↓-9.22E+18 ↑9.22E+18 | Long (I32) |
| $i2$ | Second integer input of the block | ↓-9.22E+18 ↑9.22E+18 | Long (I32) |

**Parameters**

<table>
<thead>
<tr>
<th>$v$ type</th>
<th>Numeric type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 ...</td>
<td>Byte (range 0... 255)</td>
</tr>
<tr>
<td>3 ...</td>
<td>Short (range -32768... 32767)</td>
</tr>
<tr>
<td>4 ...</td>
<td>Long (range -2147483648... 2147483647)</td>
</tr>
<tr>
<td>5 ...</td>
<td>Word (range 0... 65536)</td>
</tr>
<tr>
<td>6 ...</td>
<td>DWord (range 0... 4294967295)</td>
</tr>
<tr>
<td>10 ...</td>
<td>Large (range -9223372036854775808... 9223372036854775807)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SAT</th>
<th>Saturation (overflow) checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Overflow is not checked</td>
</tr>
<tr>
<td>on</td>
<td>Overflow is checked</td>
</tr>
</tbody>
</table>

| $n$ | Integer difference between the input signals | Long (I32) |
| $E$ | Error flag | Bool |

| $off$ | No error |
| $on$  | An error occurred |
IMUL – Integer multiplication

Block Symbol

Function Description
The IMUL block multiplies two integer input signals \( n = i_1 \times i_2 \). The range of integer numbers in a computer is always restricted by the variable type. This block uses the \( vtype \) parameter to specify the type. If the multiple fits in the range of the given type, the result is the ordinary multiple. In the other cases the result depends on the \( SAT \) parameter.

The overflow is not checked for \( SAT = \text{off} \), i.e. the output \( E = \text{off} \) and the output value \( n \) corresponds with the arithmetics of the processor. E.g. for the \( \text{Short} \) type, which has the range of \(-32768..+32767\), we obtain \( 2000 \times 20 = -25536 \).

For \( SAT = \text{on} \) the overflow results in setting the error output to \( E = \text{on} \) and the \( n \) output to the nearest displayable value. For the above mentioned example we get \( 2000 \times 20 = 32767 \).

Inputs

- \( i_1 \)      First integer input of the block             \( \downarrow-9.22E+18 \uparrow 9.22E+18 \)  \( \text{Long (I32)} \)
- \( i_2 \)      Second integer input of the block            \( \downarrow-9.22E+18 \uparrow 9.22E+18 \)  \( \text{Long (I32)} \)

Parameters

- \( vtype \)    Numeric type                                    \( \odot 4 \)  \( \text{Long (I32)} \)
  - \(-- \ldots\)
  - \( 2 \ldots \)  Byte (U8)
  - \( 3 \ldots \)  Short (I16)
  - \( 4 \ldots \)  Long (I32)
  - \( 5 \ldots \)  Word (U16)
  - \( 6 \ldots \)  DWord (U32)
  - \(-- \ldots\)
  - \(-- \ldots\)
  - \( 10 \ldots \) Large (I64)

- \( SAT \)    Saturation (overflow) checking                  \( \text{Bool} \)
  - \( \text{off} \ldots \)  Overflow is not checked
  - \( \text{on} \ldots \)  Overflow is checked
### Outputs

<table>
<thead>
<tr>
<th>n</th>
<th>Integer product of the input signals</th>
<th>Long (132)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Error flag</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... No error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... An error occurred</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 4. MATH - MATH BLOCKS

IDIV – Integer division

Block Symbol

Licence: STANDARD

Function Description

The IDIV block performs an integer division of two integer input signals, \( n = i_1 \div i_2 \), where \( \div \) stands for integer division operator. If the ordinary (non-integer, normal) quotient of the two operands is an integer number, the result of integer division is the same. In other cases the resulting value is obtained by trimming the non-integer quotient’s decimals (i.e. rounding towards lower integer number). In case \( i_2 = 0 \), the output \( E \) is set to on and the output \( n \) is substituted by \( n = nerr \).

Inputs

\begin{align*}
  i_1 & \quad \text{First integer input of the block} & \quad [-9.22E+18 \uparrow 9.22E+18] & \quad \text{Long (I32)} \\
  i_2 & \quad \text{Second integer input of the block} & \quad [-9.22E+18 \uparrow 9.22E+18] & \quad \text{Long (I32)}
\end{align*}

Outputs

\begin{align*}
  n & \quad \text{Integer quotient of the inputs} & \quad \text{Long (I32)} \\
  E & \quad \text{Error flag – division by zero} & \quad \text{Bool}
\end{align*}

Parameters

\begin{align*}
  \text{vtype} & \quad \text{Numeric type} & \quad \odot 4 & \quad \text{Long (I32)} \\
  & \quad 2 \quad \text{Byte} & \quad \odot 4 & \quad \text{Long (I32)} \\
  & \quad 3 \quad \text{Short} & \quad \odot 4 & \quad \text{Long (I32)} \\
  & \quad 4 \quad \text{Long} & \quad \odot 4 & \quad \text{Long (I32)} \\
  & \quad 5 \quad \text{Word} & \quad \odot 4 & \quad \text{Long (I32)} \\
  & \quad 6 \quad \text{DWord} & \quad \odot 4 & \quad \text{Long (I32)} \\
  & \quad 10 \quad \text{Large} & \quad \odot 4 & \quad \text{Long (I32)}
\end{align*}

nerr & \quad \text{Substitute value for an error case} & \quad \odot 1 & \quad \text{Long (I32)}
**IMOD – Remainder after integer division**

**Block Symbol**

![Block Symbol](image)

**Function Description**

The IMOD block divides two integer input signals, \( n = i1 \% i2 \), where \( \% \) stands for remainder after integer division operator (modulo). If both numbers are positive and the divisor is greater than one, the result is either zero (for commensurable numbers) or a positive integer lower than the divisor. In the case that one of the numbers is negative, the result has the sign of the dividend, e.g. \( 15 \% 10 = 5 \), \( 15 \% (-10) = 5 \), but \( (-15) \% 10 = -5 \). In case \( i2 = 0 \), the output \( E \) is set to \( \text{on} \) and the output \( n \) is substituted by \( n = \text{nerr} \).

**Inputs**

- \( i1 \): First integer input of the block  
  \( \downarrow -9.22E+18 \uparrow 9.22E+18 \)  
  Long (I32)

- \( i2 \): Second integer input of the block  
  \( \downarrow -9.22E+18 \uparrow 9.22E+18 \)  
  Long (I32)

**Outputs**

- \( n \): Remainder after integer division  
  Long (I32)

- \( E \): Error flag – division by zero  
  Bool

**Parameters**

- \( vtype \): Numeric type  
  \( \odot 4 \)  
  Long (I32)

  - 2: Byte
  - 3: Short
  - 4: Long
  - 5: Word
  - 6: DWord
  - 10: Large

- \( nerr \): Substitute value for an error case  
  \( \odot 1 \)  
  Long (I32)
LIN – Linear interpolation

Block Symbol

Function Description

The LIN block performs linear interpolation. The following figure illustrates the influence of the input \( u \) and given interpolation points \([u_1, y_1]\) and \([u_2, y_2]\) on the output \( y \).

\[ \begin{align*}
  y_2 & \quad \text{Output analog output of the block} \\
  y_1 & \quad \text{Input analog input of the block}
\end{align*} \]

Output

\[ \begin{align*}
  y & \quad \text{Analog output of the block} \\
  u & \quad \text{Analog input of the block}
\end{align*} \]

Input

Parameters

\[ \begin{align*}
  u_1 & \quad \text{x-coordinate of the 1st interpolation node} \\
  y_1 & \quad \text{y-coordinate of the 1st interpolation node} \\
  u_2 & \quad \text{x-coordinate of the 2nd interpolation node} \\
  y_2 & \quad \text{y-coordinate of the 2nd interpolation node}
\end{align*} \]

\[ \begin{align*}
  & \quad \text{Double (F64)} \\
  & \quad \text{Double (F64)} \\
  & \quad \text{1.0} \text{ Double (F64)} \\
  & \quad \text{1.0} \text{ Double (F64)}
\end{align*} \]
MUL – Multiplication of two signals

Block Symbol

Function Description
The MUL block multiplies two analog input signals \( y = u_1 \cdot u_2 \).

Inputs
\[
\begin{array}{ll}
\text{u1} & \text{First analog input of the block} \\
\text{u2} & \text{Second analog input of the block}
\end{array}
\]

Output
\[
\begin{array}{ll}
\text{y} & \text{Product of the input signals}
\end{array}
\]

Licence: STANDARD
POL – Polynomial evaluation

Block Symbol

Function Description
The POL block evaluates the polynomial of the form:

\[ y = a_0 + a_1 u + a_2 u^2 + a_3 u^3 + a_4 u^4 + a_5 u^5 + a_6 u^6 + a_7 u^7 + a_8 u^8. \]

The polynomial is internally evaluated by using the Horner scheme to improve the numerical robustness.

Input
- \( u \) Analog input of the block Double (F64)

Output
- \( y \) Analog output of the block Double (F64)

Parameters
- \( a_i \) The \( i \)-th coefficient of the polynomial, \( i = 0, 1, \ldots, 8 \) Double (F64)
REC – Reciprocal value

Block Symbol

\[
\begin{array}{c}
\frac{1}{u} \\
\text{REC}
\end{array}
\]

Function Description

The REC block computes the reciprocal value of the input signal \( u \). The output is then

\[ y = \frac{1}{u}. \]

In case \( u = 0 \), the error indicator is set to \( E = \text{on} \) and the output is set to the substitutional value \( y = \text{yerr} \).

Input

\( u \)  
Analog input of the block 
Double (F64)

Outputs

\( y \)  
Analog output of the block 
Double (F64)

\( E \)  
Error flag – division by zero 
Bool

Parameter

\( \text{yerr} \)  
Substitute value for an error case 
\( \circ 1.0 \) 
Double (F64)
REL – Relational operator

Block Symbol

Function Description
The REL block evaluates the binary relation $u1 \circ u2$ between the values of the input signals and sets the output $Y$ according to the result of the relation "$\circ". The output is set to $Y = \text{on}$ when relation holds, otherwise it is zero (relation does not hold). The binary operation codes are listed below.

Inputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u1$</td>
<td>First analog input of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$u2$</td>
<td>Second analog input of the block</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y$</td>
<td>Logical output indicating whether the relation holds</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Parameter

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Relation type</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_{rel}$</td>
<td></td>
<td>$\circ1$ Long (I32)</td>
</tr>
</tbody>
</table>

1 . . . . equality ($==$)
2 . . . . inequality ($!=$)
3 . . . . less than ($<$)
4 . . . . greater than ($>$)
5 . . . . less than or equal to ($\leq$)
6 . . . . greater than or equal to ($\geq$)
RT0I – Real to integer number conversion

Block Symbol Licence: STANDARD

Function Description

The RT0I block converts the real number \( r \) to a signed integer number \( i \). The resulting rounded value is defined by:

\[
i := \begin{cases} 
-2147483648 & \text{for } r \leq -2147483648.0 \\
\text{round}(r) & \text{for } -2147483648.0 < r \leq 2147483647.0 \\
2147483647 & \text{for } r > 2147483647.0
\end{cases}
\]

where \( \text{round}(r) \) stands for rounding to the nearest integer number. The number of the form \( n+0.5 \) (\( n \) is integer) is rounded to the integer number with the higher absolute value, i.e. \( \text{round}(1.5) = 2 \), \( \text{round}(-2.5) = -3 \). Note that the numbers \(-2147483648\) and \(2147483647\) correspond with the lowest and the highest signed number representable in 32-bit format respectively (0x7FFFFFFF and 0x80000000 in hexadecimal form in the C language).

Input

\( r \) Analog input of the block Double (F64)

Output

\( i \) Rounded and converted input signal Long (I32)
**SQR – Square value**

Block Symbol

![SQR Block Symbol]

Licence: **STANDARD**

**Function Description**

The SQR block raises the input \( u \) to the power of 2. The output is then

\[
y = u^2.
\]

**Input**

\( u \quad \text{Analog input of the block} \qquad \text{Double (F64)} \)

**Output**

\( y \quad \text{Square of the input signal} \qquad \text{Double (F64)} \)
**SQRT_ – Square root**

Block Symbol

![SQRT_ symbol](image)

**Licence: STANDARD**

**Function Description**

The `SQRT_` block computes the square root of the input `u`. The output is then

\[ y = \sqrt{u}. \]

In case `u < 0`, the error indicator is activated (E = on) and the output `y` is set to the substitute value `y = yerr`.

**Input**

- `u` Analog input of the block
  - Type: Double (F64)

**Outputs**

- `y` Square root of the input signal
  - Type: Double (F64)
- `E` Error flag
  - `off` ... No error
  - `on` ... Square root of negative number

**Parameter**

- `yerr` Substitute value for an error case
  - Default: 1.0
  - Type: Double (F64)
**SUB – Subtraction of two signals**

Block Symbol

Licence: **STANDARD**

Function Description

The **SUB** block subtracts two input signals. The output is given by

\[ y = u_1 - u_2. \]

Consider using the **ADDOCT** block for addition or subtraction of multiple signals.

**Inputs**

- **u1**: Analog input of the block
- **u2**: Analog input of the block

**Output**

- **y**: Difference between the two input signals

**Double (F64)**
Chapter 5

ANALOG – Analog signal processing

Contents

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSROT</td>
<td>Processing data from absolute position sensor</td>
<td>101</td>
</tr>
<tr>
<td>ASW</td>
<td>Switch with automatic selection of input</td>
<td>103</td>
</tr>
<tr>
<td>AVG</td>
<td>Moving average filter</td>
<td>105</td>
</tr>
<tr>
<td>AVS</td>
<td>Motion control unit</td>
<td>106</td>
</tr>
<tr>
<td>BPF</td>
<td>Band-pass filter</td>
<td>107</td>
</tr>
<tr>
<td>CMP</td>
<td>Comparator with hysteresis</td>
<td>108</td>
</tr>
<tr>
<td>CNDR</td>
<td>Nonlinear conditioner</td>
<td>109</td>
</tr>
<tr>
<td>DEL</td>
<td>Delay with initialization</td>
<td>111</td>
</tr>
<tr>
<td>DELM</td>
<td>Time delay</td>
<td>112</td>
</tr>
<tr>
<td>DER</td>
<td>Derivation, filtering and prediction from the last n+1 samples</td>
<td>113</td>
</tr>
<tr>
<td>EVAR</td>
<td>Moving mean value and standard deviation</td>
<td>115</td>
</tr>
<tr>
<td>INTE</td>
<td>Controlled integrator</td>
<td>116</td>
</tr>
<tr>
<td>KDER</td>
<td>Derivation and filtering of the input signal</td>
<td>118</td>
</tr>
<tr>
<td>LPF</td>
<td>Low-pass filter</td>
<td>120</td>
</tr>
<tr>
<td>MINMAX</td>
<td>Running minimum and maximum</td>
<td>121</td>
</tr>
<tr>
<td>NSCL</td>
<td>Nonlinear scaling factor</td>
<td>122</td>
</tr>
<tr>
<td>RDFT</td>
<td>Running discrete Fourier transform</td>
<td>123</td>
</tr>
<tr>
<td>RLIM</td>
<td>Rate limiter</td>
<td>125</td>
</tr>
<tr>
<td>S1OF2</td>
<td>One of two analog signals selector</td>
<td>126</td>
</tr>
<tr>
<td>SAI</td>
<td>Safety analog input</td>
<td>129</td>
</tr>
<tr>
<td>SEL</td>
<td>Selector switch for analog signals</td>
<td>132</td>
</tr>
<tr>
<td>SELQUAD, SELCCT, SELHEXD</td>
<td>Selector switch for analog signals</td>
<td>133</td>
</tr>
<tr>
<td>SHIFTCT</td>
<td>Data shift register</td>
<td>135</td>
</tr>
<tr>
<td>SHLD</td>
<td>Sample and hold</td>
<td>137</td>
</tr>
</tbody>
</table>
SINT – Simple integrator .......................... 138
SPIKE – Spike filter .............................. 139
SSW – Simple switch ............................. 141
SWR – Selector with ramp ........................ 142
VDEL – Variable time delay ...................... 143
ZV4IS – Zero vibration input shaper ............ 144
ABSROT – Processing data from absolute position sensor

Block Symbol

Licence: ADVANCED

Function Description

The ABSROT function block is intended for processing the data from absolute position sensor on rotary equipment, e.g. a shaft. The absolute sensor has a typical range of 5° to 355° (or -175° to 175°) but in some cases it is necessary to control the rotation over a range of more than one revolution. The function block assumes a continuous position signal, therefore the transition from 355° to 5° in the input signal means that one revolution has been completed and the angle is in fact 365°.

In the case of long-term unidirectional operation the precision of the estimated position y deteriorates due to the precision of the double data type. For that case the R1 input is available to reset the position y to the base range of the sensor. If the RESR flag is set to RESR = on, the irev revolutions counter is also reset by the R1 input. In all cases it is necessary to reset all accompanying signals (e.g. the sp input of the corresponding controller).

The MPI output indicates that the absolute sensor reading is near to the middle of the range, which may be the appropriate time to reset the block. On the other hand, the OLI output indicates that the sensor reached the so-called dead-angle where it cannot report valid data.

Inputs

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>Signal from the absolute position sensor</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>R1</td>
<td>Block reset</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Position output</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>irev</td>
<td>Number of finished revolutions</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>MPI</td>
<td>Mid-point indicator</td>
<td>Bool</td>
</tr>
<tr>
<td>OLI</td>
<td>Off-limits indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Value</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>lolim</td>
<td>Lower limit of the sensor reading</td>
<td>-3.14159265</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>hilim</td>
<td>Upper limit of the sensor reading</td>
<td>3.14159265</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>tol</td>
<td>Tolerance for the mid-point indicator</td>
<td>0.5</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>hys</td>
<td>Hysteresis for the mid-point indicator</td>
<td>Double (F64)</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>---------------------------------------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td>RESR</td>
<td>Flag for resetting the revolutions counter</td>
<td>Bool</td>
<td></td>
</tr>
<tr>
<td></td>
<td>off . . . Reset only the estimated position y</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>on . . . Reset also the \texttt{irev} revolutions counter</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**ASW – Switch with automatic selection of input**

**Block Symbol**

License: **ADVANCED**

**Function Description**

The ASW block copies one of the inputs \(u_1, \ldots, u_4\) or one of the parameters \(p_1, \ldots, p_4\) to the output \(y\). The appropriate input signal is copied to the output as long as the input signal \(iSW\) belongs to the set \(\{1, 2, 3, 4\}\) and the parameters are copied when \(iSW\) belongs to the set \(\{-1, -2, -3, -4\}\) (i.e. \(y = p_1\) for \(iSW = -1\), \(y = u_3\) for \(iSW = 3\) etc.). If the \(iSW\) input signal differs from any of these values (i.e. \(iSW = 0\) or \(iSW < -4\) or \(iSW > 4\)), the output is set to the value of input or parameter which has changed the most recently. The signal or parameter is considered changed when it differs by more than \(\text{delta}\) from its value at the moment of its last change (i.e. the changes are measured integrally, not as a difference from the last sample). The following priority order is used when changes occur simultaneously in more than one signal: \(p_4, p_3, p_2, p_1, u_4, u_3, u_2, u_1\). The identifier of input signal or parameter which is copied to the output \(y\) is always available at the \(oSW\) output.

The ASW block has one special feature. The updated value of \(y\) is copied to all the parameters \(p_1, \ldots, p_4\). This results in all external tools reading the same value \(y\). This is particularly useful in higher-level systems which use the set&follow method (e.g. a slider in Iconics Genesis). This feature is not implemented in Simulink as there are no ways to read the values of inputs by external programs.

**ATTENTION!** One of the inputs \(u_1, \ldots, u_4\) can be delayed by one step when the block is contained in a loop. This might result in an illusion, that the priority is broken (the \(oSW\) output then shows that the most recently changed signal is the delayed one). In such a situation the \texttt{LPRK} block(s) must be used in appropriate positions.

**Inputs**

- **\(u_1..u_4\)**: Analog input signals to be selected from
- **\(iSW\)**: Active signal or parameter selector

**Outputs**

- **\(y\)**: The selected analog signal or parameter
- **\(oSW\)**: Identifier of the selected signal or parameter
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>delta</td>
<td>Threshold for detecting a change</td>
<td>1e-06</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>p1..p4</td>
<td>Parameters to be selected from</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
AVG – Moving average filter

Block Symbol

Licence: STANDARD

Function Description

The AVG block computes a moving average from the last \( n \) samples according to the formula
\[
y_k = \frac{1}{n} (u_k + u_{k-1} + \cdots + u_{k-n+1}).
\]

There is a limitation \( n < N \), where \( N \) depends on the implementation.

If the last \( n \) samples are not yet known, the average is computed from the samples available.

Input
\[
u \quad \text{Input signal to be filtered} \\
\]
Double (F64)

Output
\[
y \quad \text{Filtered output signal} \\
\]
Double (F64)

Parameter
\[
\begin{align*}
n & \quad \text{Number of samples to compute the average from} \\
\text{Long (I32)} \quad \downarrow 1 \uparrow 10000000 \odot 10 \\
\text{nmax} & \quad \text{Limit for parameter } n \text{ (used for internal memory allocation)} \\
\text{Long (I32)} \quad \downarrow 10 \uparrow 10000000 \odot 100
\end{align*}
\]
AVS – Motion control unit

Block Symbol

Function Description

The AVS block generates time-optimal trajectory from initial steady position 0 to a final steady position sm while respecting the constraints on the maximal acceleration am, maximal deceleration dm and maximal velocity vm. When rising edge (off→on) occurs at the SET input, the block is initialized for current values of the inputs am, dm, vm and sm. The RDY output is set to off before the first initialization and during the initialization phase, otherwise it is set to 1. When rising edge (off→on) occurs at the START input, the block generates the trajectory at the outputs a, v, s and tt, where the signals correspond to acceleration, velocity, position and time respectively. The BSY output is set to on while the trajectory is being generated, otherwise it is off.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>Starting signal (rising edge)</td>
<td>Bool</td>
</tr>
<tr>
<td>SET</td>
<td>Initialize/compute the trajectory for the current inputs</td>
<td>Bool</td>
</tr>
<tr>
<td>am</td>
<td>Maximal allowed acceleration [m/s²]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>dm</td>
<td>Maximal allowed deceleration [m/s²]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>vm</td>
<td>Maximum allowed velocity [m/s]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>sm</td>
<td>Desired final position [m] (initial position is 0)</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Acceleration [m/s²]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>v</td>
<td>Velocity [m/s]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>s</td>
<td>Position [m]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>tt</td>
<td>Time [s]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>RDY</td>
<td>Flag indicating that the block is ready to generate the trajectory</td>
<td>Bool</td>
</tr>
<tr>
<td>BSY</td>
<td>Flag indicating that the trajectory is being generated</td>
<td>Bool</td>
</tr>
</tbody>
</table>
BPF – Band-pass filter

Block Symbol

Function Description
The BPF implements a second order filter in the form

\[ F_s = \frac{2\xi as}{a^2s^2 + 2\xi as + 1}, \]

where \( a \) and \( \xi \) are the block parameters \( \text{fm} \) and \( \text{xi} \) respectively. The \( \text{fm} \) parameter defines the middle of the frequency transmission band and \( \xi \) is the relative damping coefficient.

If \( \text{ISSF} = \text{on} \), then the state of the filter is set to the steady value at the block initialization according to the input signal \( u \).

Input
\( u \)  Input signal to be filtered  Double (F64)

Output
\( y \)  Filtered output signal  Double (F64)

Parameters
\( \text{fm} \)  Peak frequency, middle of the frequency transmission band [Hz]  Double (F64)  \( \ominus 1.0 \)
\( \text{xi} \)  Relative damping coefficient (recommended value 0.5 to 1)  Double (F64)  \( \ominus 0.707 \)
\( \text{ISSF} \)  Steady state at start-up flag
\( \text{off} \) ... Zero initial state
\( \text{on} \) .... Initial steady state  Bool
CMP – Comparator with hysteresis

Block Symbol

Function Description

The CMP block compares the inputs \( u_1 \) and \( u_2 \) with the hysteresis \( h \) as follows:

\[
\begin{align*}
Y_{-1} &= 0, \\
Y_k &= \text{hyst}(e_k), \quad k = 0, 1, 2, \ldots
\end{align*}
\]

where

\[
e_k = u_{1_k} - u_{2_k}
\]

and

\[
\text{hyst}(e_k) = \begin{cases} 
0 & \text{for } e_k \leq -h \\
Y_{k-1} & \text{for } e_k \in (-h, h) \\
1 & \text{for } e_k \geq h \quad (e_k > h \text{ for } h = 0)
\end{cases}
\]

The indexed variables refer to the values of the corresponding signal in the cycle defined by the index, i.e. \( Y_{k-1} \) denotes the value of output in the previous cycle/step. The value \( Y_{-1} \) is used only once when the block is initialized \( (k = 0) \) and the difference of the input signals \( e_k \) is within the hysteresis limits.

Inputs

- \( u_1 \) First analog input of the block Double (F64)
- \( u_2 \) Second analog input of the block Double (F64)

Output

- \( Y \) Logical output of the block Bool

Parameter

- \( \text{hys} \) Hysteresis \( \odot 0.5 \) Double (F64)
**CNDR – Nonlinear conditioner**

**Block Symbol**

![Block Symbol](image)

**Function Description**

The CNDR block can be used for compensation of complex nonlinearities by a piecewise linear transformation which is depicted below.

![Diagram](image)

It is important to note that in the case of \( u < u_0 \) or \( u > u_{n-1} \) the output depends on the SATF parameter.

**Input**

\( u \)  
Analog input of the block  
Double (F64)

**Outputs**

\( y \)  
Analog output of the block  
Double (F64)

\( is \)  
Active sector of nonlinearity (see the figure above)  
Long (I32)

**Parameters**

\( n \)  
Number of \((u, y)\) node pairs  
\( \odot 6 \)  
Long (I32)

\( SATF \)  
Saturation flag  
\( \odot \text{on} \)  
Bool

\( \text{off} \ldots \text{Signal not limited} \)  
\( \ldots \text{Saturation limits active} \)
up  Vector of increasing $u$-coordinates  Double (F64)
    $\circ [0.0\ 3.9\ 3.9\ 9.0\ 14.5\ 20.0]$

yp  Vector of $y$-coordinates  Double (F64)
    $\circ [0.0\ 0.0\ 15.8\ 38.4\ 72.0\ 115.0]$
**DEL – Delay with initialization**

**Block Symbol**

![Block Symbol]

**Licence:** STANDARD

**Function Description**

The DEL block implements a delay of the input signal \( u \). The signal is shifted \( n \) samples backwards, i.e.

\[
y_k = u_{k-n}.
\]

If the last \( n \) samples are not yet known, the output is set to

\[
y_k = y_0,
\]

where \( y_0 \) is the initialization input signal. This can happen after restarting the control system or after resetting the block (R1: off → on → off) and it is indicated by the output \( RDY = \text{off} \).

**Inputs**

- \( u \): Analog input of the block
  - Double (F64)
- \( R1 \): Block reset
  - Bool
- \( y_0 \): Initial output value
  - Double (F64)

**Outputs**

- \( y \): Delayed input signal
  - Double (F64)
- \( RDY \): Ready flag indicating that the buffer is filled with the input signal samples
  - Bool

**Parameter**

- \( n \): Delay (number of samples). The resulting time delay is \( n \cdot T_S \), where \( T_S \) is the block execution period.
  - Long (I32)
  - \( \downarrow 0 \uparrow 10000000 \odot 10 \)
- \( n_{\text{max}} \): Limit for parameter \( n \) (used for internal memory allocation)
  - Long (I32)
  - \( \downarrow 10 \uparrow 10000000 \odot 100 \)
DELm – Time delay

Block Symbol

Licence: STANDARD

Function Description

The DELm block implements a time delay of the input signal. The length of the delay is given by rounding the del parameter to the nearest integer multiple of the block execution period. The output signal is $y = 0$ for the first del seconds after initialization.

Input

$u$  Analog input of the block  Double (F64)

Output

$y$  Delayed input signal  Double (F64)

Parameter

<table>
<thead>
<tr>
<th>del</th>
<th>Time delay [s]</th>
<th>$\odot 1.0$</th>
<th>Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmax</td>
<td>Size of delay buffer del (number of samples). Used for internal memory allocation.</td>
<td>$\downarrow 10 \uparrow 10000000 \odot 100$</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>
DER – Derivation, filtering and prediction from the last \( n+1 \) samples

Block Symbol

License: STANDARD

Function Description

The DER block interpolates the last \( n + 1 \) samples (\( n \leq N - 1 \), \( N \) is implementation dependent) of the input signal \( u \) by a line \( y = at + b \) using the least squares method. The starting point of the time axis is set to the current sampling instant.

In case of \( \text{RUN} = \text{on} \) the outputs \( y \) and \( z \) are computed from the obtained parameters \( a \) and \( b \) of the linear interpolation as follows:

- **Derivation:** \( y = a \)
- **Filtering:** \( z = b, \) for \( t_p = 0 \)
- **Prediction:** \( z = at_p + b, \) for \( t_p > 0 \)
- **Retrodiction:** \( z = at_p + b, \) for \( t_p < 0 \)

In case of \( \text{RUN} = \text{off} \) or \( n + 1 \) samples of the input signal are not yet available (\( \text{RDY} = \text{off} \)), the outputs are set to \( y = 0, \) \( z = u \).

Inputs

- \( u \): Analog output of the block (Double (F64))
- \( \text{RUN} \): Enable execution (Bool)
  - \( \text{off} \)... tracking (\( z = u \))
  - \( \text{on} \)... filtering (\( y = \text{estimate of the derivative}, \) \( z = \text{estimate of } u \) at time \( t_p \))
- \( t_p \): Time instant for prediction/filtering (\( t_p = 0 \) corresponds with the current sampling instant) (Double (F64))

Outputs

- \( y \): Estimate of input signal derivative (Double (F64))
- \( z \): Predicted/fILTERED input signal (Double (F64))
- \( \text{RDY} \): Ready flag (all \( n + 1 \) samples are available) (Bool)

Parameters

- \( n \): Number of samples for interpolation (\( n + 1 \) samples are used); \( n \leq n_{max} \)
  - \( 1 \leq n \leq n_{max} \) (Long (I32))
  - \( \downarrow 1 \uparrow 10000000 \odot 10 \)
**nmax**  Limit for parameter n (used for internal memory allocation)  
Long (I32)  
↓1 ↑10000000 ⊕10
EVAR – Moving mean value and standard deviation

Block Symbol

Function Description
The **EVAR** block estimates the mean value $\mu$ ($\mu$) and standard deviation $\sigma$ ($\sigma$) from the last $n$ samples of the input signal $u$ according to the formulas

$$
\mu_k = \frac{1}{n} \sum_{i=0}^{n-1} u_{k-i}
$$

$$
\sigma_k = \sqrt{\frac{1}{n} \sum_{i=0}^{n-1} u_{k-i}^2 - \mu_k^2}
$$

where $k$ stands for the current sampling instant.

Input

$u$ Analog input of the block Double (F64)

Outputs

$\mu$ Mean value of the input signal Double (F64)

$\sigma$ Standard deviation of the input signal Double (F64)

Parameter

$n$ Number of samples to estimate the statistical properties from Long (I32)

\[2 \uparrow 10000000 \circ 100\]

$n_{max}$ Limit value of parameter $n$ (used for internal memory allocation) Long (I32)

\[10 \uparrow 10000000 \circ 200\]
INTE – Controlled integrator

Block Symbol

Function Description

The INTE block implements a controlled integrator with variable integral time constant \( t_i \) and two indicators of the output signal level (\( y_{\text{min}} \) a \( y_{\text{max}} \)). If \( \text{RUN} = \text{on} \) and \( \text{R1} = \text{off} \) then

\[
y(t) = \frac{1}{T_i} \int_0^t u(\tau) d\tau + C,
\]

where \( C = y_0 \). If \( \text{RUN} = \text{off} \) and \( \text{R1} = \text{off} \) then the output \( y \) is frozen to the last value before the falling edge at the \( \text{RUN} \) input signal. If \( \text{R1} = \text{on} \) then the output \( y \) is set to the initial value \( y_0 \). The integration uses the trapezoidal method as follows

\[
y_k = y_{k-1} + \frac{T_S}{2T_i} (u_k + u_{k-1}),
\]

where \( T_S \) is the block execution period.

Consider using the \( \text{SINT} \) block, whose simpler structure and functionality might be sufficient for elementary tasks.

Inputs

- \( u \): Analog input of the block
- \( \text{RUN} \): Enable execution
  - \( \text{off} \): Integration stopped
  - \( \text{on} \): Integration running
- \( \text{R1} \): Block reset, initialization of the integrator output to \( y_0 \)
- \( y_0 \): Initial output value
- \( t_i \): Integral time constant

Outputs

- \( y \): Integrator output
- \( Q \): Running integration indicator
- \( \text{LY} \): Lower level indicator (\( y < y_{\text{min}} \))
- \( \text{HY} \): Upper level indicator (\( y > y_{\text{max}} \))
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ymin</td>
<td>Lower level definition</td>
<td>&gt;-1.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>ymax</td>
<td>Upper level definition</td>
<td>&lt;1.0</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
KDER – Derivation and filtering of the input signal

Function Description
The KDER block is a Kalman-type filter of the \text{norder}-th order aimed at estimation of derivatives of locally polynomial signals corrupted by noise. The order of derivatives ranges from 0 to \text{norder} – 1. The block can be used for derivation of almost arbitrary input signal \( u = u_0(t) + v(t) \), assuming that the frequency spectrums of the signal and noise differ.

The block is configured by only two parameters \text{ph} and \text{norder}. The \text{ph} parameter depends on the sampling period \( T_S \), frequency properties of the input signal \( u \) and also the noise to signal ratio. An approximate formula \( \text{ph} \approx T_S \omega_0 \) can be used. The frequency spectrum of the input signal \( u \) should be located deep down below the cutoff frequency \( \omega_0 \). But at the same time, the frequency spectrum of the noise should be as far away from the cutoff frequency \( \omega_0 \) as possible. The cutoff frequency \( \omega_0 \) and thus also the \text{ph} parameter must be lowered for strengthening the noise rejection.

The other parameter \text{norder} must be chosen with respect to the order of the estimated derivations. In most cases the 2nd or 3rd order filter is sufficient. Higher orders of the filter produce better derivation estimates for non-polynomial signals at the cost of slower tracking and higher computational cost.

Input
\( u \) \quad Input signal to be filtered \quad Double (F64)

Outputs
\( y \) \quad Filtered input signal \quad Double (F64)
\( dy \) \quad Estimated 1st order derivative \quad Double (F64)
\( d2y \) \quad Estimated 2nd order derivative \quad Double (F64)
\( d3y \) \quad Estimated 3rd order derivative \quad Double (F64)
\( d4y \) \quad Estimated 4th order derivative \quad Double (F64)
\( d5y \) \quad Estimated 5th order derivative \quad Double (F64)

Parameters
\text{norder} \quad Order of the derivative filter \quad \downarrow 2 \uparrow 10 \circ 3 \quad Long (I32)
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Value</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\beta$</td>
<td>Bandwidth of the derivative filter</td>
<td>$0.0 \circ 0.1$</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
LPF – Low-pass filter

Function Description

The LPF block implements a second order filter in the form

\[ F_s = \frac{1}{a^2s^2 + 2\xi as + 1}, \]

where

\[ a = \frac{\sqrt{2\sqrt{2\xi^2 - 2\xi^2 + 1} - 2\xi^2 + 1}}{2\pi f_b}, \]

and \( f_b \) and \( \xi = xi \) are the block parameters. The \( f_b \) parameter defines the filter bandwidth and \( xi \) is the relative damping coefficient. The recommended value is \( xi = 0.71 \) for the Butterworth filter and \( xi = 0.87 \) for the Bessel filter.

If ISSF = on, then the state of the filter is set to the steady value at the block initialization according to the input signal \( u \).

Input

\( u \) Input signal to be filtered Double (F64)

Output

\( y \) Filtered output signal Double (F64)

Parameters

\( f_b \) Filter bandwidth [Hz]: the frequencies in the range \((0, fb)\) pass through the filter, the attenuation at the frequency \( fb \) is 3 dB and approximately 40 dB at \( 10 \cdot fb \); it must hold \( fb < \frac{1}{10T_s} \) for proper function of the filter, where \( T_s \) is the block execution period \( \odot 1.0 \)

\( xi \) Relative damping coefficient (recommended value 0.5 to 1) \( \odot 0.707 \)

ISSF Steady state at start-up

\( \text{off} \ldots \text{Zero initial state} \)

\( \text{on} \ldots \text{Initial steady state} \)
MINMAX – Running minimum and maximum

Block Symbol

Function Description

The MINMAX function block evaluates minimum and maximum from the last n samples of the u input signal. The output RDY = off indicates that the buffer contains less than n samples. In such a case the minimum and maximum are found among the available samples.

Inputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>Analog input of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>R1</td>
<td>Block reset</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ymin</td>
<td>Minimal value found</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>ymax</td>
<td>Maximal value found</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>RDY</td>
<td>Ready flag (buffer filled)</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Number of samples for analysis (buffer length)</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>⌊1 ⌋10000000 ⌋c100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nm</td>
<td>Limit value of parameter n (used for internal memory allocation)</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>⌊10 ⌋10000000 ⌋c200</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NSCL – Nonlinear scaling factor

Block Symbol

Licence: STANDARD

Function Description

The NSCL block compensates common nonlinearities of the real world (e.g. the servo valve nonlinearity) by using the formula

\[ y = \text{gain} \frac{u}{\text{ze} + (1 - \text{ze}) \cdot u}, \]

where \text{gain} and \text{ze} are the parameters of the block. The choice of \text{ze} within the interval \((0, 1)\) leads to concave transformation, while \text{ze} > 1 gives a convex transformation.

Input

\( u \)  
Analog input of the block  
Double (F64)

Output

\( y \)  
Analog output of the block  
Double (F64)

Parameters

\( \text{gain} \)  
Signal gain  
\( \odot 1.0 \)  
Double (F64)

\( \text{ze} \)  
Shaping parameter  
\( \odot 1.0 \)  
Double (F64)
RDFT – Running discrete Fourier transform

Block Symbol

Function Description

The RDFT function block analyzes the analog input signal using the discrete Fourier transform with the fundamental frequency $freq$ and optional higher harmonic frequencies. The computations are performed over the last $m$ samples of the input signal $u$, where $m = \text{nper}/freq/T_S$, i.e. from the time-window of the length equivalent to $\text{nper}$ periods of the fundamental frequency.

If $\text{nhar}_m > 0$ the number of monitored higher harmonic frequencies is given solely by this parameter. On the contrary, for $\text{nhar}_m = 0$ the monitored frequencies are given by the user-defined vector parameter $freq2$.

For each frequency the amplitude ($\text{vAmp}$ output), phase-shift ($\text{vPhi}$ output), real/cosine part ($\text{vRe}$ output) and imaginary/sine part ($\text{vIm}$ output). The output signals have the vector form, therefore the computed values for all the frequencies are contained within. Use the VTOR function block to disassemble the vector signals.

Inputs

- $u$: Analog input of the block
- HLD: Hold

Outputs

- $\text{amp}$: Amplitude of the fundamental frequency
- $\text{thd}$: Total harmonic distortion (only for $\text{nhar}_m \geq 1$)
- $\text{vAmp}$: Vector of amplitudes at given frequencies
- $\text{vPhi}$: Vector of phase-shifts at given frequencies
- $\text{vRe}$: Vector of real parts at given frequencies
- $\text{vIm}$: Vector of imaginary parts at given frequencies
- $E$: Error flag
- $iE$: Error code
  - $i$ . . . . . . REXYGEN general error
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Range (default)</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>freq</td>
<td>Fundamental frequency</td>
<td>( \downarrow 1 \mathrm{e}-09 \uparrow 1 \mathrm{e}+09 \circ 1 \mathrm{.}0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>nper</td>
<td>Number of periods to calculate upon</td>
<td>( \downarrow 1 \uparrow 10000 \circ 10 )</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>nharm</td>
<td>Number of monitored harmonic frequencies</td>
<td>( \downarrow 0 \uparrow 16 \circ 3 )</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>ifrunit</td>
<td>Frequency units</td>
<td>( \downarrow 1 \uparrow 2 \circ 1 )</td>
<td>Long (I32)</td>
</tr>
<tr>
<td></td>
<td>Frequency units</td>
<td>1 ..... Hz</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 ..... rad/s</td>
<td></td>
</tr>
<tr>
<td>iphunit</td>
<td>Phase shift units</td>
<td>( \downarrow 0 \uparrow 2 \circ 1 )</td>
<td>Long (I32)</td>
</tr>
<tr>
<td></td>
<td>Phase shift units</td>
<td>1 ..... degrees</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 ..... radians</td>
<td></td>
</tr>
<tr>
<td>nmax</td>
<td>Allocated size of array</td>
<td>( \downarrow 10 \uparrow 10000000 \circ 8192 )</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>freq2</td>
<td>Vector of user-defined monitored frequencies</td>
<td>( \circ [2.0 \ 3.0 \ 4.0] )</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
RLIM – Rate limiter

Block Symbol

Function Description

The RLIM block copies the input signal $u$ to the output $y$, but the maximum allowed rate of change is limited. The limits are given by the time constants $t_\text{p}$ and $t_n$:

\[
\begin{align*}
\text{the steepest rise per second:} & \quad 1/tp \\
\text{the steepest descent per second:} & \quad -1/tn
\end{align*}
\]

Input

$u$ \hspace{1cm} \text{Input signal to be filtered} \hspace{1cm} \text{Double (F64)}$

Output

$y$ \hspace{1cm} \text{Filtered output signal} \hspace{1cm} \text{Double (F64)}$

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_p$</td>
<td>Time constant defining the maximum allowed rise</td>
<td>2.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$t_n$</td>
<td>Time constant defining the maximum allowed descent (note that $t_n &gt; 0$)</td>
<td>2.0</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
S10F2 – One of two analog signals selector

Block Symbol

Function Description

The S10F2 block assesses the validity of two input signals \( u_1 \) and \( u_2 \) separately. The validation method is equal to the method used in the SAI block. If the signal \( u_1 \) (or \( u_2 \)) is marked invalid, the output \( E_1 \) (or \( E_2 \)) is set to on and the error code is sent to the \( iE_1 \) (or \( iE_2 \)) output. The S10F2 block also evaluates the difference between the two input signals. The internal flag \( D \) is set to on if the differences \( |u_1 - u_2| \) in the last \( nd \) samples exceed the given limit, which is given by the following inequation

\[
|u_1 - u_2| > p\text{dev} \cdot \frac{v_{\text{max}} - v_{\text{min}}}{100},
\]

where \( v_{\text{min}} \) and \( v_{\text{max}} \) are the minimal and maximal limits of the inputs \( u_1 \) and \( u_2 \) and \( p\text{dev} \) is the allowed percentage difference with respect to the overall range of the input signals. The value of the output \( y \) depends on the validity of the input signals (flags \( E_1 \) and \( E_2 \)) and the internal difference flag \( D \) as follows:

(i) If \( E_1 = \text{off} \) and \( E_2 = \text{off} \) and \( D = \text{off} \), then the output \( y \) depends on the mode parameter:

\[
y = \begin{cases} 
\frac{u_1 + u_2}{2}, & \text{for mode } = 1, \\
\min(u_1, u_2), & \text{for mode } = 2, \\
\max(u_1, u_2), & \text{for mode } = 3,
\end{cases}
\]

and the output \( E \) is set to off unless set to on earlier.

(ii) If \( E_1 = \text{off} \) and \( E_2 = \text{off} \) and \( D = \text{on} \), then \( y = sv \) and \( E = on \).

(iii) If \( E_1 = \text{on} \) and \( E_2 = \text{off} \) (\( E_1 = \text{off} \) and \( E_2 = \text{on} \)), then \( y = u_2 \) (\( y = u_1 \)) and the output \( E \) is set to off unless set to on earlier.

(iv) If \( E_1 = \text{on} \) and \( E_2 = \text{on} \), then \( y = sv \) and \( E = on \).

The input \( R \) resets the inner error flags F1–F4 (see the SAI block) and the \( D \) flag. For the input \( R \) set permanently to on, the invalidity indicator \( E_1 \) (\( E_2 \)) is set to on for only one cycle period whenever some invalidity condition is fulfilled. On the other hand, for \( R = 0 \), the output \( E_1 \) (\( E_2 \)) is set to on and remains true until the reset (\( R: \text{off} \rightarrow \text{on} \)).
similar rule holds for the E output. For the input R set permanently to on, the E output is set to on for only one cycle period whenever a rising edge occurs in the internal D flag (D = off → on). On the other hand, for R = 0, the output E is set to on and remains true until the reset (rising edge R: off → on). The output W is set to on only in the (iii) or (iv) cases, i.e. at least one input signal is invalid.

Inputs

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>First analog input of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>u2</td>
<td>Second analog input of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>sv</td>
<td>Substitute value for an error case, i.e. E = on</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>HF1</td>
<td>Hardware error flag for signal u1</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... The input module of the signal works normally</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... Hardware error of the input module occurred</td>
<td></td>
</tr>
<tr>
<td>HF2</td>
<td>Hardware error flag for signal u2</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... The input module of the signal works normally</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... Hardware error of the input module occurred</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>Reset inner error flags of the input signals u1 and u2</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Analog output of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>E</td>
<td>Output signal invalidity indicator</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... Signal is valid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... Signal is invalid</td>
<td></td>
</tr>
<tr>
<td>E1</td>
<td>Invalidity indicator for input u1</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... Signal is valid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... Signal is invalid, y = u2</td>
<td></td>
</tr>
<tr>
<td>E2</td>
<td>Invalidity indicator for input u2</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... Signal is valid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... Signal is invalid, y = u1</td>
<td></td>
</tr>
<tr>
<td>iE1</td>
<td>Reason of input u1 invalidity</td>
<td>Long (I32)</td>
</tr>
<tr>
<td></td>
<td>0 ..... Signal valid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 ..... Signal out of range</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ..... Signal varies too little</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 ..... Signal varies too little and signal out of range</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 ..... Signal varies too much</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 ..... Signal varies too much and signal out of range</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6 ..... Signal varies too much and too little</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7 ..... Signal varies too much and too little and signal out of range</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 ..... Hardware error</td>
<td></td>
</tr>
<tr>
<td>iE2</td>
<td>Reason of input u2 invalidity, see the iE1 output</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>W</td>
<td>Warning flag (invalid input signal)</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... Both input signals u1 and u2 are valid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... At least one of the input signals is invalid</td>
<td></td>
</tr>
</tbody>
</table>
Parameters

\textbf{nb} \quad \text{Number of samples which are not included in the validity assessment of the signals } u_1 \text{ and } u_2 \text{ after initialization of the block} \quad \text{Long (I32)}

\textbf{nc} \quad \text{Number of samples for invariability testing (see the SAI block, condition F2)} \quad \text{Long (I32)}

\textbf{nbits} \quad \text{Number of A/D converter bits (source of the signals } u_1 \text{ and } u_2) \quad \text{Long (I32)}

\textbf{nr} \quad \text{Number of samples for variability testing (see the SAI block, condition F3)} \quad \text{Long (I32)}

\textbf{prate} \quad \text{Maximum allowed percentage change of the input } u_1 (u_2) \text{ within the last } nr \text{ samples (with respect to the overall range of the input signals } \text{vmax} - \text{vmin}, \text{see the SAI block)} \quad \text{Double (F64)}

\textbf{nv} \quad \text{Number of samples for out-of-range testing (see the SAI block, condition F4)} \quad \text{Long (I32)}

\textbf{vmin} \quad \text{Lower limit for the input signals } u_1 \text{ and } u_2 \quad \text{Double (F64)}

\textbf{vmax} \quad \text{Upper limit for the input signals } u_1 \text{ and } u_2 \quad \text{Double (F64)}

\textbf{nd} \quad \text{Number of samples for deviation testing (inner flag D; D is always off for } nd = 0) \quad \text{Long (I32)}

\textbf{pdev} \quad \text{Maximum allowed percentage deviation of the inputs } u_1 \text{ and } u_2 \text{ with respect to the overall range of the input signals } \text{vmax} - \text{vmin} \quad \text{Double (F64)}

\textbf{mode} \quad \text{Defines how to compute the output signal } y \text{ when both input signals are valid (} E1 = \text{off}, E2 = \text{off and } D = \text{off)} \quad \text{Long (I32)}

1 \ldots \quad \text{Average, } y = \frac{u_1 + u_2}{2}

2 \ldots \quad \text{Minimum, } y = \text{min}(u_1, u_2)

3 \ldots \quad \text{Maximum, } y = \text{max}(u_1, u_2)
SAI – Safety analog input

Block Symbol

Function Description

The SAI block tests the input signal \( u \) and assesses its validity. The input signal \( u \) is considered invalid (the output \( E = \text{on} \) ) in the following cases:

F1: Hardware error. The input signal \( \text{HWF} = \text{on} \).

F2: The input signal \( u \) varies too little. The last \( n_c \) samples of the input \( u \) lies within the interval of width \( d_u \),

\[
d_u = \begin{cases} \frac{v_{\text{max}} - v_{\text{min}}}{2^{n_{\text{bits}}}} , & \text{for } n_{\text{bits}} \in \{8, 9, ..., 16\} \\ 0 , & \text{for } n_{\text{bits}} \notin \{8, 9, ..., 16\}, \end{cases}
\]

where \( v_{\text{min}} \) and \( v_{\text{max}} \) are the lower and upper limits of the input \( u \), respectively, and \( n_{\text{bits}} \) is the number of A/D converter bits. The situation when the input signal \( u \) varies too little is shown in the following picture:

If the parameter \( n_c \) is set to \( n_c = 0 \), the condition \( F_2 = 0 \) is never fulfilled.

F3: The input signal \( u \) varies too much. The last \( n_r \) samples of the input \( u \) filtered by the SPIKE filter have a span which is greater than \( \text{rate} \),

\[
\text{rate} = \text{prate} \frac{v_{\text{max}} - v_{\text{min}}}{100},
\]

where \( \text{prate} \) defines the allowed percentage change in the input signal \( u \) within the last \( n_r \) samples (with respect to the overall range of the input signal \( u \in (v_{\text{min}}, v_{\text{max}}) \)).
The block includes a SPIKE filter with fixed parameters \( \text{mingap} = \frac{v_{\text{max}} - v_{\text{min}}}{100} \) and \( q = 2 \) suppressing peaks in the input signal to avoid undesirable fulfilling of this condition. See the SPIKE block description for more details. The situation when the input signal \( u \) varies too much is shown in the following picture:

If the parameter \( nr \) is set to \( nr = 0 \), the condition \( F3 \) is never fulfilled.

\( F4 \): The input signal \( u \) is out of range. The last \( nv \) samples of the input signal \( u \) lie out of the allowed range \( (v_{\text{min}}, v_{\text{max}}) \).

If the parameter \( nv \) is set to \( nv = 0 \), the condition \( F4 \) is never fulfilled.

The signal \( u \) is copied to the output \( y \) without any modification when it is considered valid. In the other case, the output \( y \) is determined by a substitute value from the \( sv \) input. In such a case the output \( E \) is set to \( \text{on} \) and the output \( iE \) provides the error code. The input \( R \) resets the inner error flags \( F1-F4 \). For the input \( R \) set permanently to \( \text{on} \), the invalidity indicator \( E \) is set to \( \text{on} \) for only one cycle period whenever some invalidity condition is fulfilled. On the other hand, for \( R = \text{off} \), the output \( E \) is set to \( \text{on} \) and remains true until the reset (rising edge \( R: \text{off} \rightarrow \text{on} \)).

The table of error codes \( iE \) resulting from the inner error flags \( F1-F4 \):

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>iE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>8</td>
</tr>
</tbody>
</table>

The \( nb \) parameter defines the number of samples which are not included in the validity assessment after initialization of the block (restart). Recommended setting is \( nb \geq 5 \) to allow the SPIKE filter initial conditions to fade away.
**Inputs**

\[ u \] Analog input of the block  
\[ sv \] Substitute value to be used when the signal \( u \) is marked as invalid  
\[ HWF \] Hardware error indicator  
\[ R \] Reset inner error flags F1–F4

**Outputs**

\[ y \] Analog output of the block  
\[ yf \] Filtered analog output signal \( y \), output of the SPIKE filter  
\[ E \] Output signal invalidity indicator  
\[ iE \] Reason of invalidity

**Parameters**

\[ nb \] Number of samples which are not included in the validity assessment of the signal \( u \) after initialization of the block  
\[ nc \] Number of samples for invariability testing (the F2 condition)  
\[ nbits \] Number of A/D converter bits  
\[ nr \] Number of samples for variability testing (the F3 condition)  
\[ prate \] Maximum allowed percentage change of the input \( u \) within the last \( nr \) samples (with respect to the overall range of the input signal \( v_{\text{max}} - v_{\text{min}} \))  
\[ nv \] Number of samples for out-of-range testing (the F4 condition)  
\[ vmin \] Lower limit for the input signal \( u \)  
\[ vmax \] Upper limit for the input signal \( u \)
SEL — Selector switch for analog signals

Block Symbol

Function Description

The SEL block is obsolete, replace it by the SELQUAD block. Note the difference in binary selector signals SWn.

The SEL block selects one of the four input signals u1, u2, u3 and u4 and copies it to the output signal y. The selection is based on the iSW input or the binary inputs SW1 and SW2. These two modes are distinguished by the BINF binary flag. The signal is selected according to the following table:

<table>
<thead>
<tr>
<th>iSW</th>
<th>SW1</th>
<th>SW2</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>off</td>
<td>off</td>
<td>u1</td>
</tr>
<tr>
<td>1</td>
<td>off</td>
<td>on</td>
<td>u2</td>
</tr>
<tr>
<td>2</td>
<td>on</td>
<td>off</td>
<td>u3</td>
</tr>
<tr>
<td>3</td>
<td>on</td>
<td>on</td>
<td>u4</td>
</tr>
</tbody>
</table>

Inputs

- u1: First analog input of the block, Double (F64)
- u2: Second analog input of the block, Double (F64)
- u3: Third analog input of the block, Double (F64)
- u4: Fourth analog input of the block, Double (F64)
- iSW: Active signal selector, active when BINF = off, Long (I32)
- SW1: Binary signal selector, active when BINF = on, Bool
- SW2: Binary signal selector, active when BINF = on, Bool

Output

- y: The selected signal, Double (F64)

Parameter

- BINF: Enable the binary selectors, Bool
  - off ... Disabled (analog selector)
  - on .... Enabled (binary selectors)
SELQUAD, SELOCT, SELHEXD – Selector switch for analog signals

Function Description

The SELQUAD, SELOCT and SELHEX blocks select one of the input signals and copy it to the output signal y. The selection of the active signal u0...u15 is based on the iSW input or the binary inputs SW0...SW3. These two modes are distinguished by the BINF binary flag. The signal is selected according to the following table:

<table>
<thead>
<tr>
<th>iSW</th>
<th>SW0</th>
<th>SW1</th>
<th>SW2</th>
<th>SW3</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>u0</td>
</tr>
<tr>
<td>1</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>u1</td>
</tr>
<tr>
<td>2</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>u2</td>
</tr>
<tr>
<td>3</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>u3</td>
</tr>
<tr>
<td>4</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>u4</td>
</tr>
<tr>
<td>5</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>u5</td>
</tr>
<tr>
<td>6</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>u6</td>
</tr>
<tr>
<td>7</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>u7</td>
</tr>
<tr>
<td>8</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>u8</td>
</tr>
<tr>
<td>9</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>u9</td>
</tr>
<tr>
<td>10</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>u10</td>
</tr>
<tr>
<td>11</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>u11</td>
</tr>
<tr>
<td>12</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>u12</td>
</tr>
<tr>
<td>13</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>u13</td>
</tr>
<tr>
<td>14</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>u14</td>
</tr>
<tr>
<td>15</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>u15</td>
</tr>
</tbody>
</table>

Please note that the only difference among the blocks is the number of inputs.
CHAPTER 5. ANALOG – ANALOG SIGNAL PROCESSING

Inputs

- $u_{0..15}$: Analog inputs of the block
- $iSW$: Active signal selector
- $SN0..3$: Binary signal selectors

Output

- $y$: The selected input signal

Parameter

- BINF: Enable the binary selectors
  - off ... Disabled (analog selector)
  - on ... Enabled (binary selectors)

Double (F64)
Long (I32)
Bool
Double (F64)
**SHIFTOCT – Data shift register**

Block Symbol

![Block Diagram](image)

**Function Description**

The SHIFTOCT block works as a shift register with eight outputs of arbitrary data type.

If the **RUN** input is active, the following assignment is performed with each algorithm tick:

\[
\begin{align*}
    y_i &= y_{i-1}, \quad i = 1..7 \\
    y_0 &= u
\end{align*}
\]

Thus the value on each output \(y_0\) to \(y_6\) is shifted to the following output and the value on input \(u\) is assigned to output \(y_0\).

The block works with any data type of signal connected to the input \(u\). Data type has to be specified by the \texttt{vtype} parameter. Outputs \(y_0\) to \(y_8\) then have the same data type.

If you need a triggered shift register, place the \texttt{EDGE} block in front of the \texttt{RUN} input.

**Inputs**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(u)</td>
<td>Data input of the register</td>
<td>Unknown</td>
</tr>
<tr>
<td><strong>RUN</strong></td>
<td>Enables outputs shift</td>
<td>\texttt{Bool}</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(y_0)</td>
<td>First output of the block</td>
<td>Unknown</td>
</tr>
<tr>
<td>(y_1)</td>
<td>Second output of the block</td>
<td>Unknown</td>
</tr>
<tr>
<td>(y_2)</td>
<td>Third output of the block</td>
<td>Unknown</td>
</tr>
<tr>
<td>(y_3)</td>
<td>Fourth output of the block</td>
<td>Unknown</td>
</tr>
<tr>
<td>(y_4)</td>
<td>Fifth output of the block</td>
<td>Unknown</td>
</tr>
<tr>
<td>(y_5)</td>
<td>Sixth output of the block</td>
<td>Unknown</td>
</tr>
<tr>
<td>(y_6)</td>
<td>Seventh output of the block</td>
<td>Unknown</td>
</tr>
<tr>
<td>(y_7)</td>
<td>Eighth output of the block</td>
<td>Unknown</td>
</tr>
</tbody>
</table>
## Parameters

<table>
<thead>
<tr>
<th>vtype</th>
<th>Output data type</th>
<th>32</th>
<th>8 Long (I32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Byte (U8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Short (I16)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Long (I32)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Word (U16)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>DWord (U32)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Float (F32)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Double (F64)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Large (I64)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SHLD – Sample and hold

Block Symbol

Function Description

The SHLD block is intended for holding the value of the input signal. It processes the input signal according to the mode parameter.

In Triggered sampling mode the block sets the output signal \( y \) to the value of the input signal \( u \) when rising edge (off→on) occurs at the SETH input. The output is held constant unless a new rising edge occurs at the SETH input.

If Hold last value mode is selected, the output signal \( y \) is set to the last value of the input signal \( u \) before the rising edge at the SETH input occurred. It is kept constant as long as SETH = on. For SETH = off the input signal \( u \) is simply copied to the output \( y \).

In Hold current value mode the \( u \) input is sampled right when the rising edge (off→on) occurs at the SETH input. It is kept constant as long as SETH = on. For SETH = off the input signal \( u \) is simply copied to the output \( y \).

The binary input R1 sets the output \( y \) to the value \( y_0 \), it overpowers the SETH input signal.

See also the PARR block, which can be used for storing a numeric value as well.

Inputs

- \( u \) Analog input of the block
- SETH Trigger for the set and hold operation
- R1 Block reset, \( R1 = on \) → \( y = y_0 \)

Outputs

- \( y \) Analog output of the block

Parameters

- \( y_0 \) Initial output value
- mode Sampling mode
  - 1 . . . . Triggered sampling
  - 2 . . . . Hold last value
  - 3 . . . . Hold current value

Licence: STANDARD
**SINT – Simple integrator**

**Block Symbol**

![SINT Symbol]

**Licence:** STANDARD

**Function Description**

The SINT block implements a discrete integrator described by the following difference equation

\[ y_k = y_{k-1} + \frac{T_s}{2T_i}(u_k + u_{k-1}), \]

where \( T_s \) is the block execution period and \( T_i \) is the integral time constant. If \( y_k \) falls out of the saturation limits \( y_{\text{min}} \) and \( y_{\text{max}} \), the output and state of the block are appropriately modified.

For more complex tasks, consider using the INTE block, which provides extended functionality.

**Input**

\[ u \quad \text{Analog input of the block} \quad \text{Double (F64)} \]

**Output**

\[ y \quad \text{Analog output of the block} \quad \text{Double (F64)} \]

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_i )</td>
<td>Integral time constant ( T_i )</td>
<td>1.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>( y_0 )</td>
<td>Initial output value</td>
<td></td>
<td>Double (F64)</td>
</tr>
<tr>
<td>( y_{\text{max}} )</td>
<td>Upper limit of the output signal</td>
<td>1.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>( y_{\text{min}} )</td>
<td>Lower limit of the output signal</td>
<td>-1.0</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
SPIKE – Spike filter

Block Symbol

Licence: ADVANCED

Function Description

The SPIKE block implements a nonlinear filter for suppressing isolated peaks (pulses) in the input signal \( u \). One cycle of the SPIKE filter performs the following transformation \( (u, y) \rightarrow y \):

\[
delta := y - u;
\]

\[
\text{if } \text{abs}(\delta) < \text{gap}
\]

\[
\begin{aligned}
\text{then} \\
\text{begin} \\
\quad y := u; \\
\quad \text{gap} := \frac{\text{gap}}{q}; \\
\quad \text{ifgap} < \text{mingap} \text{ then } \text{gap} := \text{mingap}; \\
\text{end}
\end{aligned}
\]

\[
\text{else}
\]

\[
\begin{aligned}
\text{begin} \\
\quad \text{if } \delta < 0 \\
\quad \text{then } y := y + \text{gap} \\
\quad \text{else } y := y - \text{gap}; \\
\quad \text{gap} := \text{gap} \times q; \\
\text{end}
\end{aligned}
\]

where \( \text{mingap} \) and \( q \) are the block parameters.

The signal passes through the filter unaffected for sufficiently large \( \text{mingap} \) parameter, which defines the minimal size of the tolerance window. By lowering this parameter it is possible to find an appropriate value, which leads to suppression of the undesirable peaks but leaves the input signal intact otherwise. The recommended value is 1 % of the overall input signal range. The \( q \) parameter determines the adaptation speed of the tolerance window.

Input

\( u \quad \text{Input signal to be filtered} \quad \text{Double (F64)} \)

Output

\( y \quad \text{Filtered output signal} \quad \text{Double (F64)} \)
### Parameters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Default Value</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{mingap} )</td>
<td>Minimum size of the tolerance window</td>
<td>( \odot 0.01 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>( q )</td>
<td>Tolerance window adaptation speed</td>
<td>( \downarrow 1.0 \odot 2.0 )</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
SSW – Simple switch

Block Symbol

Function Description
The SSW block selects one of two input signals $u_1$ and $u_2$ with respect to the binary input $SW$. The selected input is copied to the output $y$. If $SW = \text{off}$ ($SW = \text{on}$), then the selected signal is $u_1$ ($u_2$).

Inputs

- $u_1$ First analog input of the block $\rightarrow$ Double (F64)
- $u_2$ Second analog input of the block $\rightarrow$ Double (F64)
- $SW$ Signal selector $\rightarrow$ Bool
  
  - off $\ldots$ The $u_1$ signal is selected, $y = u_1$
  - on $\ldots$ The $u_2$ signal is selected, $y = u_2$

Output

- $y$ Analog output of the block $\rightarrow$ Double (F64)
**SWR – Selector with ramp**

Block Symbol

License: **STANDARD**

**Function Description**

The SWR block selects one of two input signals \( u_1 \) and \( u_2 \) with respect to the binary input \( SW \). The selected input is copied to the output \( y \). If \( SW = \text{off} \ (SW = \text{on}) \), then the selected signal is \( u_1 \) (\( u_2 \)). The output signal is not set immediately to the value of the selected input signal but tracks the selected input with given rate constraint (i.e. it follows a ramp). This rate constraint is configured independently for each input \( u_1 \), \( u_2 \) and is defined by time constants \( t_1 \) and \( t_2 \). As soon as the output reaches the level of the selected input signal, the rate limiter is disabled and remains inactive until the next signal switching.

**Inputs**

- \( u_1 \): First analog input of the block
- \( u_2 \): Second analog input of the block
- \( SW \): Signal selector
  - \( \text{off} \): The \( u_1 \) signal is selected
  - \( \text{on} \): The \( u_2 \) signal is selected

**Parameters**

- \( t_1 \): Rate limiter time constant for switching from \( u_2 \) to \( u_1 \)
- \( t_2 \): Rate limiter time constant for switching from \( u_1 \) to \( u_2 \)
- \( y_0 \): Initial output value to start the tracking from (before the first switching of signals occurs)

**Output**

- \( y \): Analog output of the block
**VDEL – Variable time delay**

**Block Symbol**

![VDEL symbol]

**Licence:** STANDARD

**Function Description**

The VDEL block delays the input signal $u$ by the time defined by the input signal $d$. More precisely, the delay is given by rounding the input signal $d$ to the nearest integer multiple of the block execution period ($n \cdot T_S$). A substitute value $y_0$ is used until $n$ previous samples are available after the block initialization.

**Inputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u$</td>
<td>Analog input of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$d$</td>
<td>Time delay [s]</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

**Output**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>Delayed input signal</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

**Parameter**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_0$</td>
<td>Initial/substitute output value</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$n_{max}$</td>
<td>Size of delay buffer (number of samples) for the time delay $d.$</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>

Used for internal memory allocation. $\downarrow$10 $\uparrow$10000000 $\odot$1000
ZV4IS — Zero vibration input shaper

Block Symbol

Licence: ADVANCED

Function Description

The function block ZV4IS implements a band-stop frequency filter. The main field of application is in motion control of flexible systems where the low stiffness of mechanical construction causes an excitation of residual vibrations which can be observed in form of mechanical oscillations. Such vibration can cause significant deterioration of quality of control or even instability of control loops. They often lead to increased wear of mechanical components. Generally, the filter can be used in arbitrary application for a purpose of control of an oscillatory system or in signal processing for selective suppression of particular frequency.

The input shaping filter can be used in two different ways. By using an open loop connection, the input reference signal for an feedback loop coming from human operator or higher level of control structure is properly shaped in order to attenuate any unwanted oscillations. The internal dynamics of the filter does not influence a behaviour of the interior loop. The only condition is correct tuning of feedback compensator \( C(s) \), which has to work in linear mode. Otherwise, the frequency spectrum of the manipulating variable gets corrupted and unwanted oscillations can still be excited in a plant \( P(s) \). The main disadvantage is passive vibration damping which works only in reference signal path. In case of any external disturbances acting on the plant, the vibrations may still arise. The second possible way of use is feedback connection. The input shaper is placed on the output side of feedback compensator \( C(s) \) and modifies the manipulating variable acting on the plant. An additional dynamics of the filter is introduced and the compensator \( C(s) \) needs to be properly tuned.

The algorithm of input shaper can be described in time domain

\[
y(t) = A_1u(t - t_1) + A_2u(t - t_2) + A_3u(t - t_3) + A_4u(t - t_4)
\]

Thus, the filter has a structure of sum of weighted time delays of an input signal. The gains \( A_1..A_4 \) and time delay values \( t_1..t_4 \) depend on a choice of filter type, natural
frequency and damping of controlled oscillatory mode of the system. The main advantage of this structure compared to commonly used notch filters is finite impulse response (which is especially important in motion control applications), warranted stability and monotone step response of the filter and generally lower dynamic delay introduced into a signal path.

For correct function of the filter, natural frequency omega and damping xi of the oscillatory mode need to be set. The parameter ipar sets a filter type. For ipar = 1, one of ten basic filter types chosen by istyle is used. Particular basic filters differ in shape and width of stop band in frequency domain. In case of precise knowledge of natural frequency and damping, the ZV (Zero Vibration) or ZVD filters can be used, because their response to input signal is faster compared to the other filters. In case of large uncertainty in system/signal model, robust UEI (Extra Insensitive) or UTHEI filters are good choice. Their advantage is wider stopband at the cost of slower response. The number on the end of the name has the meaning of maximum allowed level of excited vibrations for the given omega and xi (one, two or five percent).

For precise tuning of the filter, complete parameterization ipar = 2 can be selected. For this choice, three parameters p_alpha, p_a2 and p_a3 which affect the shape of the filter frequency response can freely be assigned. These parameters can be used for finding of optimal compromise between robustness of the filter and introduced dynamical delay.

The asymmetry parameter p_alpha determines relative location of the stopband of filter frequency response with respect to chosen natural frequency. Positive values mean a shift to higher frequency range, negative values to lower frequency range, zero value leads to symmetrical shape of the characteristic (see the figure above). The parameter p_alpha also affects the overall filter length, thus the overall delay introduced into a signal path. Lower values result in slower filters and higher delay. Asymmetric filters can be used in cases where a lower or higher bound of the uncertainty in natural frequency parameter is known.
Insensitivity parameter \( p_{a2} \) determines the width and attenuation level of the filter stopband. Higher values result in wider stopband and higher attenuation. For most applications, the value \( p_{a2} = 0.5 \) is recommended for highest achievable robustness with respect to modeling errors.

The additional parameter \( p_{a3} \) needs to be chosen for symmetrical filters (\( p_{\text{alpha}} = 0 \)). A rule for the most of the practical applications is to chose equal values \( p_{a2} = p_{a3} \) from interval \( < 0, 0.75 > \). Overall filter length is constant for this choice and only the shape of filter stopband is affected. Lower values lead to robust shapers with wide stopband and frequency response shape similar to standard THEI (Two-hump extra insensitive) filters. Higher values lead to narrow stopband and synchronous drop of two stopband peaks. The choice \( p_{a2} = p_{a3} = 0.75 \) results in standard ZVDD filter with maximally flat and symmetric stopband shape. The proposed scheme can be used for systematic tuning of the filter.

**Input**

\[
\begin{align*}
\text{u} & \quad \text{Input signal to be filtered} \\
\end{align*}
\]
Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Filtered output signal</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>E</td>
<td>Error flag</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td><strong>off</strong> ... No error <strong>on</strong> ... An error occurred</td>
<td></td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>omega</td>
<td>Natural frequency</td>
<td>0.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>xi</td>
<td>Relative damping coefficient</td>
<td></td>
<td>Double (F64)</td>
</tr>
<tr>
<td>ipar</td>
<td>Specification</td>
<td>0</td>
<td>Long (I32)</td>
</tr>
<tr>
<td></td>
<td><strong>1</strong> .... Basic types of IS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>2</strong> .... Complete parametrization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>istype</td>
<td>Type</td>
<td>0</td>
<td>Long (I32)</td>
</tr>
<tr>
<td></td>
<td><strong>1</strong> .... ZV</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>2</strong> .... ZVD</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>3</strong> .... ZVDD</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>4</strong> .... MISZV</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>5</strong> .... UEI1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>6</strong> .... UEI2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>7</strong> .... UEI5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>8</strong> .... UTHEI1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>9</strong> .... UTHEI2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>10</strong> .... UTHEI5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p_alpha</td>
<td>Shaper duration/assymetry parameter</td>
<td>0.2</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>p_a2</td>
<td>Insensitivity parameter</td>
<td>0.5</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>p_a3</td>
<td>Additional parameter (only for p_alpha = 0)</td>
<td>0.5</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>nmax</td>
<td>Size of data buffer (number of samples). Used for internal memory allocation.</td>
<td>10</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>
Chapter 6

GEN – Signal generators

Contents

AML – Controlled generator of piecewise linear function ........ 150
BINS – Controlled binary sequence generator .................. 152
BIS – Binary sequence generator ............................... 154
MP – Manual pulse generator .................................. 156
PRBS – Pseudo-random binary sequence generator .......... 157
SG, SGI – Signal generators .................................... 159
**ANLS – Controlled generator of piecewise linear function**

Block Symbol

Function Description
The ANLS block generates a piecewise linear function of time given by nodes \( t_1, y_1; t_2, y_2; t_3, y_3; t_4, y_4 \). The initial value of output \( y \) is defined by the \( y_0 \) parameter. The generation of the function starts when a rising edge occurs at the RUN input (and the internal timer is set to 0). The output \( y \) is then given by

\[
y = y_i + \frac{y_{i+1} - y_i}{t_{i+1} - t_i} (t - t_i)
\]

within the time intervals \( (t_i, t_{i+1}], i = 0, \ldots, 3, t_0 = 0 \). To generate a step change in the output signal, it is necessary to define two nodes in the same time instant (i.e. \( t_i = t_{i+1} \)). The generation ends when time \( t_4 \) is reached or when time \( t_i \) is reached and the following node precedes the active one (i.e. \( t_{i+1} < t_i \)). The output holds its final value afterwards. But for the RPT parameter set to on, instead of holding the final value, the block returns to its initial state \( y_0 \), the internal block timer is set to 0 and the sequence is generated repeatedly. This can be used to generate square or sawtooth functions. The generation can also be prematurely terminated by the RUN input signal set to off. In that case the block returns to its initial state \( y_0 \), the internal block timer is set to 0 and is \( = 0 \) becomes the active time interval.

**Input**

RUN Enable execution, run the analog sequence generation Bool

**Outputs**

\( y \) Analog output of the block Double (F64)
\( is \) Index of the active time interval Long (I32)

**Parameters**

\( y_0 \) Initial output value Double (F64)
\( t_1 \) Node 1 time Double (F64)
\( y_1 \) Node 1 value Double (F64)
\( t_2 \) Node 2 time Double (F64)
\( y_2 \) Node 2 value Double (F64)
t3  Node 3 time  0.2.0  Double (F64)
y3  Node 3 value  0.1.0  Double (F64)
t4  Node 4 time  0.2.0  Double (F64)
y4  Node 4 value  Double (F64)
RPT Repeating sequence  
   off ... Disabled  
   on ...  Enabled
**BINS – Controlled binary sequence generator**

Block Symbol  

```
  START Y
  BINS
```

**Licence:**  

STANDARD

**Function Description**

The **BINS** block generates a binary sequence at the **Y** output, similarly to the **BIS** block. The binary sequence is given by the block parameters:

- The initial value of the output is given by the **Y0** parameter.
- Whenever a rising edge (off → on) occurs at the **START** input (even when a binary sequence is being generated), the internal timer of the block is set to 0 and started.
- Whenever a rising edge occurs at the **START** input, the output **Y** is set to **Y0**.
- The output value is inverted at time instants **t1**, **t2**, ..., **t8** (off → on, on → off).
- For **RPT** = off, the last switching of the output occurs at time **t_i** where **t_i+1 = 0** and the output then holds its value until another rising edge (off → on) occurs at the **START** input.
- For **RPT** = on, instead of switching the output for the last time, the block returns to its initial state, the **Y** output is set to **Y0**, the internal block timer is set to 0 and started. As a result, the binary sequence is generated repeatedly.

On the contrary to the **BIS** block the changes in parameters **t1**...**t8** are accepted only when a rising edge occurs at the **START** input.

The switching times are internally rounded to the nearest integer multiple of the execution period, which may result in e.g. disappearing of very thin pulses (< **T_E/2**) or melting successive thin pulses into one thick pulse. Therefore it is strongly recommended to use integer multiples of the execution period as the switching times.

**Input**

<table>
<thead>
<tr>
<th>START</th>
<th>Starting signal (rising edge)</th>
<th>Bool</th>
</tr>
</thead>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Y</th>
<th>Logical output of the block</th>
<th>Bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>is</td>
<td>Index of the active time interval</td>
<td>Long (132)</td>
</tr>
<tr>
<td>Parameters</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>Y0</td>
<td>Initial output value</td>
<td>off ... Disabled/false  on ... Enabled/true</td>
</tr>
<tr>
<td>t1</td>
<td>Switching time 1 [s]</td>
<td>↓0.0 @1.0</td>
</tr>
<tr>
<td>t2</td>
<td>Switching time 2 [s]</td>
<td>↓0.0 @2.0</td>
</tr>
<tr>
<td>t3</td>
<td>Switching time 3 [s]</td>
<td>↓0.0 @3.0</td>
</tr>
<tr>
<td>t4</td>
<td>Switching time 4 [s]</td>
<td>↓0.0 @4.0</td>
</tr>
<tr>
<td>t5</td>
<td>Switching time 5 [s]</td>
<td>↓0.0 @5.0</td>
</tr>
<tr>
<td>t6</td>
<td>Switching time 6 [s]</td>
<td>↓0.0 @6.0</td>
</tr>
<tr>
<td>t7</td>
<td>Switching time 7 [s]</td>
<td>↓0.0 @7.0</td>
</tr>
<tr>
<td>t8</td>
<td>Switching time 8 [s]</td>
<td>↓0.0 @8.0</td>
</tr>
<tr>
<td>RPT</td>
<td>Repeating sequence</td>
<td>off ... Disabled  on ... Enabled</td>
</tr>
</tbody>
</table>
**BIS – Binary sequence generator**

Block Symbol

![Block Symbol Image]

**Licence:** STANDARD

**Function Description**

The BIS block generates a binary sequence at the Y output. The sequence is given by the block parameters.

- The initial value of the output is given by the Y0 parameter.
- The internal timer of the block is set to 0 when the block initializes.
- The internal timer of the block is immediately started when the block initializes.
- The output value is inverted at time instants t1, t2, ..., t8 (off→on, on→off).
- For BPT = off, the last switching of the output occurs at time t_i, where t_{i+1} = 0 and the output then holds its value indefinitely.
- For BPT = on, instead of switching the output for the last time, the block returns to its initial state, the Y output is set to Y0, the internal block timer is set to 0 and started. As a result, the binary sequence is generated repeatedly.

All the parameters t1...t8 can be changed in runtime and all changes are immediately accepted.

The switching times are internally rounded to the nearest integer multiple of the execution period, which may result in e.g. disappearing of very thin pulses (< T_S/2) or melting successive thin pulses into one thick pulse. Therefore it is strongly recommended to use integer multiples of the execution period as the switching times.

See also the BINS block, which allows for triggering the sequence by external signal.

**Outputs**

<table>
<thead>
<tr>
<th>Y</th>
<th>Logical output of the block</th>
<th>Bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>is</td>
<td>Index of the active time interval</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Y0</th>
<th>Initial output value</th>
<th>Bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Disabled/False</td>
<td>on</td>
</tr>
<tr>
<td>t1</td>
<td>Switching time 1 [s]</td>
<td>(0.0 \leq 1.0)</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>$t_2$</td>
<td>Switching time 2 [s]</td>
<td>$\downarrow0.0 \uparrow2.0$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>Switching time 3 [s]</td>
<td>$\downarrow0.0 \uparrow3.0$</td>
</tr>
<tr>
<td>$t_4$</td>
<td>Switching time 4 [s]</td>
<td>$\downarrow0.0 \uparrow4.0$</td>
</tr>
<tr>
<td>$t_5$</td>
<td>Switching time 5 [s]</td>
<td>$\downarrow0.0 \uparrow5.0$</td>
</tr>
<tr>
<td>$t_6$</td>
<td>Switching time 6 [s]</td>
<td>$\downarrow0.0 \uparrow6.0$</td>
</tr>
<tr>
<td>$t_7$</td>
<td>Switching time 7 [s]</td>
<td>$\downarrow0.0 \uparrow7.0$</td>
</tr>
<tr>
<td>$t_8$</td>
<td>Switching time 8 [s]</td>
<td>$\downarrow0.0 \uparrow8.0$</td>
</tr>
<tr>
<td>RPT</td>
<td>Repeating sequence</td>
<td></td>
</tr>
</tbody>
</table>

- off ... Disabled
- on ... Enabled
**MP – Manual pulse generator**

Block Symbol

---

**Function Description**

The MP block generates a pulse of width `pwidth` when a rising edge occurs at the BSTATE parameter (off → on). The algorithm immediately reverts the BSTATE parameter back to off (BSTATE stands for a shortly pressed button). If repetition is enabled (RPTF = on), it is possible to extend the pulse by repeated setting the BSTATE parameter to on. When repetition is disabled, the parameter BSTATE is not taken into account during generation of a pulse, i.e. the output pulses have always the specified width of `pwidth`.

The MP block reacts only to rising edge of the BSTATE parameter, therefore it cannot be used for generating a pulse immediately at the start of the REXYGEN system executive. Use the BIS block for such a purpose.

**Output**

- **Y** Logical output of the block

**Parameters**

- **pwidth** Pulse width [s]  
  
  - **BSTATE** Output pulse activation
    - off ... No action  
    - on ... Generate output pulse  
  
  - **RPTF** Allow pulse extension
    - off ... Disabled  
    - on ... Enabled

---

Licence: STANDARD
PRBS – Pseudo-random binary sequence generator

Block Symbol

Function Description

The PRBS block generates a pseudo-random binary sequence. The figure below displays how the sequence is generated.

The initial and final values of the sequence are val0. The sequence starts from this value when rising edge occurs at the START input (off→on), the output y is immediately switched to the valhi value. The generator then switches the output to the other limit value with the period of swper seconds and the probability of switching swprob. After seqt seconds the output is set back to val0. A waitt-second period follows to allow the settling of the controlled system response. Only then it is possible to start a new sequence. It is possible to terminate the sequence prematurely by the BRK = on input when necessary.

Inputs

| START | Starting signal (rising edge) | Bool |
| BRK   | Termination signal            | Bool |

Outputs

| y      | Generated pseudo-random binary sequence | Double (F64) |
| BSY    | Busy flag                               | Bool |

Parameters

<p>| val0   | Initial and final value                | Double (F64) |
| valhi  | Upper level of the y output            | 0.0           Double (F64) |
| vallo  | Lower level of the y output            | -1.0          Double (F64) |</p>
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Default</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>swper</td>
<td>Period of random output switching [s]</td>
<td>1.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>swprob</td>
<td>Probability of switching</td>
<td>0.0-1.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>seqt</td>
<td>Length of the sequence [s]</td>
<td>0.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>waitt</td>
<td>Settling period [s]</td>
<td>0.0</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
SG, SGI – Signal generators

Block Symbols

Function Description
The SG and SGI blocks generate periodic signals of chosen type (isig parameter): sine wave, square, sawtooth and white noise with uniform distribution. The amplitude and frequency of the output signal y are given by the amp and freq parameter respectively. The output y can have a phase shift of phase ∈ (0, 2π) in the deterministic signals (isig ∈ {1,2,3}).

The SGI block allows synchronization of multiple generators using the RUN and SYN inputs. The RUN parameter must be set to on to enable the generator, the SYN input synchronizes the generators during the output signal generation.

Inputs
- **RUN**: Enable execution, run the binary sequence generation
  - Value: Bool

- **SYN**: Synchronization signal
  - Value: Bool

Output
- **y**: Analog output of the block
  - Value: Double (F64)

Parameters
- **isig**: Generated signal type
  - Value: Long (I32)
    - 1 ..... Sine wave
    - 2 ..... Symmetrical rectangular signal
    - 3 ..... Sawtooth signal
    - 4 ..... White noise with uniform distribution

- **amp**: Amplitude of the generated signal
  - Value: Double (F64)
    - 0.0

- **freq**: Frequency of the generated signal
  - Value: Double (F64)
    - 0.0

- **phase**: Phase shift of the generated signal
  - Value: Double (F64)
    - 0.0

- **offset**: Value added to the generated signal
  - Value: Double (F64)
    - 0.0

- **ifrunit**: Frequency units
  - Value: Long (I32)
    - 1 ..... Hz
    - 2 ..... rad/s

- **iphunit**: Phase shift units
  - Value: Long (I32)
    - 1 ..... degrees
    - 2 ..... radians
Chapter 7

REG – Function blocks for control

Contents

ARLY – Advance relay .................................................. 163
FLCU – Fuzzy logic controller unit ................................. 164
FRID – Frequency response identification ........................ 166
I3PM – Identification of a three parameter model .............. 168
LC – Lead compensator .................................................. 170
LLC – Lead-lag compensator ......................................... 171
MCU – Manual control unit ........................................... 172
PIDAT – PID controller with relay autotuner .................... 174
PIDE – PID controller with defined static error ................. 177
PIDGS – PID controller with gain scheduling .................... 179
PIDMA – PID controller with moment autotuner ............... 181
PIDU – PID controller unit ............................................. 187
PIDUI – PID controller unit with variable parameters ........... 190
POUT – Pulse output ..................................................... 192
PRGM – Setpoint programmer ........................................ 193
PSMPC – Pulse-step model predictive controller ................. 195
PWM – Pulse width modulation ..................................... 199
RLY – Relay with hysteresis ......................................... 201
SAT – Saturation with variable limits ............................... 202
SC2FA – State controller for 2nd order system with frequency autotuner ............................................................. 204
SCU – Step controller with position feedback ..................... 211
SCUV – Step controller unit with velocity input .................. 214
SELU – Controller selector unit ...................................... 217
SMHCC – Sliding mode heating/cooling controller ............... 218
SMHCCA – Sliding mode heating/cooling controller with autotuner 222
SWU – Switch unit ...................................................... 229
ARLY – Advance relay

Block Symbol

Licence: **STANDARD**

Function Description

The ARLY block is a modification of the RLY block, which allows lowering the amplitude of steady state oscillations in relay feedback control loops. The block transforms the input signal \( u \) to the output signal \( y \) according to the diagram below.

\[ \begin{align*}
\text{Input} & \quad \text{Output} \\
\text{Parameter} & \quad \text{Description} & \quad \text{Default Value} & \quad \text{Type} \\
\text{ap} & \quad \text{Value of the y output in the "On" state} & \quad 1.0 & \quad \text{Double (F64)} \\
\text{an} & \quad \text{Value of the y output in the "Off" state} & \quad -1.0 & \quad \text{Double (F64)} \\
\text{y0} & \quad \text{Initial output value} & \quad & \quad \text{Double (F64)} \\
\end{align*} \]
FLCU – Fuzzy logic controller unit

Block Symbol

Function Description

The FLCU block implements a simple fuzzy logic controller with two inputs and one output. Introduction to fuzzy logic problems can be found in [3].

The output is defined by trapezoidal membership functions of linguistic terms of the \( u \) and \( v \) inputs, impulse membership functions of linguistic terms of the \( y \) output and inference rules in the form

\[
\text{If (} u \text{ is } U_i \text{) AND (} v \text{ is } V_j \text{), then (} y \text{ is } Y_k \text{),}
\]

where \( U_i, i = 1, \ldots, n_u \) are the linguistic terms of the \( u \) input; \( V_j, j = 1, \ldots, n_v \) are the linguistic terms of the \( v \) input and \( Y_k, k = 1, \ldots, n_y \) are the linguistic terms of the \( y \) output. Trapezoidal (triangular) membership functions of the \( u \) and \( v \) inputs are defined by four numbers as depicted below.

Not all numbers \( x_1, \ldots, x_4 \) are mutually different in triangular functions. The matrices of membership functions of the \( u \) and \( v \) input are composed of rows \([x_1, x_2, x_3, x_4]\). The dimensions of matrices \( m_u \) and \( m_v \) are \((n_u \times 4)\) and \((n_v \times 4)\) respectively.

The impulse 1st order membership functions of the \( y \) output are defined by the triplet

\[
y_k, a_k, b_k,
\]

where \( y_k \) is the value assigned to the linguistic term \( Y_k, k = 1, \ldots, n_y \) in the case of \( a_k = b_k = 0 \). If \( a_k \neq 0 \) and \( b_k \neq 0 \), then the term \( Y_k \) is assigned the value of \( y_k + a_k u + b_k v \).

The output membership function matrix \( m_y \) has a dimension of \((n_y \times 3)\) and contains the rows \([y_k, a_k, b_k], k = 1, \ldots, n_y\). The set of inference rules is also a matrix whose rows are \([i_l, j_l, k_l, w_l], l = 1, \ldots, n_r\), where \( i_l, j_l \) and \( k_l \) defines a particular linguistic term of the \( u \) and \( v \) inputs and \( y \) output respectively. The number \( w_l \) defines the weight of the rule in percents \( w_l \in \{0, 1, \ldots, 100\} \). It is possible to suppress or emphasize a particular inference rule if necessary.
Inputs

\( u \) First analog input of the block \( \text{Double (F64)} \)

\( v \) Second analog input of the block \( \text{Double (F64)} \)

Outputs

\( y \) Analog output of the block \( \text{Double (F64)} \)

\( ir \) Dominant rule \( \text{Long (I32)} \)

\( wr \) Degree of truth of the dominant rule \( \text{Double (F64)} \)

Parameters

\( \text{umax} \) Upper limit of the u input \( \circ 1.0 \) \( \text{Double (F64)} \)

\( \text{umin} \) Lower limit of the u input \( \circ -1.0 \) \( \text{Double (F64)} \)

\( \text{nu} \) Number of membership functions of the input u \( \downarrow 1 \uparrow 25 \circ 3 \) \( \text{Long (I32)} \)

\( \text{vmax} \) Upper limit of the v input \( \circ 1.0 \) \( \text{Double (F64)} \)

\( \text{vmin} \) Lower limit of the v input \( \circ -1.0 \) \( \text{Double (F64)} \)

\( \text{nv} \) Number of membership functions of the input v \( \downarrow 1 \uparrow 25 \circ 3 \) \( \text{Long (I32)} \)

\( \text{ny} \) Number of membership functions of the output y \( \downarrow 1 \uparrow 100 \circ 3 \) \( \text{Long (I32)} \)

\( \text{nr} \) Number of inference rules \( \downarrow 1 \uparrow 25 \circ 3 \) \( \text{Long (I32)} \)

\( \text{mfu} \) Matrix of membership functions of the input u \( \text{Double (F64)} \)

\( \begin{bmatrix} -1 & -1 & 0; & -1 & 0 & 0; & 0 & 1 & 1 \end{bmatrix} \)

\( \text{mfv} \) Matrix of membership functions of the input v \( \text{Double (F64)} \)

\( \begin{bmatrix} -1 & -1 & 0; & -1 & 0 & 0; & 0 & 1 & 1 \end{bmatrix} \)

\( \text{sty} \) Matrix of membership functions of the output y \( \text{Double (F64)} \)

\( \begin{bmatrix} -1 & 0; & 0 & 0; & 1 & 0 \end{bmatrix} \)

\( \text{rls} \) Matrix of inference rules \( \text{Byte (U8)} \)

\( \begin{bmatrix} 1 & 2 & 3 & 100; & 1 & 1 & 100; & 1 & 0 & 3 & 100 \end{bmatrix} \)
FRID – * Frequency response identification

Block Symbol

Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

- **dv**: Feedforward control variable
- **pv**: Process variable
- **ID**: Start the tuning experiment
- **HLD**: Hold
- **BRK**: Stop the tuning experiment

Parameters

- **ubias**: Static component of the exciting signal
- **ump**: Amplitude of the exciting signal
- **wb**: Frequency interval lower limit [rad/s]
- **wf**: Frequency interval higher limit [rad/s]
- **isweep**: Frequency sweeping mode
  - 1 . . . Logarithmic
  - 2 . . . Linear
- **cp**: Sweeping Rate
- **iavg**: Number of values for averaging
- **obw**: Observer bandwidth
  - 1 . . . LOW
  - 2 . . . NORMAL
  - 3 . . . HIGH
- **stme**: Settling period [s]

Licence: ADVANCED
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Default</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>umax</td>
<td>Maximum generator amplitude</td>
<td>1.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>thdmin</td>
<td>Minimum demanded THD threshold</td>
<td>0.1</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>adapt_rc</td>
<td>Maximum rate of amplitude variation</td>
<td>0.001</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>pv_max</td>
<td>Maximum desired process value</td>
<td>1.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>pv_sat</td>
<td>Maximum allowed process value</td>
<td>2.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>ADAPT_EN</td>
<td>Enable automatic amplitude adaptation</td>
<td>on</td>
<td>Bool</td>
</tr>
<tr>
<td>imode</td>
<td>Measurement mode</td>
<td>1</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>

1 . . . . . Manual specification of frequency points
2 . . . . . Linear series of n mw points in the interval < wb;wf >
3 . . . . . Logarithmic series of n mw points in the interval < wb;wf >
4 . . . . . Automatic detection of important frequencies (N/A)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Default</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>nwm</td>
<td>Number of frequency response point for automatic mode</td>
<td></td>
<td>Long (I32)</td>
</tr>
<tr>
<td>wm</td>
<td>Frequency measurement points for manual meas. mode [array of [rad/s]]</td>
<td>[2.0 4.0 6.0 8.0]</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Default</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>mv</td>
<td>Manipulated variable (controller output)</td>
<td></td>
<td>Double (F64)</td>
</tr>
<tr>
<td>SAT</td>
<td>Saturation flag</td>
<td></td>
<td>Bool</td>
</tr>
<tr>
<td>IDBSY</td>
<td>Tuner busy flag</td>
<td></td>
<td>Bool</td>
</tr>
<tr>
<td>w</td>
<td>Actual frequency [rad/s]</td>
<td></td>
<td>Double (F64)</td>
</tr>
<tr>
<td>xres</td>
<td>Real part of frequency response (sweeping)</td>
<td></td>
<td>Double (F64)</td>
</tr>
<tr>
<td>xims</td>
<td>Imaginary part of frequency response (sweeping)</td>
<td></td>
<td>Double (F64)</td>
</tr>
<tr>
<td>xrem</td>
<td>Real part of frequency response (measurement)</td>
<td></td>
<td>Double (F64)</td>
</tr>
<tr>
<td>ximm</td>
<td>Imaginary part of frequency response (measurement)</td>
<td></td>
<td>Double (F64)</td>
</tr>
<tr>
<td>epv</td>
<td>Estimated process value</td>
<td></td>
<td>Double (F64)</td>
</tr>
<tr>
<td>IDE</td>
<td>Error indicator</td>
<td></td>
<td>Bool</td>
</tr>
<tr>
<td>iIDE</td>
<td>Error code</td>
<td></td>
<td>Long (I32)</td>
</tr>
<tr>
<td>A0</td>
<td>Estimated DC value</td>
<td></td>
<td>Double (F64)</td>
</tr>
<tr>
<td>A1</td>
<td>Estimated 1st harmonics amplitude</td>
<td></td>
<td>Double (F64)</td>
</tr>
<tr>
<td>A2</td>
<td>Estimated 2nd harmonics amplitude</td>
<td></td>
<td>Double (F64)</td>
</tr>
<tr>
<td>A3</td>
<td>Estimated 3rd harmonics amplitude</td>
<td></td>
<td>Double (F64)</td>
</tr>
<tr>
<td>A4</td>
<td>Estimated 4th harmonics amplitude</td>
<td></td>
<td>Double (F64)</td>
</tr>
<tr>
<td>A5</td>
<td>Estimated 5th harmonics amplitude</td>
<td></td>
<td>Double (F64)</td>
</tr>
<tr>
<td>THD</td>
<td>Total harmonic distortion</td>
<td></td>
<td>Double (F64)</td>
</tr>
<tr>
<td>DAV</td>
<td>Data Valid</td>
<td></td>
<td>Bool</td>
</tr>
</tbody>
</table>
I3PM – Identification of a three parameter model

Block Symbol

Function Description

The I3PM block is based on the generalized moment identification method. It provides a three parameter model of the system.

Inputs

\begin{align*}
\text{u} & \quad \text{Input of the identified system} & \text{Double (F64)} \\
\text{y} & \quad \text{Output of the identified system} & \text{Double (F64)} \\
\text{u0} & \quad \text{Input steady state} & \text{Double (F64)} \\
\text{y0} & \quad \text{Output steady state} & \text{Double (F64)} \\
\text{RUN} & \quad \text{Execute identification} & \text{Bool} \\
\text{CLR} & \quad \text{Block reset} & \text{Bool} \\
\text{ips} & \quad \text{Meaning of the output signals} & \text{Long (I32)} \\
0 & \quad \text{FOPDT model} \\
& & \text{p1 \ldots \text{gain}} \\
& & \text{p2 \ldots \text{time delay}} \\
& & \text{p3 \ldots \text{time constant}} \\
1 & \quad \text{Moments of input and output} \\
& & \text{p1 \ldots \text{parameter } mu0} \\
& & \text{p2 \ldots \text{parameter } mu1} \\
& & \text{p3 \ldots \text{parameter } mu2} \\
& & \text{p4 \ldots \text{parameter } my0} \\
& & \text{p5 \ldots \text{parameter } my1} \\
& & \text{p6 \ldots \text{parameter } my2} \\
2 & \quad \text{Process moments} \\
& & \text{p1 \ldots \text{parameter } mp0} \\
& & \text{p2 \ldots \text{parameter } mp1} \\
& & \text{p3 \ldots \text{parameter } mp2} \\
3 & \quad \text{Characteristic numbers} \\
& & \text{p1 \ldots \text{parameter } } \kappa \\
& & \text{p2 \ldots \text{parameter } } \mu \\
& & \text{p3 \ldots \text{parameter } } \sigma^2 \\
& & \text{p4 \ldots \text{parameter } } \sigma
\end{align*}
Outputs

\[ p_i \quad \text{Identified parameters with respect to } \text{ips}, i = 1, \ldots, 8 \quad \text{Double (F64)} \]

BSY  Busy flag  Bool
RDY  Ready flag  Bool
E   Error flag  Bool
iE  Error code  Long (I32)

1 ..... Premature termination (RUN = off)
2 ..... mu0 = 0
3 ..... mp0 = 0
4 ..... \( \sigma^2 < 0 \)

Parameters

tident  Duration of identification [s]  \( \odot 100.0 \)  Double (F64)
irtype  Controller type (control law)  \( \odot 6 \)  Long (I32)

1 ..... D 3 ..... ID 5 ..... PD 7 ..... PID
2 ..... I 4 ..... P 6 ..... PI

ispeed  Desired closed loop speed  \( \odot 2 \)  Long (I32)

1 ..... Slow closed loop
2 ..... Normal (middle fast) closed loop
3 ..... Fast closed loop
LC – Lead compensator

Block Symbol Licence: STANDARD

Function Description

The LC block is a discrete simulator of derivative element

\[ C(s) = \frac{\text{td} \cdot s}{\text{td} \cdot s + 1}, \]

where \( \text{td} \) is the derivative constant and \( \text{nd} \) determines the influence of parasite 1st order filter. It is recommended to use \( 2 \leq \text{nd} \leq 10 \). If ISSF = on, then the state of the parasite filter is set to the steady value at the block initialization according to the input signal \( u \).

The exact discretization at the sampling instants is used for discretization of the \( C(s) \) transfer function.

Input

\( u \) Analog input of the block Double (F64)

Output

\( y \) Analog output of the block Double (F64)

Parameters

\( \text{td} \)  Derivative time constant \( \odot 1.0 \) Double (F64)
\( \text{nd} \)  Derivative filtering parameter \( \odot 10.0 \) Double (F64)
\( \text{ISSF} \)  Steady state at start-up Bool
  
  \( \text{off} \ldots \) Zero initial state
  \( \text{on} \ldots \) Initial steady state
LLC – Lead-lag compensator

Block Symbol

\[
C(s) = \frac{a \cdot \tau u \cdot s + 1}{\tau u \cdot s + 1},
\]

where \( \tau u \) is the denominator time constant and the time constant of numerator is an \( a \)-multiple of \( \tau u \) (\( a \cdot \tau u \)). If ISSF = on, then the state of the filter is set to the steady value at the block initialization according to the input signal \( u \).

The exact discretization at the sampling instants is used for discretization of the \( C(s) \) transfer function. The sampling period used for discretization is equivalent to the execution period of the LLC block.

Input

\( u \) Analog input of the block Double (F64)

Output

\( y \) Analog output of the block Double (F64)

Parameters

\( \tau u \) Time constant \( \odot 1.0 \) Double (F64)
\( a \) Numerator time constant coefficient Double (F64)
ISSF Steady state at start-up Bool
  off ... Zero initial state
  on ... Initial steady state
MCU – Manual control unit

Block Symbol

Licence: STANDARD

Function Description

The MCU block is suitable for manual setting of the numerical output value \( y \), e.g. a setpoint. In the local mode (LOC = on) the value is set using the buttons UP and DN. The rate of increasing/decreasing of the output \( y \) from the initial value \( y_0 \) is determined by the integration time constant \( t_m \) and pushing time of the buttons. After elpassing \( t_a \) seconds while a button is pushed, the rate is always multiplied by the factor \( q \) until the time \( t_f \) is elapsed. Optionally, the output \( y \) range can be constrained (SATF = on) by saturation limits lolim and hilim. If none of the buttons is pushed (UP = off and DN = off), the output \( y \) tracks the input value \( tv \). The tracking speed is controlled by the integration time constant \( tt \).

In the remote mode (LOC = off), the input \( rv \) is optionally saturated (SATF = on) and copied to the output \( y \). The detailed function of the block is depicted in the following diagram.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( tv )</td>
<td>Tracking variable</td>
</tr>
<tr>
<td>( UP )</td>
<td>The &quot;up&quot; signal</td>
</tr>
<tr>
<td>( DN )</td>
<td>The &quot;down&quot; signal</td>
</tr>
<tr>
<td>( rv )</td>
<td>Remote output value in the mode LOC = off</td>
</tr>
<tr>
<td>( LOC )</td>
<td>Local or remote mode</td>
</tr>
</tbody>
</table>
Output

| y     | Analog output of the block | Double (F64) |

Parameters

| tt    | Tracking time constant of the input tv | 1.0 | Double (F64) |
| tm    | Initial value of integration time constant | 100.0 | Double (F64) |
| y0    | Initial output value | | Double (F64) |
| q     | Multiplication quotient | 5.0 | Double (F64) |
| ta    | Interval after which the rate is changed [s] | 4.0 | Double (F64) |
| tf    | Interval after which the rate changes no more [s] | 8.0 | Double (F64) |
| SATF  | Saturation flag | | Bool |
|       | off ... Signal not limited | | |
|       | on .... Saturation limits active | | |
| hilim | Upper limit of the output signal | 1.0 | Double (F64) |
| lolim | Lower limit of the output signal | -1.0 | Double (F64) |
PIDAT – PID controller with relay autotuner

Block Symbol

Function Description

The PIDAT block has the same control function as the PIDU block. Additionally it is equipped with the relay autotuning function.

In order to perform the autotuning experiment, it is necessary to drive the system to approximately steady state (at a suitable working point), choose the type of controller to be autotuned (PI or PID) and activate the TUNE input by setting it to on. The controlled process is regulated by special adaptive relay controller in the experiment which follows. One point of frequency response is estimated from the data measured during the experiment. Based on this information the controller parameters are computed. The amplitude of the relay controller (the level of system excitation) and its hysteresis is defined by the amp and hys parameters. In case of hys=0 the hysteresis is determined automatically according to the measurement noise properties on the controlled variable signal. The signal TBSY is set to on during the tuning experiment. A successful experiment is indicated by and the controller parameters can be found on the outputs pk, pti, ptd, pnd and pb. The c weighting factor is assumed (and recommended) <=0. A failure during the experiment causes TE = on and the output ite provides further information about the problem. It is recommended to increase the amplitude amp in the case of error. The controller is equipped with a built-in function which decreases the amplitude when the deviation of output from the initial steady state exceeds the maxdev limit. The tuning experiment can be prematurely terminated by activating the TBRK input.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>dv</td>
<td>Feedforward control variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>sp</td>
<td>Setpoint variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>pv</td>
<td>Process variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>tv</td>
<td>Tracking variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>hv</td>
<td>Manual value</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>MAN</td>
<td>Manual or automatic mode</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off . . . Automatic mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on . . . Manual mode</td>
<td></td>
</tr>
</tbody>
</table>
TUNE  Start the tuning experiment  Bool
TBRK  Stop the tuning experiment  Bool

**Outputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>mv</td>
<td>Manipulated variable (controller output)</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>de</td>
<td>Deviation error</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>SAT</td>
<td>Saturation flag</td>
<td>Bool</td>
</tr>
<tr>
<td>TBSY</td>
<td>Tuner busy flag</td>
<td>Bool</td>
</tr>
<tr>
<td>TE</td>
<td>Tuning error</td>
<td>Bool</td>
</tr>
<tr>
<td>ite</td>
<td>Error code; expected time (in seconds) to finishing the tuning experiment</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>pk</td>
<td>Proposed controller gain</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>pti</td>
<td>Proposed integral time constant</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>ptd</td>
<td>Proposed derivative time constant</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>pnd</td>
<td>Proposed derivative component filtering</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>pb</td>
<td>Proposed weighting factor – proportional component</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>itype</td>
<td>Controller type (control law)</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>RACT</td>
<td>Reverse action flag</td>
<td>Bool</td>
</tr>
<tr>
<td>k</td>
<td>Controller gain $K$. By definition, the value 0 turns the controller off.</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>ti</td>
<td>Integral time constant $T_i$. The value 0 disables the integrating part</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>td</td>
<td>Derivative time constant $T_d$. The value 0 disables the derivative part</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>nd</td>
<td>Derivative filtering parameter $N$. The value 0 disables the derivative part</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
CHAPTER 7. REG - FUNCTION BLOCKS FOR CONTROL

b  Setpoint weighting – proportional part  \( \downarrow 0.0 \uparrow 1.0 \) Double (F64)
c  Setpoint weighting – derivative part  \( \downarrow 0.0 \) Double (F64)
tt Tracking time constant.  \( \downarrow 0.0 \uparrow 1.0 \) Double (F64)
hilim Upper limit of the controller output  \( \uparrow 1.0 \) Double (F64)
lolim Lower limit of the controller output  \( \downarrow -1.0 \) Double (F64)
ainf Type of a priori information  \( \uparrow 1 \) Long (I32)

1 . . . . No a priori information
2 . . . . A static process (process with integration)
3 . . . . Low order process
4 . . . . Static process + slow closed loop step response
5 . . . . Static process + middle fast (normal) closed loop step response
6 . . . . Static process + fast closed loop step response

k0 Static gain of the process (must be provided in case of \( \text{ainf} \neq 3, 4, 5 \))  \( \uparrow 0.1 \) Double (F64)
n1 Maximum number of half-periods for estimation of frequency response point  \( \uparrow 20 \) Long (I32)
mm Maximum number of half-periods for averaging  \( \uparrow 4 \) Long (I32)
amp Relay controller amplitude  \( \uparrow 0.1 \) Double (F64)
uhys Relay controller hysteresis  Double (F64)
nime Length of noise amplitude estimation period at the beginning of the tuning experiment [s]  \( \uparrow 5.0 \) Double (F64)
errap Termination value of the oscillation amplitude relative error  \( \uparrow 0.1 \) Double (F64)
aerrph Termination value of the absolute error in oscillation phase  \( \uparrow 1.0 \) Double (F64)
maxdev Maximal admissible deviation error from the initial steady state  \( \uparrow 1.0 \) Double (F64)

It is recommended not to change the parameters \( n1, mm, nime, \text{errap} \) and \( \text{aerrph} \).
PIDE – PID controller with defined static error

Block Symbol

Licence: ADVANCED

Function Description

The PIDE block is a basis for creating a modified PI(D) controller which differs from the standard PI(D) controller (the PIDU block) by having a finite static gain (in fact, the value \( \varepsilon \) which causes the saturation of the output is entered). In the simplest case it can work autonomously and provide the standard functionality of the modified PID controller with two degrees of freedom in the automatic (MAN = off) or manual mode (MAN = on).

If in automatic mode and if the saturation limits are not active, the controller implements a linear control law given by

\[
U(s) = \pm K \left[ \delta W(s) - Y(s) + \frac{1}{T_1 s + \beta} E(s) + \frac{T_d s}{N + 1} (cW(s) - Y(s)) \right] + Z(s),
\]

where

\[
\beta = \frac{K \varepsilon}{1 - K \varepsilon}
\]

\( U(s) \) is the Laplace transform of the manipulated variable \( \text{mv} \), \( W(s) \) is the Laplace transform of the setpoint \( \text{sp} \), \( Y(s) \) is the Laplace transform of the process variable \( \text{pv} \), \( E(s) \) is the Laplace transform of the deviation error, \( Z(s) \) is the Laplace transform of the feedforward control variable \( \text{dv} \) and \( K, T_1, T_d, N, \varepsilon \ (= \delta^p/100) \), \( \delta \) and \( c \) are the controller parameters. The sign of the right-hand side depends on the parameter \text{RACI}. The range of the manipulated variable \( \text{mv} \) (position controller output) is limited by parameters \( \text{hi} \text{lim}, \text{lo} \text{lim} \).

By connecting the output \( \text{mv} \) of the controller to the controller input \( \text{tv} \) and properly setting the tracking time constant \( \tau \) we obtain the bumpless operation of the controller in the case of the mode switching (manual, automatic) and also the correct operation of the controller when saturation of the output \( \text{mv} \) occurs (antiwindup).

In the manual mode (MAN = on), the input \( \text{hv} \) is copied to the output \( \text{mv} \) unless saturated. In this mode the inner controller state tracks the signal connected to the \( \text{tv} \) input so the successive switching to the automatic mode is bumpless. But the tracking is not precise for \( \varepsilon > 0 \).
CHAPTER 7. REG - FUNCTION BLOCKS FOR CONTROL

Inputs

dv  Feedforward control variable            Double (F64)
sp  Setpoint variable                        Double (F64)
pv  Process variable                         Double (F64)
tv  Tracking variable                        Double (F64)
hv  Manual value                             Double (F64)
MAN Manual or automatic mode
     off ... Automatic mode
     on    . . . . Manual mode

Outputs

mv  Manipulated variable (controller output) Double (F64)
de  Deviation error                           Double (F64)
SAT Saturation flag                           Bool
     off ... The controller implements a linear control law
     on    . . . . The controller output is saturated

Parameters

irtype Controller type (control law)          06 Long (I32)
     1 . . . . D  4 . . . . P  7 . . . . PID
     2 . . . . I  5 . . . . PD
     3 . . . . ID  6 . . . . PI
RACT Reverse action flag                      Bool
     off ... Higher mv \rightarrow higher pv
     on    . . . . Higher mv \rightarrow lower pv
k   Controller gain $K$                         \(0.0 \sim 1.0\) Double (F64)
ti  Integral time constant $T_i$              \(0.0 \sim 4.0\) Double (F64)
td  Derivative time constant $T_d$             \(0.0 \sim 1.0\) Double (F64)
nd  Derivative filtering parameter $N$        \(0.0 \sim 10.0\) Double (F64)
b   Setpoint weighting - proportional part     \(0.0 \sim 1.0\) Double (F64)
c   Setpoint weighting - derivative part       \(0.0\) Double (F64)
tt  Tracking time constant. No meaning for controllers without integrator. \(0.0 \sim 1.0\) Double (F64)
bp  Static error coefficient                   Double (F64)
hilim Upper limit of the controller output    \(1.0\) Double (F64)
lolim Lower limit of the controller output     \(-1.0\) Double (F64)
PIDGS – PID controller with gain scheduling

Block Symbol Licence: ADVANCED

Function Description

The functionality of the PIDGS block is completely equivalent to the PIDU block. The only difference is that the PIDGS block has at most six sets of basic PID controller parameters and allow bumpless switching of these sets by the ip (parameter set index) or vp inputs. In the latter case it is necessary to set GSCF = on and provide an array of threshold values thrsha. The following rules define the active parameter set: the set 0 is active for \( vp < \text{thrsha}(0) \), the set 1 for \( \text{thrsha}(1) < vp < \text{thrsha}(2) \) etc. till the set 5 for \( \text{thrsha}(5) < vp \). The index of the active parameter set is available at the kp output.

Inputs

\[
\begin{align*}
dv & \quad \text{Feedforward control variable} & \text{Double (F64)} \\
dp & \quad \text{Setpoint variable} & \text{Double (F64)} \\
dp & \quad \text{Process variable} & \text{Double (F64)} \\
dv & \quad \text{Tracking variable} & \text{Double (F64)} \\
hv & \quad \text{Manual value} & \text{Double (F64)} \\
\text{MAN} & \quad \text{Manual or automatic mode} & \text{Bool} \\
\text{off} & \quad \text{Automatic mode} \\
\text{on} & \quad \text{Manual mode} \\
\text{IH} & \quad \text{Integrator hold} & \text{Bool} \\
\text{off} & \quad \text{Integration enabled} \\
\text{on} & \quad \text{Integration disabled} \\
ip & \quad \text{Parameter set index} & \text{\(\downarrow 0 \uparrow 5\)} \text{ Long (I32)} \\
vp & \quad \text{Switching analog signal} & \text{Double (F64)}
\end{align*}
\]

Outputs

\[
\begin{align*}
\text{mv} & \quad \text{Manipulated variable (controller output)} & \text{Double (F64)} \\
\text{dmsv} & \quad \text{Controller velocity output (difference)} & \text{Double (F64)} \\
de & \quad \text{Deviation error} & \text{Double (F64)} \\
\text{SAT} & \quad \text{Saturation flag} & \text{Bool} \\
\text{off} & \quad \text{The controller implements a linear control law} \\
\text{on} & \quad \text{The controller output is saturated}
\end{align*}
\]
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>kp</td>
<td>Active parameter set index</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>hilim</td>
<td>Upper limit of the controller output</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>lolim</td>
<td>Lower limit of the controller output</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>dz</td>
<td>Dead zone</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>icotype</td>
<td>Controller output type</td>
<td>Long (I32)</td>
</tr>
<tr>
<td></td>
<td>1 .... Analog output</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 .... Pulse width modulation (PWM)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 .... Step controller unit with position feedback (SCU)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 .... Step controller unit without position feedback (SCU)</td>
<td></td>
</tr>
<tr>
<td>npars</td>
<td>Number of controller parameter sets</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>GSCF</td>
<td>Switch parameters by analog signal vp</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... Index-based switching</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... Analog signal based switching</td>
<td></td>
</tr>
<tr>
<td>hys</td>
<td>Hysteresis for controller parameters switching</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>irtypea</td>
<td>Vector of controller types (control laws)</td>
<td>Byte (U8)</td>
</tr>
<tr>
<td></td>
<td>1 .... D 4 .... P 7 .... PID</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 .... I 5 .... PD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 .... ID 6 .... PI</td>
<td></td>
</tr>
<tr>
<td>RACTA</td>
<td>Vector of reverse action flags</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>0 .... Higher mv → higher pv</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 .... Higher mv → lower pv</td>
<td></td>
</tr>
<tr>
<td>ka</td>
<td>Vector of controller gains $K$</td>
<td>Double (F64)</td>
</tr>
<tr>
<td></td>
<td>$[1.0 1.0 1.0 1.0 1.0 1.0]$</td>
<td></td>
</tr>
<tr>
<td>tia</td>
<td>Vector of integral time constants $T_i$</td>
<td>Double (F64)</td>
</tr>
<tr>
<td></td>
<td>$[4.0 4.0 4.0 4.0 4.0 4.0]$</td>
<td></td>
</tr>
<tr>
<td>tda</td>
<td>Vector of derivative time constants $T_d$</td>
<td>Double (F64)</td>
</tr>
<tr>
<td></td>
<td>$[1.0 1.0 1.0 1.0 1.0 1.0]$</td>
<td></td>
</tr>
<tr>
<td>nda</td>
<td>Vector of derivative filtering parameters $N$</td>
<td>Double (F64)</td>
</tr>
<tr>
<td></td>
<td>$[10.0 10.0 10.0 10.0 10.0 10.0]$</td>
<td></td>
</tr>
<tr>
<td>ba</td>
<td>Setpoint weighting factors – proportional part</td>
<td>Double (F64)</td>
</tr>
<tr>
<td></td>
<td>$[1.0 1.0 1.0 1.0 1.0 1.0]$</td>
<td></td>
</tr>
<tr>
<td>ca</td>
<td>Setpoint weighting factors – derivative part</td>
<td>Double (F64)</td>
</tr>
<tr>
<td></td>
<td>$[0.0 0.0 0.0 0.0 0.0 0.0]$</td>
<td></td>
</tr>
<tr>
<td>tta</td>
<td>Vector of tracking time constants</td>
<td>Double (F64)</td>
</tr>
<tr>
<td></td>
<td>$[1.0 1.0 1.0 1.0 1.0 1.0]$</td>
<td></td>
</tr>
<tr>
<td>thrsha</td>
<td>Vector of thresholds for switching the parameters</td>
<td>Double (F64)</td>
</tr>
<tr>
<td></td>
<td>$[0.1 0.2 0.3 0.4 0.5 0]$</td>
<td></td>
</tr>
</tbody>
</table>
PIDMA – PID controller with moment autotuner

Block Symbol

Licence: AUTOTUNING

Function Description

The PIDMA block has the same control function as the PIDU block. Additionally it is equipped with the moment autotuning function.

In the automatic mode (MAN = off), the block PIDMA implements the PID control law with two degrees of freedom in the form

\[ U(s) = \pm K \left\{ bW(s) - Y(s) + \frac{1}{T_1s} [W(s) - Y(s)] + \frac{T_d s}{N s + 1} [cW(s) - Y(s)] \right\} + Z(s) \]

where \( U(s) \) is Laplace transform of the manipulated variable \( \text{mv} \), \( W(s) \) is Laplace transform of the setpoint variable \( \text{sp} \), \( Y(s) \) is Laplace transform of the process variable \( \text{pv} \), \( Z(s) \) is Laplace transform of the feedforward control variable \( \text{dv} \) and \( K \), \( T_1 \), \( T_d \), \( N \), \( b \) and \( c \) are the parameters of the controller. The sign of the right hand side depends on the parameter RACT. The range of the manipulated variable \( \text{mv} \) (position controller output) is limited by parameters \( \text{hilim, lolim} \). The parameter \( \text{dz} \) determines the dead zone in the integral part of the controller. The integral part of the control law can be switched off and fixed on the current value by the integrator hold input \( \text{IH} = \text{on} \). For the proper function of the controller it is necessary to connect the output \( \text{mv} \) of the controller to the controller input \( \text{tv} \) and properly set the tracking time constant \( \text{tt} \).

The rule of thumb for a PID controller is \( \text{tt} \approx \sqrt{T_1 T_d} \). For a PI controller the formula is \( \text{tt} \approx T_1 / 2 \). In this way we obtain the bumpless operation of the controller in the case of the mode switching (manual, automatic) and also the correct operation of the controller when saturation of the output \( \text{mv} \) occurs (antwindup).

The additional outputs \( \text{dmv, de and SAT} \) generate the velocity output (difference of \( \text{mv} \)), deviation error and saturation flag, respectively.

If the PIDMA block is connected with the block SCUV to configure the 3-point step controller without the positional feedback, then the parameter icotype must be set to 4 and the meaning of the outputs \( \text{mv} \) and \( \text{dmv} \) and SAT is modified in the following way: \( \text{mv} \) and \( \text{dmv} \) give the PD part and difference of I part of the control law, respectively, and...
SAT provides the information for the SCUV block whether the deviation error is less than the dead zone \( dz \) in the automatic mode. In this case, the setpoint weighting factor \( c \) should be zero.

In the manual mode (\( MAN = \text{on} \)), the input \( hv \) is copied to the output \( mv \) unless saturated. The overall control function of the PIDMA block is quite clear from the following diagram:

The block PIDMA extends the control function of the standard PID controller by the built-in autotuning feature. Before start of the autotuner the operator have to reach the steady state of the process at a suitable working point (in manual or automatic mode) and specify the required type of the controller \( \text{ittype} \) (PI or PID) and other tuning parameters (\( \text{isinf, DGC, tdg, tn, amp, dy} \) and \( \text{ispeed} \)). The identification experiment is started by the input \( \text{TUNE} \) (input \( \text{TBRK} \) finishes the experiment). In this mode (\( \text{TSY} = \text{on} \)), first of all the noise and possible drift gradient (\( \text{DGC = on} \)) are estimated during the user specified time (\( \text{tdg+tn} \)) and then the rectangle pulse is applied to the input of the process and the first three process moments are identified from the pulse response. The amplitude of the pulse is set by the parameter \( \text{amp} \). The pulse is finished when the process variable \( \text{pv} \) deviates from the steady value more than the \( \text{dy} \) threshold defines. The threshold is an absolute difference, therefore it is always a positive value. The duration of the tuning experiment depends on the dynamic behavior of the process. The remaining time to the end of the tuning is provided by the output \( \text{trem} \).

If the identification experiment is properly finished (\( \text{TE = off} \)) and the input \( \text{ips} \) is equal to zero, then the optimal parameters immediately appear on the block outputs \( \text{pk, pti, pt, pd, pb, pc} \). In the opposite case (\( \text{TE = on} \)) the output \( \text{ite} \) specifies the experiment error more closely. Other values of the \( \text{ips} \) input are reserved for custom specific purposes.

The function of the autotuner is illustrated in the following picture.
During the experiment, the output \textit{ite} indicates the autotuner phases. In the phase of estimation of the response decay rate ($\textit{ite} = -4$) the tuning experiment may be finished manually before its regular end. In this case the controller parameters are designed but the potential warning is indicated by setting the output $\textit{ite}=100$.

At the end of the experiment (TBSY on→off), the function of the controller depends on the current controller mode. If the TAFF = on the designed controller parameters are immediately accepted.

\textbf{Inputs}

\begin{itemize}
  \item \textbf{dv} \hspace{1cm} \text{Feedforward control variable} \\
  \item \textbf{sp} \hspace{1cm} \text{Setpoint variable} \\
  \item \textbf{pv} \hspace{1cm} \text{Process variable} \\
  \item \textbf{tv} \hspace{1cm} \text{Tracking variable} \\
  \item \textbf{hv} \hspace{1cm} \text{Manual value} \\
  \item \textbf{MAN} \hspace{1cm} \text{Manual or automatic mode} \hspace{1cm} \text{Bool} \\
    \text{off} \hspace{1cm} \text{Automatic mode} \\
    \text{on} \hspace{1cm} \text{Manual mode} \\
  \item \textbf{IH} \hspace{1cm} \text{Integrator hold} \hspace{1cm} \text{Bool} \\
    \text{off} \hspace{1cm} \text{Integration enabled} \\
    \text{on} \hspace{1cm} \text{Integration disabled} \\
  \item \textbf{TUNE} \hspace{1cm} \text{Start the tuning experiment (off→on) or force transition to the next tuning phase (see the description of the ite output)} \hspace{1cm} \text{Bool} \\
  \item \textbf{TBK} \hspace{1cm} \text{Stop the tuning experiment} \hspace{1cm} \text{Bool} \\
  \item \textbf{TAFF} \hspace{1cm} \text{Tuning affirmation; determines the way the computed parameters are handled} \hspace{1cm} \text{Bool} \\
    \text{off} \hspace{1cm} \text{Parameters are only computed} \\
    \text{on} \hspace{1cm} \text{Parameters are set into the control law}
\end{itemize}
IPS

Meaning of the output signals pk, pti, ptd, pnd, pb and pc

0 .... Designed parameters k, ti, td, nd, b and c of the PID control law
1 .... Process moments: static gain (pk), resident time constant (pti), measure of the system response length (ptd)
2 .... Three-parameter first-order plus dead-time model: static gain (pk), dead-time (pti), time constant (ptd)
3 .... Three-parameter second-order plus dead-time model with double time constant: static gain (pk), dead-time (pti), time constant (ptd)
4 .... Estimated boundaries for manual fine-tuning of the PID controller (irtype = 7) gain k: upper boundary k_hi (pk), lower boundary k_lo (pti)
>99 ... Reserved for diagnostic purposes

Outputs

MV Manipulated variable (controller output) Double (F64)
D MV Controller velocity output (difference) Double (F64)
DE Deviation error Double (F64)
SAT Saturation flag
  off .... The controller implements a linear control law
  on .... The controller output is saturated

TBSY Tuner busy flag Bool
TE Tuning error
  off .... Autotuning successful
  on .... An error occurred during the experiment
ITE Error code Long (I32)

Tuning error codes (after the experiment):

0 .... No error or waiting for steady state
1 .... Too small pulse getdown threshold
2 .... Too large pulse amplitude
3 .... Steady state condition violation
4 .... Too small pulse amplitude
5 .... Peak search procedure failure
6 .... Output saturation occurred during experiment
7 .... Selected controller type not supported
8 .... Process not monotonous
9 .... Extrapolation failure
10 .... Unexpected values of moments (fatal)
11 .... Abnormal manual termination of tuning
12 .... Wrong direction of manipulated variable
100 .... Manual termination of tuning (warning)
Tuning phases codes (during the experiment):

0 .... Steady state reaching before the start of the experiment
-1 .... Drift gradient and noise estimation phase
-2 .... Pulse generation phase
-3 .... Searching the peak of system response
-4 .... Estimation of the system response decay rate

Remark about terminating the tuning phases
TUNE .... The rising edge of the TUNE input during the phases
-2, -3 and -4 causes the finishing of the current phase and transition to the next one (or finishing the experiment in the phase -4).

trem Estimated time to finish the tuning experiment [s]            Double (F64)
pk    Proposed controller gain $K$ (ips = 0)                      Double (F64)
pti   Proposed integral time constant $T_i$ (ips = 0)             Double (F64)
ptd   Proposed derivative time constant $T_d$ (ips = 0)           Double (F64)
pnd   Proposed derivative component filtering $N$ (ips = 0)       Double (F64)
pb    Proposed weighting factor – proportional component (ips = 0) Double (F64)
pc    Proposed weighting factor – derivative component (ips = 0)  Double (F64)

Parameters

<table>
<thead>
<tr>
<th>irtype</th>
<th>Controller type (control law)</th>
<th>06 Long (I32)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 .... D 4 .... P 7 .... PID</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 .... I 5 .... PD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 .... ID 6 .... PI</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RACT</th>
<th>Reverse action flag</th>
<th>Bool</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>off ... Higher mv \rightarrow higher pv</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... Higher mv \rightarrow lower pv</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>k</th>
<th>Controller gain $K$. By definition, the value 0 turns the controller off. Negative values are not allowed, use the RACT parameter for such a purpose.</th>
<th>Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ti</th>
<th>Integral time constant $T_i$. The value 0 disables the integrating part (the same effect as disabling it by the irtype parameter).</th>
<th>Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>td</th>
<th>Derivative time constant $T_d$. The value 0 disables the derivative part (the same effect as disabling it by the irtype parameter).</th>
<th>Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>nd</th>
<th>Derivative filtering parameter $N$. The value 0 disables the derivative part (the same effect as disabling it by the irtype parameter).</th>
<th>Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 0.0</td>
<td></td>
</tr>
</tbody>
</table>

| b      | Setpoint weighting – proportional part | \downarrow 0.0 \uparrow 2.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 | Double (F64) |

| c      | Setpoint weighting – derivative part | \downarrow 0.0 \uparrow 2.0 \downarrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 \downarrow 0.0 \uparrow 1.0 | Double (F64) |
Tracking time constant. The value 0 stands for an implicit value, which is \( T_i / 2 \) or \( \sqrt{T_i T_d} \) (see above) for controllers with integrating part. For controllers without integrating part, the value 0 disables tracking. If tracking is needed for a P or PD controller, it can be enabled by entering a positive value greater than the sampling time. It is not possible to turn off tracking for controllers with the integrating part (due to the windup effect).

\[
\downarrow 0.0 \rightarrow 1.0
\]

Upper limit of the controller output

\( \circ 1.0 \) Double (F64)

Lower limit of the controller output

\( \circ -1.0 \) Double (F64)

Dead zone

Double (F64)

Controller output type

\( \circ 1 \) Long (I32)

Controller type to be designed

\( \circ 6 \) Long (I32)

Type of a priori information

\( \circ 1 \) Long (I32)

Drift gradient compensation

\( \circ \text{on} \) Bool

Drift gradient estimation time [s]

\( \circ 60.0 \) Double (F64)

Length of noise estimation period [s]

\( \circ 5.0 \) Double (F64)

Tuning pulse amplitude

\( \circ 0.5 \) Double (F64)

Tuning pulse get down threshold (absolute difference from the steady pv value)

\( \downarrow 0.0 \rightarrow 0.1 \)

Desired closed loop speed

\( \circ 2 \) Long (I32)

PID controller form

\( \circ 1 \) Long (I32)
**PIDU – PID controller unit**

**Block Symbol**

![Block Symbol](image)

**Licence:** STANDARD

**Function Description**

The PIDU block is a basic block for creating a complete PID controller (or P, I, PI, PD, PID, PI+S). In the most simple case it works as a standalone unit with the standard PID controller functionality with two degrees of freedom. It can operate in automatic mode (MAN = off) or manual mode (MAN = on).

In the automatic mode (MAN = off), the block PIDU implements the PID control law with two degrees of freedom in the form

\[
U(s) = \pm K \left\{ b W(s) - Y(s) + \frac{1}{T_i s} [W(s) - Y(s)] + \frac{T_d s}{N s + 1} [c W(s) - Y(s)] \right\} + Z(s)
\]

where \( U(s) \) is Laplace transform of the manipulated variable \( \text{mv} \), \( W(s) \) is Laplace transform of the setpoint variable \( \text{sp} \), \( Y(s) \) is Laplace transform of the process variable \( \text{pv} \), \( Z(s) \) is Laplace transform of the feedforward control variable \( \text{dv} \) and \( K, T_i, T_d, N, b \) and \( c \) are the parameters of the controller. The sign of the right hand side depends on the parameter \( \text{RACE} \). The range of the manipulated variable \( \text{mv} \) (position controller output) is limited by parameters \( \text{hilim}, \text{lolim} \). The parameter \( \text{dz} \) determines the dead zone in the integral part of the controller. The integral part of the control law can be switched off and fixed on the current value by the integrator hold input \( \text{IH} \) (IH = on). For the proper function of the controller it is necessary to connect the output \( \text{mv} \) of the controller to the controller input \( \text{tv} \) and properly set the tracking time constant \( \text{tt} \).

The rule of thumb for a PID controller is \( \text{tt} \approx \sqrt{\frac{T_i}{T_d}} \). For a PI controller the formula is \( \text{tt} \approx T_i/2 \). In this way we obtain the bumpless operation of the controller in the case of the mode switching (manual, automatic) and also the correct operation of the controller when saturation of the output \( \text{mv} \) occurs (antiwindup).

By adjusting the \( \text{tt} \) parameter, it is possible to tune the behaviour at saturation limits (so-called bouncing from limits due to noise) and when switching multiple controllers (bump in the controller output occurs when switching controllers while the control error is non-zero).

The additional outputs \( \text{dmv}, \text{de} \) and \( \text{SAT} \) generate the velocity output (difference of \( \text{mv} \)), deviation error and saturation flag, respectively.

If the PIDU block is connected with the SCUV block to configure the 3-point step controller without the positional feedback, then the parameter \( \text{icotype} \) must be set to...
4 and the meaning of the outputs mv and dmv and SAT is modified in the following way: 
mv and dmv give the PD part and difference of I part of the control law, respectively, and 
SAT provides the information for the SCUV block whether the deviation error is less than 
the dead zone dz in the automatic mode. In this case, the setpoint weighting factor c 
should be zero.

In the manual mode (MAN = on), the input hv is copied to the output mv unless 
saturated. The overall control function of the PIĐU block is quite clear from the following 
diagram:

**Inputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>dv</td>
<td>Feedforward control variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>sp</td>
<td>Setpoint variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>pv</td>
<td>Process variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>tv</td>
<td>Tracking variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>hv</td>
<td>Manual value</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>MAN</td>
<td>Manual or automatic mode</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... Automatic mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... Manual mode</td>
<td></td>
</tr>
<tr>
<td>IH</td>
<td>Integrator hold</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... Integration enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... Integration disabled</td>
<td></td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>mv</td>
<td>Manipulated variable (controller output)</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>dmv</td>
<td>Controller velocity output (difference)</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>de</td>
<td>Deviation error</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
SAT  Saturation flag  
off ... The controller implements a linear control law
on .... The controller output is saturated

Parameters

irtype  Controller type (control law)  \( \circ 6 \)  Long (I32)

1  ...  D  4  ...  P  7  ...  PID
2  ...  I  5  ...  PD
3  ...  ID  6  ...  PI

RACT  Reverse action flag  
off ... Higher mv \( \rightarrow \) higher pv
on .... Higher mv \( \rightarrow \) lower pv

k  Controller gain \( K \). By definition, the value 0 turns the controller off. Negative values are not allowed, use the RACT parameter for such a purpose.  \( \downarrow 0.0 \circ 1.0 \)  Double (F64)

ti  Integral time constant \( T_i \). The value 0 disables the integrating part (the same effect as disabling it by the irtype parameter).  \( \downarrow 0.0 \circ 4.0 \)  Double (F64)

td  Derivative time constant \( T_d \). The value 0 disables the derivative part (the same effect as disabling it by the irtype parameter).  \( \downarrow 0.0 \circ 1.0 \)  Double (F64)

nd  Derivative filtering parameter \( N \). The value 0 disables the derivative part (the same effect as disabling it by the irtype parameter).  \( \downarrow 0.0 \circ 10.0 \)  Double (F64)

b  Setpoint weighting - proportional part  \( \downarrow 0.0 \uparrow 2.0 \circ 1.0 \)  Double (F64)

c  Setpoint weighting - derivative part  \( \downarrow 0.0 \uparrow 2.0 \)  Double (F64)

tt  Tracking time constant. The value 0 stands for an implicit value, which is \( T_i/2 \) or \( \sqrt{T_i T_d} \) (see above) for controllers with integrating part. For controllers without integrating part, the value 0 disables tracking. If tracking is needed for a P or PD controller, it can be enabled by entering a positive value greater than the sampling time. It is not possible to turn off tracking for controllers with the integrating part (due to the windup effect).  \( \downarrow 0.0 \circ 1.0 \)  Double (F64)

hilim  Upper limit of the controller output  \( \circ 1.0 \)  Double (F64)

lolim  Lower limit of the controller output  \( \circ -1.0 \)  Double (F64)

dz  Dead zone  Double (F64)

icotype  Controller output type  \( \circ 1 \)  Long (I32)

1  ...  Analog output
2  ...  Pulse width modulation (PWM)
3  ...  Step controller unit with position feedback (SCU)
4  ...  Step controller unit without position feedback (SCUV)
PIDUI – PID controller unit with variable parameters

Function Description
The functionality of the PIDUI block is completely equivalent to the PIDU block. The only difference is that the PID control algorithm parameters are defined by the input signals and therefore they can depend on the outputs of other blocks. This allows creation of special adaptive PID controllers.

Inputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>dv</td>
<td>Feedforward control variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>sp</td>
<td>Setpoint variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>pv</td>
<td>Process variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>tv</td>
<td>Tracking variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>hv</td>
<td>Manual value</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>MAN</td>
<td>Manual or automatic mode</td>
<td>Boolean</td>
</tr>
<tr>
<td>IH</td>
<td>Integrator hold</td>
<td>Boolean</td>
</tr>
<tr>
<td>k</td>
<td>Controller gain $K$</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>ti</td>
<td>Integral time constant $T_i$</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>td</td>
<td>Derivative time constant $T_d$</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>nd</td>
<td>Derivative filtering parameter $N$</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>b</td>
<td>Setpoint weighting – proportional part</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>c</td>
<td>Setpoint weighting – derivative part</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>mv</td>
<td>Manipulated variable (controller output)</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>dmv</td>
<td>Controller velocity output (difference)</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>de</td>
<td>Deviation error</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
SAT  Saturation flag  Bool
    off ... The controller implements a linear control law
    on ... The controller output is saturated

Parameters

irtype Controller type (control law)  o6  Long (I32)
    1 ...... D  4 ...... P  7 ...... PID
    2 ...... I  5 ...... PD
    3 ...... ID  6 ...... PI

RACT Reverse action flag  Bool
    off ... Higher mv → higher pv
    on ... Higher mv → lower pv

tt Tracking time constant  o1.0  Double (F64)

hilim Upper limit of the controller output  o1.0  Double (F64)

lolim Lower limit of the controller output  o-1.0  Double (F64)

dz Dead zone  Double (F64)

icotype Controller output type  o1  Long (I32)
    1 ...... Analog output
    2 ...... Pulse width modulation (PWM)
    3 ...... Step controller unit with position feedback (SCU)
    4 ...... Step controller unit without position feedback (SCUV)
CHAPTER 7. REG – FUNCTION BLOCKS FOR CONTROL

**POUT – Pulse output**

**Block Symbol**

![POUT Symbol]

**Licence:** STANDARD

**Function Description**

The **POUT** block shapes the input pulses **U** in such a way, that the output pulse **Y** has a duration of at least **dtime** seconds and the idle period between two successive output pulses is at least **btime** seconds. The input pulse occurring sooner than the period of **btime** seconds since the last falling edge of the output signal elapses has no effect on the output signal **Y**.

**Input**

- **U**  Logical input of the block  **Bool**

**Output**

- **Y**  Logical output of the block  **Bool**

**Parameters**

- **dtime**  Minimum width of the output pulse [s]  
  - 1.0  Double (F64)
- **btime**  Minimum delay between two successive output pulses [s]  
  - 1.0  Double (F64)
PRGM – Setpoint programmer

Block Symbol

Licence: STANDARD

Function Description

The PRGM block generates functions of time (programs) composed of \( n \) linear parts defined by \( (n + 1) \)-dimensional vectors of time \( \tau = \{t_0, \ldots, t_n\} \) and output values \( y = \{y_0, \ldots, y_n\} \). The generated time-course is continuous piecewise linear, see figure below. This block is most commonly used as a setpoint generator for a controller. The program generation starts when RUN = on. In the case of RUN = off the programmer is set back to the initial state. The input DEF = on sets the output sp to the value spv. It follows a ramp to the nearest future node of the time function when DEF = off. The internal time of the generator is not affected by this input. The input HLD = on freezes the output sp and the internal time, thus also the outputs tsc, tt, and rt. The program follows from freezing point as planned when HLD = off unless the input CON = on at the moment when the signal HLD on → off. In that case the program follows a ramp to reach the node with index ind in time trt. The node index ind must be equal to or higher than the index of current sector isc (at the moment when HLD on → off). If RPT = on, the program is generated repeatedly.

![Diagram](image)

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>Enable execution</td>
<td>Bool</td>
</tr>
<tr>
<td>DEF</td>
<td>Initialize sp to the value of spv</td>
<td>Bool</td>
</tr>
<tr>
<td>spv</td>
<td>Initializing constant</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>HLD</td>
<td>Output and timer freezing</td>
<td>Bool</td>
</tr>
</tbody>
</table>
### CHAPTER 7. REG - FUNCTION BLOCKS FOR CONTROL

<table>
<thead>
<tr>
<th>CDN</th>
<th>Continue from defined node</th>
<th>Bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>ind</td>
<td>Index of the node to continue from</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>trt</td>
<td>Time to reach the defined node with index ind</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>RPT</td>
<td>Repetition flag</td>
<td>Bool</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>sp</th>
<th>Setpoint variable (function value of the time function at given time)</th>
<th>Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>isc</td>
<td>Current function sector</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>tsc</td>
<td>Time elapsed since the start of current sector</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>tt</td>
<td>Time elapsed since the start of program generation</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>rt</td>
<td>Remaining time till the end of program</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>CNF</td>
<td>Flag indicating that the configured curve is being followed</td>
<td>Bool</td>
</tr>
<tr>
<td>E</td>
<td>Error flag – the node times are not ascending</td>
<td>Bool</td>
</tr>
</tbody>
</table>

**Parameters**

| n    | Number of sectors | 1↑100000000 ≤2 | Long (I32) |
| tunits | Time units        | ≤1 | Long (I32) |
|       | 1 ...... seconds   |   |   |
|       | 2 ...... minutes   |   |   |
|       | 3 ...... hours     |   |   |
| tm   | (n + 1)-dimensional vector of ascending node times | ⊗[0 1 2] | Double (F64) |
| y    | (n + 1)-dimensional vector of node values (values of the time function) | ⊗[0 1 0] | Double (F64) |
PSMPC – Pulse-step model predictive controller

Block Symbol

Licence: ADVANCED

Function Description

The PSMPC block can be used for control of hardly controllable linear time-invariant systems with manipulated value constraints (e.g. time delay or non-minimum phase systems). It is especially well suited for the case when fast transition without overshoot from one level of controlled variable to another is required. In general, the PSMPC block can be used where the PID controllers are commonly used.

The PSMPC block is a predictive controller with explicitly defined constraints on the amplitude of manipulated variable.

The prediction is based on the discrete step response \( g(j), j = 1, \ldots, N \) is used. The figure above shows how to obtain the discrete step response \( g(j), j = 0, 1, \ldots, N \) and the discrete impulse response \( h(j), j = 0, 1, \ldots, N \) with sampling period \( T_S \) from continuous step response. Note that \( N \) must be chosen such that \( N \cdot T_S > t_{95} \), where \( t_{95} \) is the time to reach 95% of the final steady state value.

For stable, linear and t-invariant systems with monotonous step response it is also possible to use the moment model set approach [4] and describe the system by only 3 characteristic numbers \( \kappa, \mu, \) and \( \sigma^2 \), which can be obtained easily from a very short
and simple experiment. The controlled system can be approximated by first order plus
dead-time system

$$F_{FOPDT}(s) = \frac{K}{\tau s + 1} \cdot e^{-Ds}, \quad \kappa = K, \quad \mu = \tau + D, \quad \sigma^2 = \tau^2$$  \hspace{1cm} (7.1)

or second order plus dead-time system

$$F_{SOPDT}(s) = \frac{K}{(\tau s + 1)^2} \cdot e^{-Ds}, \quad \kappa = K, \quad \mu = 2\tau + D, \quad \sigma^2 = 2\tau^2$$  \hspace{1cm} (7.2)

with the same characteristic numbers. The type of approximation is selected by the $\text{imtype}$ parameter.

To lower the computational burden of the open-loop optimization, the family of
admissible control sequences contains only sequences in the so-called pulse-step shape
depicted below:

![Pulse-Step Shape Diagram]

Note that each of these sequences is uniquely defined by only four numbers $n_1, n_2 \in \{0, \ldots, N_C\}$, $p_0$ and $u^\infty \in (u^-, u^+)$, where $N_C \in \{0, 1, \ldots\}$ is the control horizon and $u^-, u^+$ stand for the given lower and upper limit of the manipulated variable. The on-line
optimization (with respect to $p_0$, $n_1$, $n_2$ and $u^\infty$) minimizes the criterion

$$I = \sum_{i=N_1}^{N_2} \hat{e}(k+i|k)^2 + \lambda \sum_{i=0}^{N_C} \Delta \hat{u}(k+i|k)^2 \rightarrow \min,$$  \hspace{1cm} (7.3)

where $\hat{e}(k+i|k)$ is the predicted control error at time $k$ over the coincidence interval
$i \in \{N_1, N_2\}$, $\Delta \hat{u}(k+i|k)$ are the differences of the control signal over the interval $i \in \{0, N_C\}$ and $\lambda$ penalizes the changes in the control signal. The algorithm used for solving
the optimization task (7.3) combines brute force and the least squares method. The value $u^\infty$ is determined using the least squares method for all admissible combinations of $p_0$, $n_1$ and $n_2$ and the optimal control sequence is selected afterwards. The selected sequence in the pulse-step shape is optimal in the open-loop sense. To convert from open-loop to
closed-loop control strategy, only the first element of the computed control sequence is
applied and the whole optimization procedure is repeated in the next sampling instant.

The parameters $N_1$, $N_2$, $H_C$, and $\lambda$ in the criterion (7.3) take the role of design
parameters. Only the last parameter $\lambda$ is meant for manual tuning of the controller.
In the case the model in the form (7.1) or (7.2) is used, the parameters $N_1$ and $N_2$
are determined automatically with respect to the $\mu$ and $\sigma^2$ characteristic numbers. The
controller can be then effectively tuned by adjusting the characteristic numbers \( \kappa, \mu \) and \( \sigma^2 \).

**Warning**

It is necessary to set the \( n_{sr} \) parameter to sufficiently large number to avoid Matlab/Simulink crash when using the FIS MPC block for simulation purposes. Especially when using FOPDT or SOPDT model, the \( n_{sr} \) parameter must be greater than the length of the internally computed discrete step response.

**Inputs**

- **sp** Setpoint variable  
  Double (F64)
- **pv** Process variable  
  Double (F64)
- **tv** Tracking variable (applied control signal)  
  Double (F64)
- **hv** Manual value  
  Double (F64)
- **MAN** Manual or automatic mode  
  - **off** ... Automatic mode  
  - **on** ... Manual mode  
  Bool

**Outputs**

- **mv** Manipulated variable (controller output)  
  Double (F64)
- **dsv** Controller velocity output (difference)  
  Double (F64)
- **de** Deviation error  
  Double (F64)
- **SAT** Saturation flag  
  - **off** ... The controller implements a linear control law  
  - **on** ... The controller output is saturated  
  Bool
- **pve** Predicted process variable based on the controlled process model  
  Double (F64)
- **iE** Error code  
  - **0** ..... No error  
  - **1** ..... Incorrect FOPDT model  
  - **2** ..... Incorrect SOPDT model  
  - **3** ..... Invalid step response sequence  
  Long (I32)

**Parameters**

- **nc** Control horizon length \( (N_C) \)  
  \( \odot 5 \)  
  Long (I32)
- **np1** Start of coincidence interval \( (N_1) \)  
  \( \odot 1 \)  
  Long (I32)
- **np2** End of coincidence interval \( (N_2) \)  
  \( \odot 10 \)  
  Long (I32)
- **lambda** Control signal penalization coefficient \( (\lambda) \)  
  \( \odot 0.05 \)  
  Double (F64)
- **umax** Upper limit of the controller output \( (u^+) \)  
  \( \odot 1.0 \)  
  Double (F64)
- **umin** Lower limit of the controller output \( (u^-) \)  
  \( \odot -1.0 \)  
  Double (F64)
<table>
<thead>
<tr>
<th>imtype</th>
<th>Controlled process model type</th>
<th>0:3</th>
<th>Long (I32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FOPDT model (7.1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SOPDT model (7.2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Discrete step response</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| kappa    | Static gain ($\kappa$)        | 0:1.0 | Double (F64) |
| mu       | Resident time constant ($\mu$) | 0:20.0 | Double (F64) |
| sigma    | Measure of the system response length ($\sqrt{\sigma^2}$) | 0:10.0 | Double (F64) |
| nvr      | Length of the discrete step response ($N$), see the warning above | | Long (I32) |
| ssa      | Discrete step response sequence ($[g(1), \ldots, g(N)]$) | | Double (F64) |
|          |                               | 0:0.2642 0.5940 0.8009 0.9084 0.9596 0.9826 0.9927 0.9970 0.9988 0.9995 |
**PWM – Pulse width modulation**

**Block Symbol**

![PWM Block Symbol]

**Licence:** STANDARD

**Function Description**

The **PWM** block implements a pulse width modulation algorithm for proportional actuators. In the general, it is assumed the input signal \( u \) ranges in the interval from -1 to +1. The width \( L \) of the output pulse is computed by the expression:

\[
L = \text{pertm} \cdot |u|,
\]

where \( \text{pertm} \) is the modulation time period. If \( u > 0 \) (\( u < 0 \)), the pulse is generated in the output \( \text{UP} \) (\( \text{DN} \)). However, the width of the generated pulses are affected by other parameters of the block. The asymmetry factor \( \text{asyfac} \) determines the ratio of negative pulses duration to positive pulses duration. The modified pulse widths are given by:

\[
\text{if } u > 0 \text{ then } L(\text{UP}) := \begin{cases} 
L & \text{for } \text{asyfac} \leq 1.0 \\
L/\text{asyfac} & \text{for } \text{asyfac} > 1.0
\end{cases}
\]

\[
\text{if } u < 0 \text{ then } L(\text{DN}) := \begin{cases} 
L \cdot \text{asyfac} & \text{for } \text{asyfac} \leq 1.0 \\
L & \text{for } \text{asyfac} > 1.0
\end{cases}
\]

Further, if the computed width is less than minimum pulse duration \( \text{dtime} \) the width is set to zero. If the pulse width differs from the modulation period \( \text{pertm} \) less than minimum pulse break time \( \text{btime} \) then width of the pulse is set to \( \text{pertm} \). In the case the positive pulse is succeeded by the negative one (or vice versa) the latter pulse is possibly shifted in such a way that the distance between these pulses is at least equal to the minimum off time \( \text{offtime} \). If SYNCH = on, then the change of the input value \( u \) causes the immediate recalculation of the current pulse widths if a synchronization condition is violated.

**Input**

\( u \) Analog input of the block

**Double (F64)**

**Outputs**

| \( \text{UP} \) | The "up" signal | Bool |
| \( \text{DN} \) | The "down" signal | Bool |
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{pertz} )</td>
<td>Modulation period length [s]</td>
<td>( \odot 0.0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>( \text{dtime} )</td>
<td>Minimum width of the output pulse [s]</td>
<td>( \odot 0.1 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>( \text{btime} )</td>
<td>Minimum delay between output pulses [s]</td>
<td>( \odot 0.1 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>( \text{offtime} )</td>
<td>Minimum delay when altering direction [s]</td>
<td>( \odot 1.0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>( \text{asymfac} )</td>
<td>Asymmetry factor</td>
<td>( \odot 1.0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>\text{SYNCH}</td>
<td>Synchronization flag of the period start</td>
<td></td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>( \text{off} \ldots ) Synchronization disabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \text{on} \ldots ) Synchronization enabled</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RLY – Relay with hysteresis

Block Symbol

Function Description
The RLY block transforms the input signal $u$ to the output signal $y$ according to the figure below.

Input
$u$ Analog input of the block Double (F64)

Output
$y$ Analog output of the block Double (F64)

Parameters
$ep$ The value $u > ep$ causes $y = ap$ ("On") $\Theta 1.0$ Double (F64)
$en$ The value $u < en$ causes $y = an$ ("Off") $\Theta -1.0$ Double (F64)
ap Output value $y$ in the "On" state $\Theta 1.0$ Double (F64)
an Output value $y$ in the "Off" state $\Theta -1.0$ Double (F64)
y0 Initial output value at start-up Double (F64)
SAT – Saturation with variable limits

Block Symbol

Function Description

The SAT block copies the input u to the output y if the input signal satisfies lolim ≤ u and u ≤ hilim, where lolim and hilim are state variables of the block. If u < lolim (u > hilim), then y = lolim (y = hilim). The upper and lower limits are either constants (HLD = on) defined by parameters hilim0 and lolim0 respectively or input-driven variables (HLD = off, hi and lo inputs). The maximal rate at which the active limits may vary is given by time constants tp (positive slope) and tn (negative slope). These rates are active even if the saturation limits are changed manually (HLD = on) using the hilim0 and lolim0 parameters. To allow immediate changes of the saturation limits, set tp = 0 and tn = 0. The HL and LL outputs indicate the upper and lower saturation respectively.

If necessary, the hilim0 and lolim0 parameters are used as initial values for the input-driven saturation limits.

Inputs

<table>
<thead>
<tr>
<th>u</th>
<th>Analog input of the block</th>
<th>Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>hi</td>
<td>Upper limit of the output signal (for the case HLD = off)</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>lo</td>
<td>Lower limit of the output signal (for the case HLD = off)</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>y</th>
<th>Analog output of the block</th>
<th>Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HL</td>
<td>Upper limit saturation indicator</td>
<td>Bool</td>
</tr>
<tr>
<td>LL</td>
<td>Lower limit saturation indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>tp</th>
<th>Time constant defining the maximal positive slope of active limit changes</th>
<th>Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tn</td>
<td>Time constant defining the maximum negative slope of active limit changes</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>hilim0</td>
<td>Upper limit of the output (valid for HLD = on)</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>lolim0</td>
<td>Lower limit of the output (valid for HLD = on)</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
HLD  Fixed saturation limits  o on  Bool
     off  . . Variable limits   on  . . Fixed limits
SC2FA – State controller for 2nd order system with frequency autotuner

Block Symbol

Licence: AUTOTUNING

Function Description

The SC2FA block implements a state controller for 2nd order system (7.4) with frequency autotuner. It is well suited especially for control (active damping) of lightly damped systems ($\xi < 0.1$). But it can be used as an autotuning controller for arbitrary system which can be described with sufficient precision by the transfer function

$$F(s) = \frac{b_1s + b_0}{s^2 + 2\xi\Omega s + \Omega^2},$$

(7.4)

where $\Omega > 0$ is the natural (undamped) frequency, $\xi$, $0 < \xi < 1$, is the damping coefficient and $b_1, b_0$ are arbitrary real numbers. The block has two operating modes: "Identification and design mode" and "Controller mode".

The "Identification and design mode" is activated by the binary input $\text{ID} = \text{on}$. Two points of frequency response with given phase delay are measured during the identification experiment. Based on these two points a model of the controlled system is built. The experiment itself is initiated by the rising edge of the $\text{RUN}$ input. A harmonic signal with amplitude $\text{amp}$, frequency $\omega$ and bias $\text{bias}$ then appears at the output $\text{mv}$. The frequency runs through the interval $(\text{wb}, \text{wf})$, it increases gradually. The current frequency is copied to the output $w$. The rate at which the frequency changes (sweeping) is determined by the $\text{cp}$ parameter, which defines the relative shrinking of the initial period $T_b = \frac{2\pi}{\text{wb}}$ of the exciting sine wave in time $T_b$, thus

$$\text{cp} = \frac{\text{wb}}{\omega(T_b)} = \frac{\text{wb}}{\text{wb}e^{-\gamma T_b}} = e^{-\gamma T_b}.$$

The $\text{cp}$ parameter usually lies within the interval $\text{cp} \in (0.95; 1)$. The lower the damping coefficient $\xi$ of the controlled system is, the closer to one the $\text{cp}$ parameter must be.
At the beginning of the identification period the exciting signal has a frequency of \( \omega = \omega_0 \). After a period of \( \text{stime} \) seconds the estimation of current frequency response point starts. Its real and imaginary parts are available at the \( xre \) and \( xim \) outputs. If the \( \text{MANF} \) parameter is set to 0, then the frequency sweeping is stopped two times during the identification period. This happens when points with phase delay of \( \phi_1 \) and \( \phi_2 \) are reached for the first time. The breaks are \( \text{stime} \) seconds long. Default phase delay values are \(-60^\circ\) and \(-120^\circ\), respectively, but these can be changed to arbitrary values within the interval \((-360^\circ, 0^\circ)\), where \( \phi_1 > \phi_2 \). At the end of each break an arithmetic average is computed from the last \( \text{iavg} \) frequency point estimates. Thus we get two points of frequency response which are successively used to compute the controlled process model in the form of (7.4). If the \( \text{MANF} \) parameter is set to 1, then the selection of two frequency responses points is manual. To select the frequency, set the input \( \text{HLD} = \text{on} \), which stops the frequency sweeping. The identification experiment continues after returning the input \( \text{HLD} \) to 0. The remaining functionality is unchanged.

It is possible to terminate the identification experiment prematurely in case of necessity by the input \( \text{BRK} = \text{on} \). If the two points of frequency response are already identified at that moment, the controller parameters are designed in a standard way. Otherwise the controller design cannot be performed and the identification error is indicated by the output signal \( \text{IDE} = \text{on} \).

The \( \text{IDBSY} \) output is set to 1 during the "identification and design" phase. It is set back to 0 after the identification experiment finishes. A successful controller design is indicated by the output \( \text{IDE} = \text{off} \). During the identification experiment the output \( \text{iIDE} \) displays the individual phases of the identification: \( \text{iIDE} = -1 \) means approaching the first point, \( \text{iIDE} = 1 \) means the break at the first point, \( \text{iIDE} = -2 \) means approaching the second point, \( \text{iIDE} = 2 \) means the break at the second point and \( \text{iIDE} = -3 \) means the last phase after leaving the second frequency response point. An error during the identification phase is indicated by the output \( \text{IDE} = \text{on} \) and the output \( \text{iIDE} \) provides more information about the error.

The computed state controller parameters are taken over by the control algorithm as soon as the \( \text{SETC} \) input is set to 1 (i.e. immediately if \( \text{SETC} \) is constantly set to \( \text{on} \)). The identified model and controller parameters can be obtained from the \( \text{p1} \), \( \text{p2} \), \ldots, \( \text{p6} \) outputs after setting the \( \text{ips} \) input to the appropriate value. After a successful identification it is possible to generate the frequency response of the controlled system model, which is initiated by a rising edge at the \( \text{MFR} \) input. The frequency response can be read from the \( \text{w} \), \( xre \) and \( xim \) outputs, which allows easy confrontation of the model and the measured data.

The "Controller mode" (binary input \( \text{ID} = \text{off} \)) has manual (\( \text{MAN} = \text{on} \)) and automatic (\( \text{MAN} = \text{off} \)) submodes. After a cold start of the block with the input \( \text{ID} = \text{off} \) it is assumed that the block parameters \( mb0, mb1, ma0 \) and \( ma1 \) reflect formerly identified coefficients \( b_0, b_1, a_0 \) and \( a_1 \) of the controlled system transfer function and the state controller design is performed automatically. Moreover if the controller is in the automatic mode and \( \text{SETC} = \text{on} \), then the control law uses the parameters from the very beginning. If in this way the identification phase can be skipped when starting the block repeatedly.
The diagram above is a simplified inner structure of the frequency autotuning part of the controller. The diagram below shows the state feedback, observer and integrator anti-wind-up. The diagram does not show the fact, that the controller design block automatically adjusts the observer and state feedback parameters \( f_1, \ldots, f_5 \) after identification experiment (and SETC = on).
The controlled system is assumed in the form of (7.4). Another forms of this transfer function are

\[ F(s) = \frac{(b_1 s + b_0)}{s^2 + a_1 s + a_0} \]  

and

\[ F(s) = \frac{K_0 \Omega^2(\tau s + 1)}{s^2 + 2\xi \Omega s + \Omega^2}. \]

The coefficients of these transfer functions can be found at the outputs p1,...,p6 after the identification experiment \( \texttt{IDBSY = off} \). The output signals meaning is switched when a change occurs at the \( \texttt{ip5} \) input.

**Inputs**

- \( \text{dv} \)  
  Feedforward control variable  
  Double (F64)

- \( \text{sp} \)  
  Setpoint variable  
  Double (F64)

- \( \text{pv} \)  
  Process variable  
  Double (F64)

- \( \text{tv} \)  
  Tracking variable  
  Double (F64)

- \( \text{hv} \)  
  Manual value  
  Double (F64)
### CHAPTER 7. REG - FUNCTION BLOCKS FOR CONTROL

#### MAN
Manual or automatic mode
- **off** . . . Automatic mode
- **on** . . . Manual mode

#### ID
Identification or controller operating mode
- **off** . . . Controller mode
- **on** . . . Identification and design

#### TUNE
Start the tuning experiment (off→on), the exciting harmonic signal is generated

#### HLD
Stop frequency sweeping

#### BRK
Termination signal

#### SETC
Flag for accepting the new controller parameters and updating the control law
- **off** . . . Parameters are only computed
- **on** . . . Parameters are accepted as soon as computed
- **off→on** . One-shot confirmation of the computed parameters

#### ips
Switch for changing the meaning of the output signals

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Two points of frequency response</td>
</tr>
<tr>
<td>1</td>
<td>Second order model in the form (7.5)</td>
</tr>
<tr>
<td>2</td>
<td>Second order model in the form (7.6)</td>
</tr>
<tr>
<td>3</td>
<td>State feedback parameters</td>
</tr>
</tbody>
</table>

#### MFR
Generation of the parametric model frequency response at the **xre** and **xim** outputs (off→on triggers the generator)

#### Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mv</strong></td>
<td>Manipulated variable (controller output)</td>
<td>Double (F64)</td>
</tr>
<tr>
<td><strong>de</strong></td>
<td>Deviation error</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>SAT</td>
<td>Saturation flag</td>
<td>Bool</td>
</tr>
<tr>
<td>-------</td>
<td>---------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>off</td>
<td>The controller implements a linear control law</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>The controller output is saturated</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IDBSY</th>
<th>Identification running</th>
<th>Bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Identification not running</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>Identification in progress</td>
<td></td>
</tr>
</tbody>
</table>

| w     | Frequency response point estimate - frequency in rad/s                         | Double (F64) |
| xre   | Frequency response point estimate - real part                                   | Double (F64) |
| xim   | Frequency response point estimate - imaginary part                              | Double (F64) |
| epv   | Reconstructed pv signal                                                        | Double (F64) |

<table>
<thead>
<tr>
<th>IDE</th>
<th>Identification error indicator</th>
<th>Bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Successful identification experiment</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>Identification error occurred</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iIDE</th>
<th>Error code</th>
<th>Long (I32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Sampling period too low</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>Error identifying one or both frequency response point(s)</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>Manipulated variable saturation occurred during the identification experiment</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>Invalid process model</td>
<td></td>
</tr>
</tbody>
</table>

| pi..p6| Results of identification and design phase                                     | Double (F64) |

**Parameters**

<table>
<thead>
<tr>
<th>ubias</th>
<th>Static component of the exciting harmonic signal</th>
<th>Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>uamp</td>
<td>Amplitude of the exciting harmonic signal</td>
<td></td>
</tr>
<tr>
<td>wb</td>
<td>Frequency interval lower limit [rad/s]</td>
<td></td>
</tr>
<tr>
<td>wf</td>
<td>Frequency interval upper limit [rad/s]</td>
<td></td>
</tr>
<tr>
<td>isweep</td>
<td>Frequency sweeping mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 ..... Logarithmic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ..... Linear (not implemented yet)</td>
<td></td>
</tr>
<tr>
<td>cp</td>
<td>Sweeping rate</td>
<td></td>
</tr>
<tr>
<td>iavg</td>
<td>Number of values for averaging</td>
<td></td>
</tr>
<tr>
<td>alpha</td>
<td>Relative positioning of the observer poles (in identification phase)</td>
<td></td>
</tr>
<tr>
<td>xi</td>
<td>Observer damping coefficient (in identification phase)</td>
<td></td>
</tr>
<tr>
<td>MANF</td>
<td>Manual frequency response points selection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>off ... Disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... Enabled</td>
<td></td>
</tr>
<tr>
<td>ph1</td>
<td>Phase delay of the 1st point in degrees</td>
<td></td>
</tr>
<tr>
<td>ph2</td>
<td>Phase delay of the 2nd point in degrees</td>
<td></td>
</tr>
<tr>
<td>stime</td>
<td>Settling period [s]</td>
<td></td>
</tr>
<tr>
<td>ralpha</td>
<td>Relative positioning of the observer poles</td>
<td></td>
</tr>
<tr>
<td>rxi</td>
<td>Observer damping coefficient</td>
<td></td>
</tr>
<tr>
<td>acl1</td>
<td>Relative positioning of the 1st closed-loop poles couple</td>
<td></td>
</tr>
<tr>
<td>xic11</td>
<td>Damping of the 1st closed-loop poles couple</td>
<td></td>
</tr>
<tr>
<td>INTGF</td>
<td>Integrator flag</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>off . . . State-space model without integrator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on . . . Integrator included in the state-space model</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>apcl</th>
<th>Relative position of the real pole</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISF</th>
<th>Disturbance flag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>off . . . State-space model without disturbance model</td>
</tr>
<tr>
<td></td>
<td>on . . . Disturbance model is included in the state-space model</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dom</th>
<th>Disturbance model natural frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dxi</th>
<th>Disturbance model damping coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>acl2</th>
<th>Relative positioning of the 2nd closed-loop poles couple</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xicl2</th>
<th>Damping of the 2nd closed-loop poles couple</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.707</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tt</th>
<th>Tracking time constant</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>hilim</th>
<th>Upper limit of the controller output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lolim</th>
<th>Lower limit of the controller output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-0.1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mb1p</th>
<th>Controlled system transfer function coefficient b1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mb0p</th>
<th>Controlled system transfer function coefficient b0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ma1p</th>
<th>Controlled system transfer function coefficient a1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.2.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ma0p</th>
<th>Controlled system transfer function coefficient a0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1.0</td>
</tr>
</tbody>
</table>
**SCU – Step controller with position feedback**

**Block Symbol**

![](image)

**Function Description**

The SCU block implements the secondary (inner) position controller of the step controller loop. PIDU function block or some of the derived function blocks (PIDMA, etc.) is assumed as the primary controller.

The SCU block processes the control deviation \( sp - pv \) by a three state element with parameters (thresholds) thron and throff (see the TSE block, use parameters \( ep = \text{thron} \), \( epoff = \text{throff} \), \( en = -\text{thron} \) and \( enoff = -\text{throff} \)). The parameter RACT determines whether the UP or DN pulse is generated for positive or negative value of the controller deviation. Two pulse outputs of the three state element are further shaped so that minimum pulse duration \( \text{dtime} \) and minimum pulse break time \( \text{btime} \) are guaranteed at the block UP and DN outputs. If signals from high and low limit switches of the valve are available, they should be connected to the HS and LS inputs.

There is also a group of input signals for manual control available. The manual mode is activated by the MAN = on input signal. Then it is possible to move the motor back and forth by the MUP and MDN input signals. It is also possible to specify a position increment/decrement request by the mdv input. In this case the request must be confirmed by a rising edge (off→on) in the DVC input signal.

The control function of the SCU block is quite clear from the following diagram.

![Diagram](image)

The complete structure of the three-state step controller is depicted in the following
**Inputs**

- `sp`: Setpoint (output of the primary controller)  
  Double (F64)
- `pv`: Controlled variable (position of the motorized valve drive)  
  Double (F64)
- `HS`: Upper end switch (detects the upper limit position of the valve)  
  Bool
- `LS`: Lower end switch (detects the lower limit position of the valve)  
  Bool
- `MUP`: Manual UP signal  
  Bool
- `MDN`: Manual DN signal  
  Bool
- `mdv`: Manual differential value (requested position increment/decrement with higher priority than direct signals MUP/MDN)  
  Double (F64)
- `DVC`: Differential value change command (off→on)  
  Bool
- `MAN`: Manual or automatic mode  
  Bool
  - `off`: Automatic mode  
  - `on`: Manual mode

**Outputs**

- `UP`: The "up" signal  
  Bool
- `DN`: The "down" signal  
  Bool
- `de`: Deviation error  
  Double (F64)

**Parameters**

- `thron`: Switch-on value  
  \[0.0 \circ 0.02\]  
  Double (F64)
- `thoff`: Switch-off value  
  \[0.0 \circ 0.01\]  
  Double (F64)
- `dtim`: Minimum width of the output pulse [s]  
  \[0.0 \circ 0.1\]  
  Double (F64)
- `btim`: Minimum delay between two subsequent output pulses [s]  
  \[0.0 \circ 0.1\]  
  Double (F64)
<table>
<thead>
<tr>
<th>RACT</th>
<th>Reverse action flag</th>
<th>Bool</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>off ... Higher mv → higher pv</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ..... Higher mv → lower pv</td>
<td></td>
</tr>
<tr>
<td>trun</td>
<td>Motor time constant (determines the time during which the motor position changes by one unit)</td>
<td>Double (F64)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0 10.0</td>
</tr>
</tbody>
</table>
SCUV – Step controller unit with velocity input

Block Symbol

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sv</td>
<td>Sat velocity</td>
</tr>
<tr>
<td>dmv</td>
<td>Deviation motor velocity</td>
</tr>
<tr>
<td>SAT</td>
<td>Sat position</td>
</tr>
<tr>
<td>HS</td>
<td>Happy signal</td>
</tr>
<tr>
<td>LS</td>
<td>Lousy signal</td>
</tr>
<tr>
<td>MUP</td>
<td>Move up</td>
</tr>
<tr>
<td>MDN</td>
<td>Move down</td>
</tr>
<tr>
<td>pos</td>
<td>Position</td>
</tr>
<tr>
<td>MR</td>
<td>Manual reset</td>
</tr>
<tr>
<td>DVC</td>
<td>DVC input</td>
</tr>
<tr>
<td>MAN</td>
<td>Manual</td>
</tr>
<tr>
<td>UP</td>
<td>Up</td>
</tr>
<tr>
<td>DN</td>
<td>Down</td>
</tr>
<tr>
<td>RACT</td>
<td>Reaction time</td>
</tr>
</tbody>
</table>

Function Description

The block SCUV substitutes the secondary position controller SCU in the step controller loop when the position signal is not available. The primary controller PIDU (or some of the derived function blocks) is connected with the block SCUV using the block inputs sv, dmv and SAT.

If the primary controller uses P or PD control law (CWOI = off), then all three inputs sv, dmv and SAT of the block SCUV are sequentially processed by the special integration algorithm and by the three state element with parameters thron and throff (see the TSE block, use parameters ep = thron, epoff = throff, en = -thron and enoff = -throff). Pulse outputs of the three state element are further shaped in such a way that the minimum pulse duration time dtime and minimum pulse break time btime are guaranteed at the block outputs UP and DN. The parameter RACT determines the direction of motor moving. Note, the velocity output of the primary controller is reconstructed from input signals sv and dmv. Moreover, if the deviation error of the primary controller with icotype = 4 (working in automatic mode) is less than its dead zone (SAT = on), then the output of the corresponding internal integrator is set to zero.

The position pos of the valve is estimated by an integrator with the time constant trun. If signals from high and low limit switches of the valve are available, they should be connected to the inputs HS and LS.

If the primary controller uses PI or PID control law (CWOI = on), then the deviation error of the primary controller can be eliminated by the bias ub manually. In this case, the control algorithm is slightly modified, the position of the motor pos is used and the proper settings of thron, throff and the tracking time constant tt are necessary for the suppressing of up/down pulses in the steady state.

There is also a group of input signals for manual control available. The manual mode is activated by the MAN = on input signal. Then it is possible to move the motor back and forth by the MUP and MDN input signals. It is also possible to specify a position increment/decrement request by the dmv input. In this case the request must be confirmed by a rising edge (off→on) in the DVC input signal.
The overall control function of the SCUV block is obvious from the following diagram:

The complete structures of the three-state controllers are depicted in the following figures:

**Primary controller with integration: I, PI, PID**

**Primary controller without integration: P, PD**

**Inputs**

- \( \text{mv} \) Manipulated variable (controller output)  
  - Double (F64)
- \( \text{dsv} \) Controller velocity output (difference)  
  - Double (F64)
### CHAPTER 7. REG - FUNCTION BLOCKS FOR CONTROL

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ub</td>
<td>Bias (only for P or PD primary controller)</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>SAT</td>
<td>Internal integrator reset (connected to the SAT output of the primary controller)</td>
<td>Bool</td>
</tr>
<tr>
<td>HS</td>
<td>Upper end switch (detects the upper limit position of the valve)</td>
<td>Bool</td>
</tr>
<tr>
<td>LS</td>
<td>Lower end switch (detects the lower limit position of the valve)</td>
<td>Bool</td>
</tr>
<tr>
<td>MUP</td>
<td>Manual UP signal</td>
<td>Bool</td>
</tr>
<tr>
<td>MDN</td>
<td>Manual DN signal</td>
<td>Bool</td>
</tr>
<tr>
<td>mdv</td>
<td>Manual differential value (requested position increment/decrement with higher priority than direct signals MUP/MDN)</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>DWC</td>
<td>Differential value change command (off→on)</td>
<td>Bool</td>
</tr>
<tr>
<td>MAN</td>
<td>Manual or automatic mode</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... Automatic mode on .... Manual mode</td>
<td></td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>UP</td>
<td>The &quot;up&quot; signal</td>
<td>Bool</td>
</tr>
<tr>
<td>DN</td>
<td>The &quot;down&quot; signal</td>
<td>Bool</td>
</tr>
<tr>
<td>pos</td>
<td>Position output of motor simulator</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>MR</td>
<td>Request to move the motor</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... Motor idle (UP = off and DN = off)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... Request to move (UP = on or DN = on)</td>
<td></td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>thron</td>
<td>Switch-on value</td>
<td>0.0000 to 0.02</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>throff</td>
<td>Switch-off value</td>
<td>0.0001 to 0.01</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>dtine</td>
<td>Minimum width of the output pulse [s]</td>
<td>0.00001 to 0.1</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>btine</td>
<td>Minimum delay between two subsequent output pulses [s]</td>
<td>0.0001 to 0.1</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>RACT</td>
<td>Reverse action flag</td>
<td></td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... Higher mv → higher pv</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... Higher mv → lower pv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trun</td>
<td>Motor time constant (determines the time during which the motor position changes by one unit)</td>
<td>0.00000 to 0.1000</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>CNOI</td>
<td>Controller without integration flag</td>
<td></td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... The primary controller has an integrator (I, PI, PID)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... The primary controller does not have an integrator (P, PD)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tt</td>
<td>Tracking time constant</td>
<td>0.0001 to 0.1</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
SELU – Controller selector unit

Block Symbol

Licence: STANDARD

Function Description

The SELU block is tailored for selecting the active controller in selector control. It chooses one of the input signals u₁, u₂, u₃, u₄ and copies it to the output y. For BINF = off the active signal is selected by the iSW input. In the case of BINF = on the selection is based on the binary inputs SW₁ and SW₂ according to the following table:

<table>
<thead>
<tr>
<th>iSW</th>
<th>SW₁</th>
<th>SW₂</th>
<th>y</th>
<th>U₁</th>
<th>U₂</th>
<th>U₃</th>
<th>U₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>off</td>
<td>off</td>
<td>u₁</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>1</td>
<td>off</td>
<td>on</td>
<td>u₂</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>2</td>
<td>on</td>
<td>off</td>
<td>u₃</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>3</td>
<td>on</td>
<td>on</td>
<td>u₄</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>off</td>
</tr>
</tbody>
</table>

This table also explains the meaning of the binary outputs U₁, U₂, U₃ and U₄, which are used by the inactive controllers in selector control for tracking purposes (via the SWU blocks).

Inputs

- u₁..u₄: Signals to be selected from
- iSW: Active signal selector in case of BINF = off
- SW₁: Binary signal selector, used when BINF = on
- SW₂: Binary signal selector, used when BINF = on

Double (F64), Long (I32), Bool

Outputs

- y: Analog output of the block
- U₁..U₄: Binary output signal for selector control

Double (F64), Bool

Parameter

- BINF: Enable the binary selectors

  - off: Disabled (analog selector)
  - on: Enabled (binary selectors)
SMHCC – Sliding mode heating/cooling controller

Block Symbol

Licence: ADVANCED

Function Description

The sliding mode heating/cooling controller SMHCC is a novel high quality control algorithm intended for temperature control of heating-cooling (possibly asymmetrical) processes with ON-OFF heaters and/or ON-OFF coolers. The plastic extruder is a typical example of such process. However, it can also be applied to many similar cases, for example in thermal systems where a conventional thermostat is employed. To provide the proper control function the block SMHCC must be combined with the block PWM (Pulse Width Modulation) as depicted in the following figure.

It is important to note that the block SMHCC works with two time periods. The first period \( T_S \) is the sampling time of the process temperature, and this period is equal to the period with which the block SMHCC itself is executed. The second period \( T_C = i_{pwmc} T_S \) is the control period with which the block SMHCC generates manipulated variable. This period \( T_C \) is also equal to the cycle time of PWM block. At every instant when the manipulated variable \( m_v \) is changed by SMHCC the PWM algorithm recalculates the width of the output pulse and starts a new PWM cycle. The time resolution \( T_R \) of the PWM block is third time period involved with. This period is equal to the period with which the block PWM is run and generally may be different from \( T_S \). To achieve the high quality of control it is recommended to choose \( T_S \) as minimal as possible (\( i_{pwmc} \) as maximal as possible), the ratio \( T_C/T_S \) as maximal as possible but \( T_C \) should be sufficiently small with respect to the process dynamics. An example of reasonable values for an extruder temperature control is as follows:

\[
T_S = 0.1, \quad i_{pwmc} = 100, \quad T_C = 10s, \quad T_R = 0.01s.
\]
The control law of the block SMHCC in automatic mode (MAN = off) is based on the discrete dynamic sliding mode control technique and special 3rd order filters for estimation of the first and second derivatives of the control error.

The first control stage, after a setpoint change or upset, is the reaching phase when the dynamic sliding variable

\[ s_k = \triangle \hat{e}_k + 2\xi \hat{e}_k + \Omega^2 e_k \]

is forced to zero. In the above definition of the sliding variable, \( e_k, \hat{e}_k, \hat{\hat{e}}_k \) denote the filtered deviation error (pv – sp) and its first and second derivatives in the control period \( k \), respectively, and \( \xi, \Omega \) are the control parameters described below. In the second phase, \( s_k \) is held at the zero value (the sliding phase) by the proper control "bangs". Here, the heating action is alternated by cooling action and vice versa rapidly. The amplitudes of control actions are adapted appropriately to guarantee \( s_k = 0 \) approximately. Thus, the hypothetical continuous dynamic sliding variable

\[ s = \triangle \dot{e} + 2\xi \dot{e} + \Omega^2 e \]

is approximately equal to zero at any time. Therefore the control deviation behaves according to the second order differential equation

\[ s = \triangle \dot{e} + 2\xi \dot{e} + \Omega^2 e = 0 \]

describing so called zero sliding dynamics. From it follows that the evolution of \( e \) can be prescribed by the parameters \( \xi, \Omega \). For stable behavior, it must hold \( \xi > 0, \Omega > 0 \).

A typical optimal value of \( \xi \) ranges in the interval \([4, 8]\) and \( \xi \) about 6 is often a satisfactory value. The optimal value of \( \Omega \) strongly depends on the controlled process. The slower processes the lower optimal \( \Omega \). The recommended value of \( \Omega \) for start of tuning is \( \pi/(5T_C) \).

The manipulated variable \( MV \) usually ranges in the interval \([-1, 1]\). The positive (negative) value corresponds to heating (cooling). For example, \( MV = 1 \) means the full heating. The limits of \( MV \) can be reduced when needed by the controller parameters \( \text{hilim}_p \) and \( \text{hilim}_m \). This reduction is probably necessary when the asymmetry between heating and cooling is significant. For example, if in the working zone the cooling is much more aggressive than heating, then these parameters should be set as \( \text{hilim}_p = 1 \) and \( \text{hilim}_m < 1 \). If we want to apply such limitation only in some time interval after a change of setpoint (during the transient response) then it is necessary to set initial value of the heating (cooling) action amplitude \( u_0_p \) (\( u_0_m \)) to the suitable value less than \( \text{hilim}_p \) (\( \text{hilim}_m \)). Otherwise set \( u_0_p = \text{hilim}_p \) and \( u_0_m = \text{hilim}_m \).

The current amplitudes of heating and cooling \( UK_p, UK_m \), respectively, are automatically adapted by the special algorithm to achieve so called quasi sliding mode, where the sign of \( s_k \) alternately changes its value. In such a case the controller output \( MV \) alternates the values 1 and -1. The rate of adaptation of the heating (cooling) amplitude is given by the time constant \( \tau_{up} \) (\( \tau_{down} \)). Both of these time constants have to be sufficiently high to provide the proper function of adaptation but the fine tuning is not necessary.
Note for completeness that the manipulated variable $mv$ is determined from the action amplitudes $uk_p$, $uk_m$ by the following expression

$$\text{if } (s_k < 0.0) \text{ then } mv = uk_p \text{ else } mv = -uk_m.$$ 

Further, it is important to note that quasi sliding is seldom achievable because of a process dead time or disturbances. The suitable indicator of the quality of sliding is again the output $isv$. If the extraordinary fine tuning is required then it may be tried to find the better value for the bandwidth parameter $\beta$ of derivative filter, otherwise the default value 0.1 is preferred. In the manual mode ($\text{MAN = on}$) the controller input $hv$ is (after limitation to the range $[-\text{hilim}_m, \text{hilim}_p]$) copied to the manipulated variable $mv$.

**Inputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sp$</td>
<td>setpoint variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$pv$</td>
<td>process variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$hv$</td>
<td>manual value</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$\text{MAN}$</td>
<td>controller mode</td>
<td>Bool</td>
</tr>
</tbody>
</table>

0 ...... automatic mode 1 ...... manual mode

**Outputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$mv$</td>
<td>manipulated variable (position controller output)</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$mv_e$</td>
<td>equivalent manipulated variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$de$</td>
<td>deviation error</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>SAT</td>
<td>saturation flag</td>
<td>Bool</td>
</tr>
</tbody>
</table>

0 ...... the controller implements a linear control law 1 ...... the controller output is saturated, $mv \geq \text{hilim}_p$ or $mv \leq -\text{hilim}_m$

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$isv$</td>
<td>number of the positive (+) or negative (−) sliding variable steps</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>$t_{ukp}$</td>
<td>current amplitude of heating</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$t_{ukm}$</td>
<td>current amplitude of cooling</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$t_{sk}$</td>
<td>discrete dynamic sliding variable $s_k$</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$t_{pv}$</td>
<td>filtered control error $-de$</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$t_{dpv}$</td>
<td>filtered first derivative of the control error $t_{sk}$</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$t_{d2pv}$</td>
<td>filtered second derivative of the control error $t_{sk}$</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ipwmc$</td>
<td>PWM cycle in the sampling periods of SMHCC ($T_C/T_S$)</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>$xi$</td>
<td>relative damping $\xi$ of sliding zero dynamics $xi \geq 0$</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$cm$</td>
<td>natural frequency $\Omega$ of sliding zero dynamics $\downarrow(0.0)$</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$taup$</td>
<td>time constant for adaptation of heating action amplitude in seconds</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td>Minimum</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>$t_{aum}$</td>
<td>time constant for adaptation of cooling action amplitude in seconds</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>bandwidth parameter of the derivative filter</td>
<td>( \downarrow 0 )</td>
</tr>
<tr>
<td>$h_{ilim_p}$</td>
<td>high limit of the heating action amplitude</td>
<td>( \downarrow 0.0 )</td>
</tr>
<tr>
<td>$h_{ilim_m}$</td>
<td>high limit of the cooling action amplitude</td>
<td>( \downarrow 0.0 )</td>
</tr>
<tr>
<td>$u_{0_p}$</td>
<td>initial value of the heating action amplitude after setpoint change and start of the block</td>
<td></td>
</tr>
<tr>
<td>$u_{0_m}$</td>
<td>initial value of the cooling action amplitude after setpoint change and start of the block</td>
<td></td>
</tr>
<tr>
<td>$sp_dif$</td>
<td>Setpoint difference threshold</td>
<td>( \geq 10.0 )</td>
</tr>
<tr>
<td>$t_{auf}$</td>
<td>Equivalent manipulated variable filter time constant</td>
<td>( \geq 400.0 )</td>
</tr>
</tbody>
</table>
SMHCCA – Sliding mode heating/cooling controller with autotuner

Function Description

The sliding mode heating/cooling controller (SMHCCA) is a novel high quality control algorithm with a built-in autotuner for automatic tuning of the controller parameters. The controller is mainly intended for temperature control of heating-cooling (possibly asymmetrical) processes with ON-OFF heaters and/or ON-OFF coolers. The plastic extruder heating/cooling system is a typical example of such processes. However, it can also be applied to many similar cases, for example, to thermal systems where a conventional thermostat is normally employed. To provide the proper control function, the SMHCCA block must be combined with the PWM block (Pulse Width Modulation) as depicted in the following figure.

It is important to note that the block SMHCCA works with two time periods. The first period $T_S$ is the sampling time of the process temperature, and this period is equal to the period with which the block SMHCCA itself is executed. The other period $T_C = iparmc T_S$ is the control period with which the block SMHCCA generates the manipulated variable. This period $T_C$ is equal to the cycle time of the PWM block. At every instant when the manipulated variable $\text{mv}$ is changed by SMHCCA the PWM algorithm recalculates the width of the output
pulse and starts a new PWM cycle. The time resolution \( T_R \) of the PWM block is third
time period involved in. This period is equal to the period with which the block PWM is
executed and generally may be different from \( T_S \). To achieve the high quality of control
it is recommended to choose \( T_S \) as minimal as possible (\( \tau_{PWM} \) as maximal as possible),
the ratio \( T_C/T_S \) as maximal as possible but \( T_C \) should be sufficiently small with respect
to the process dynamics. An example of reasonable values for an extruder temperature
control is as follows:

\[
T_S = 0.1, \ \tau_{PWM} = 50, \ \tau_C = 5 s, \ \tau_R = 0.1 s.
\]

Notice however that for a faster controlled system the sampling periods \( T_S \), \( \tau_C \) and
\( \tau_R \) must be shortened! More precisely, the three minimal time constant of the process
are important for selection of these time periods (all real thermal process has at least
three time constants). For example, the sampling period \( T_S = 0.1 \) is sufficiently short for
such processes that have at least three time constants, the minimal of them is greater
than 10s and the maximal is greater than 100s. For the proper function of the controller
it is necessary that these time parameters are suitably chosen by the user according
to the actual dynamics of the process! If SMHCCA is implemented on a processor with
floating point arithmetic then the accurate setting of the sampling periods \( T_S \), \( \tau_C \), \( \tau_R \)
and the parameter \( \text{beta} \) is critical for correct function of the controller. Also, some other
parameters with the clear meaning described below have to be chosen manually. All the
remaining parameters (\( \xi \), \( \omega \), \( \tau_{up} \), \( \tau_{am} \), \( \tau_{af} \)) can be set by the built-in autotuner
automatically. The autotuner uses the two methods for this purpose.

- The first one is dedicated to situations where the asymmetry of the process is
  not enormous (approximately, it means that the gain ratio of heating/cooling or
  cooling/heating is less than 5).

- The second method provides the tuning support for the strong asymmetric pro-
  cesses and is not implemented yet (So far, this method has been developed and
tested in Simulink only).

Despite the fact that the first method of the tuning is based only on the heating
regime, the resulting parameters are usually satisfactory for both heating and cooling
regimes because of the strong robustness of sliding mode control. The tuning proce-
dure is very quick and can be accomplished during the normal rise time period of the
process temperature from cold state to the setpoint usually without any temporization
or degradation of control performance. Thus the tuning procedure can be included in
every start up from cold state to the working point specified by the sufficiently high
temperature setpoint. Now the implemented procedure will be described in detail. The
tuning procedure starts in the tuning mode or in the manual mode. If the tuning mode
(\( \text{TMODE} = \text{on} \)) is selected the manipulated variable \( MV \) is automatically set to zero and
the output \( TBSY \) is set to 1 for indication of the tuning stage of the controller. The cold
state of the process is preserved until the initialization pulse is applied to the input \( \text{TUNE} \)
(0 \( \rightarrow \) 1). After some time (depending on \( \text{beta} \)), when the noise amplitude is estimated,
the heating is switched on with the amplitude given by the parameter ut p. The process temperature pv and its two derivatives (outputs t_pv, t_dpv, t_d2pv) are observed to obtain the optimal parameters of the controller. If the tuning procedure ends without errors, then TBSY is set to 0 and the controller begins to work in manual or automatic mode according to the input MAN. If MAN = off and affirmation input TAFF is set to 1, then the controller starts to work in automatic mode with the new parameter set provided by the tuner (if TAFF = off, then the new parameters are only displayed on the outputs p1...p6). If some error occurs during the tuning, then the tuning procedure stops immediately or stops after the condition pv>sp is fulfilled, the output TE is set to 1 and ite indicate the type of error. Also in this case, the controller starts to work in the mode determined by the input MAN. If MAN = off then works in automatic mode with the initial parameters before tuning! The tuning errors are usually caused either by an inappropriate setting of the parameter beta or by the too low value of sp. The suitable value of beta ranges in the interval (0.001,0.1). If a drift and noise in pv are large the small beta must be chosen especially for the tuning phase. The default value (beta=0.01) should work well for extruder applications. The correct value gives properly filtered signal of the second derivative of the process temperature t_d2pv. This well-filtered signal (corresponding to the low value of beta) is mainly necessary for proper tuning. For control, the parameter beta may be sometimes slightly increased. The tuning procedure may be also started from manual mode (MAN = off) with any constant value of the input iv. However, the steady state must be provided in this case. Again, the tuning is started by the initialization pulse at the input TUNE (0 → 1) and after the stop of tuning the controller continues in the manual mode. In both cases the resulting parameters appear on the outputs p1,...,p6.
The control law of the block SMICCA in automatic mode (MAN = off) is based on the discrete dynamic sliding mode control technique and special 3rd order filters for estimation of the first and second derivatives of the control error.

The first control stage, after a setpoint change or upset, is the reaching phase when the dynamic sliding variable

\[ s_k = \Delta \ddot{e}_k + 2\xi \Omega \dot{e}_k + \Omega^2 e_k \]

is forced to zero. In the above definition of the sliding variable, \( e_k, \dot{e}_k, \ddot{e}_k \) denote the filtered deviation error (\( \text{pv} - \text{sp} \)) and its first and second derivatives in the control period \( k \), respectively, and \( \xi, \Omega \) are the control parameters described below. In the second phase, \( s_k \) is hold at the zero value (the sliding phase) by the proper control "bangs". Here, the heating action is alternated by cooling action and vice versa rapidly. The amplitudes of control actions are adapted appropriately to guarantee \( s_k = 0 \) approximately. Thus, the hypothetical continuous dynamic sliding variable

\[ s = \Delta \dot{e} + 2\xi \Omega \dot{e} + \Omega^2 e \]

is approximately equal to zero at any time. Therefore the control deviation behaves according to the second order differential equation

\[ s = \Delta \dot{e} + 2\xi \Omega \dot{e} + \Omega^2 e = 0 \]

describing so called zero sliding dynamics. From it follows that the evolution of \( e \) can be prescribed by the parameters \( \xi, \Omega \). For stable behavior, it must hold \( \xi > 0, \Omega > 0 \). A typical optimal value of \( \xi \) ranges in the interval [4, 8] and \( \xi \) about 6 is often a satisfactory value. The optimal value of \( \Omega \) strongly depends on the controlled process. The slower processes the lower optimal \( \Omega \). The recommended value of \( \Omega \) for start of tuning is \( \pi/(5T_C) \).

The manipulated variable \( \text{mv} \) usually ranges in the interval \([-1, 1]\). The positive (negative) value corresponds to heating (cooling). For example, \( \text{mv} = 1 \) means the full heating. The limits of \( \text{mv} \) can be reduced when needed by the controller parameters \( \text{hlim}_p \).
and $\text{hilim}_m$. This reduction is probably necessary when the asymmetry between heating and cooling is significant. For example, if in the working zone the cooling is much more aggressive than heating, then these parameters should be set as $\text{hilim}_p = 1$ and $\text{hilim}_m < 1$. If we want to apply such limitation only in some time interval after a change of setpoint (during the transient response) then it is necessary to set initial value of the heating (cooling) action amplitude $u_{0, p}$ ($u_{0, m}$) to the suitable value less than $\text{hilim}_p$ ($\text{hilim}_m$). Otherwise set $u_{0, p} = \text{hilim}_p$ and $u_{0, m} = \text{hilim}_m$.

The current amplitudes of heating and cooling $\text{uk}_p, \text{uk}_m$, respectively, are automatically adapted by the special algorithm to achieve so called quasi sliding mode, where the sign of $s_k$ alternately changes its value. In such a case the controller output $\text{isv}$ alternates the values 1 and $-1$. The rate of adaptation of the heating (cooling) amplitude is given by time constant $\text{taup}$ ($\text{taum}$). Both of these time constants have to be sufficiently high to provide the proper function of adaptation but the fine tuning is not necessary. Note for completeness that the manipulated variable $\text{mv}$ is determined from the action amplitudes $\text{uk}_p, \text{uk}_m$ by the following expression

$$\text{if } (s_k < 0.0) \text{ then } \text{mv} = \text{uk}_p \text{ else } \text{mv} = -\text{uk}_m.$$  

Further, it is important to note that quasi sliding is seldom achievable because of a process dead time or disturbances. The suitable indicator of the quality of sliding is again the output $\text{isv}$. If the extraordinary fine tuning is required then it may be tried to find the better value for the bandwidth parameter $\text{beta}$ of derivative filter, otherwise the default value 0.1 is preferred.

In the manual mode ($\text{MAN} = \text{on}$) the controller input $\text{hv}$ is (after limitation to the range $[-\text{hilim}_m, \text{hilim}_p]$) copied to the manipulated variable $\text{mv}$. The controller output $\text{mve}$ provides the equivalent amplitude-modulated value of the manipulated variable $\text{mv}$ for informative purposes. The output $\text{mve}$ is obtained by the first order filter with the time constant $\text{taif}$ applied to $\text{mv}$.

**Inputs**

- $sp$ Setpoint variable
- $pv$ Process variable
- $hv$ Manual value
- $\text{MAN}$ Manual or automatic mode
- $\text{TMODE}$ Tuning mode
- $\text{TUNE}$ Start the tuning experiment: TUNE off $\rightarrow$ on
- $\text{TBKR}$ Stop the tuning experiment: TBKR off $\rightarrow$ on
- $\text{TAFF}$ Affirmation of the parameter set provided by the tuning procedure: TAFF = on

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sp$</td>
<td>Setpoint variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$pv$</td>
<td>Process variable</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$hv$</td>
<td>Manual value</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>$\text{MAN}$</td>
<td>Manual or automatic mode</td>
<td>Bool</td>
</tr>
<tr>
<td>$\text{TMODE}$</td>
<td>Tuning mode</td>
<td>Bool</td>
</tr>
<tr>
<td>$\text{TUNE}$</td>
<td>Start the tuning experiment: TUNE off $\rightarrow$ on</td>
<td>Bool</td>
</tr>
<tr>
<td>$\text{TBKR}$</td>
<td>Stop the tuning experiment: TBKR off $\rightarrow$ on</td>
<td>Bool</td>
</tr>
<tr>
<td>$\text{TAFF}$</td>
<td>Affirmation of the parameter set provided by the tuning procedure: TAFF = on</td>
<td>Bool</td>
</tr>
</tbody>
</table>
ips Meaning of the output signals p1,...,p6
0 ..... Controller parameters
   p1 ... recommended control period TC
   p2 ... xi
   p3 ... om
   p4 ... tau_p
   p5 ... tau_u
   p6 ... tau_f
1 ..... Auxiliary parameters
   p1 ... tpeak - time of the peak in the second derivative of pv
   p2 ... hpeak2 - peak value in the second derivative of pv
   p3 ... d2 - peak to peak amplitude of t_d2pv
   p4 ... t_gain

Outputs

mv Manipulated variable (controller output) Double (F64)
mve Equivalent manipulated variable Double (F64)
de Deviation error Double (F64)
SAT Saturation flag Bool
   0 ..... Signal not limited
   1 ..... Saturation limits active, mv ≥ hilim_p or mv ≤ -hilim_m
isv Number of the positive (+) or negative (−) sliding variable steps Long (I32)
t_ukp Current amplitude of heating Double (F64)
t_ukm Current amplitude of cooling Double (F64)
t_sk Discrete dynamic sliding variable Double (F64)
t_pv Filtered process variable pv by 3rd order filter Double (F64)
t_dpv Filtered first derivative of pv by 3rd order filter Double (F64)
t_d2pv Filtered second derivative of pv by 3rd order filter Double (F64)
TBSY Tuner busy flag (TBSY = on) Bool
TE Tuning error Bool
   off ... Autotuning successful
   on .... An error occurred during the experiment
### CHAPTER 7. REG - FUNCTION BLOCKS FOR CONTROL

**ite**  
Error code  
0       No error  
1       Noise level in pv too high, check the temperature input  
2       Incorrect parameter ut_p  
3       Setpoint sp too low  
4       The two minimal process time constants are probably too small with respect to the sampling period \( T_s \) OR too high level of noise in the second derivative of pv (try to decrease the beta parameter)  
5       Premature termination of the tuning procedure (TBK)

**pi**  
Identified parameters with respect to ips, \( i = 1, \ldots, 6 \)  
Double (F64)

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipwnc</td>
<td>PWM cycle (in sampling periods of the block, ( T_C/T_S ))</td>
<td>( \odot 100 )</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>xi</td>
<td>Relative damping of sliding zero dynamics</td>
<td>( \downarrow 0.5 \uparrow 8.0 \odot 1.0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>om</td>
<td>Natural frequency ( \omega ) of sliding zero dynamics</td>
<td>( \downarrow 0.0 \odot 0.01 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>taup</td>
<td>Time constant for adaptation of heating action amplitude [s]</td>
<td>( \odot 700.0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>taum</td>
<td>Time constant for adaptation of cooling action amplitude [s]</td>
<td>( \odot 400.0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>beta</td>
<td>Bandwidth parameter of the derivative filter</td>
<td>( \odot 0.01 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>hilim_p</td>
<td>Upper limit of the heating action amplitude</td>
<td>( \downarrow 0.0 \uparrow 1.0 \odot 1.0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>hilim_m</td>
<td>Upper limit of the cooling action amplitude</td>
<td>( \downarrow 0.0 \uparrow 1.0 \odot 1.0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>u0_p</td>
<td>Initial amplitude of the heating action</td>
<td>( \odot 1.0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>u0_m</td>
<td>Initial amplitude of the cooling action</td>
<td>( \odot 1.0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>sp_dif</td>
<td>Setpoint difference threshold for blocking of heating/cooling amplitudes reset</td>
<td>( \odot 10.0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>tauf</td>
<td>Time constant of the filter for obtaining the equivalent manipulated variable</td>
<td>( \odot 400.0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>itm</td>
<td>Tuning method</td>
<td>( \odot 1 )</td>
<td>Long (I32)</td>
</tr>
<tr>
<td></td>
<td>1       Restricted to symmetrical processes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2       Asymmetrical processes (not implemented yet)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ut_p</td>
<td>Amplitude of heating for tuning experiment</td>
<td>( \downarrow 0.0 \uparrow 1.0 \odot 1.0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>ut_m</td>
<td>Amplitude of cooling for tuning experiment</td>
<td>( \downarrow 0.0 \uparrow 1.0 \odot 1.0 )</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
**SWU – Switch unit**

**Block Symbol**

<table>
<thead>
<tr>
<th>uc</th>
<th>OR1</th>
<th>OR2</th>
<th>OR3</th>
<th>y</th>
</tr>
</thead>
</table>

**Licence:** STANDARD

**Function Description**

The SWU block is used to select the appropriate signal which should be tracked by the inactive PIDU and MCU units in complex control structures. The input signal uc is copied to the output y when all the binary inputs OR1, …, OR4 are off, otherwise the output y takes over the uo input signal.

**Inputs**

- **uc**: This input is copied to output y when all the binary inputs OR1, OR2, OR3 and OR4 are off. Double (F64)
- **uo**: This input is copied to output y when any of the binary inputs OR1, OR2, OR3, OR4 is on. Double (F64)
- OR1: First logical output of the block. Bool
- OR2: Second logical output of the block. Bool
- OR3: Third logical output of the block. Bool
- OR4: Fourth logical output of the block. Bool

**Output**

- **y**: Analog output of the block. Double (F64)
TSE – Three-state element

Block Symbol

Function Description

The TSE block transforms the analog input \( u \) to a three-state signal ("up", "idle" and "down") according to the diagram below.

Input

\( u \)  Analog input of the block  Double (F64)

Outputs

\( \text{UP} \)  The "up" signal  Bool
\( \text{DN} \)  The "down" signal  Bool

Parameters

\( \text{ep} \)  The input value \( u > \text{ep} \) results in \( \text{UP} = \text{on} \) and \( \text{DN} = \text{off} \)  Double (F64)
\( \circ 1.0 \)
\( \text{en} \)  The input value \( u < \text{en} \) results in \( \text{UP} = \text{off} \) and \( \text{DN} = \text{off} \)  Double (F64)
\( \circ -1.0 \)
\( \text{epoff} \)  UP switch off value; if \( \text{UP} = \text{on} \) and \( u < \text{epoff} \) then \( \text{UP} = \text{off} \)  Double (F64)
\( \circ 0.5 \)
\( \text{enoff} \)  DN switch off value; if \( \text{DN} = \text{on} \) and \( u > \text{enoff} \) then \( \text{DN} = \text{off} \)  Double (F64)
\( \circ -0.5 \)
Chapter 8

LOGIC – Logic control

Contents

AND_ – Logical product of two signals . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 232
ANDQUAD, ANDOCT, ANDHEXD – Logical product of multiple signals . . . . . . . . . . 233
ATMT – Finite-state automaton . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 234
BDOCT, BDHEXD – Bitwise demultiplexers . . . . . . . . . . . . . . . . . . . . . . . . 237
BITOP – Bitwise operation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 238
BMOCT, BMHEXD – Bitwise multiplexers . . . . . . . . . . . . . . . . . . . . . . . . . . 239
COUNT – Controlled counter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 240
EATMT – Extended finite-state automaton . . . . . . . . . . . . . . . . . . . . . . . . 242
EDGE_ – Falling/rising edge detection in a binary signal . . . . . . . . . . . . . . . . . 245
EQ – Equivalence of two signals . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 246
INTSM – Integer number bit shift and mask . . . . . . . . . . . . . . . . . . . . . . . 247
ISSW – Simple switch for integer signals . . . . . . . . . . . . . . . . . . . . . . . . . . 248
INTSM – Integer number bit shift and mask . . . . . . . . . . . . . . . . . . . . . . . 249
ITOI – Transformation of integer and binary numbers . . . . . . . . . . . . . . . . . . . 250
NUT_ – Boolean complementation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 251
OR_ – Logical sum of two signals . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 252
ORQUAD, OROCT, ORHEXD – Logical sum of multiple signals . . . . . . . . . . . . . 253
RS – Reset-set flip-flop circuit . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 254
SR – Set-reset flip-flop circuit . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 255
TIMER_ – Multipurpose timer . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 256
AND_ – Logical product of two signals

Block Symbol

Function Description
The AND_ block computes the logical product of two input signals U1 and U2. If you need to work with more input signals, use the ANDGCT block.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>First logical input of the block</td>
<td>Bool</td>
</tr>
<tr>
<td>U2</td>
<td>Second logical input of the block</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>Output signal, logical product (U1 &amp; U2)</td>
<td>Bool</td>
</tr>
<tr>
<td>NY</td>
<td>Boolean complementation of Y (NY = \neg Y)</td>
<td>Bool</td>
</tr>
</tbody>
</table>
ANDQUAD, ANDOCT, ANDHEXD – Logical product of multiple signals

Block Symbols

Licence: STANDARD

Function Description

The ANDQUAD, ANDOCT and ANDHEXD blocks compute the logical product of up to sixteen input signals U1, U2, ..., U16. The signals listed in the n1 parameter are negated prior to computing the logical product.

For an empty n1 parameter a simple logical product \( Y = U_1 \land U_2 \land U_3 \land U_4 \land U_5 \land U_6 \land U_7 \land U_8 \) is computed. For e.g. \( n1=1,3...5 \), the logical function is \( Y = \overline{U_1} \land U_2 \land \overline{U_3} \land \overline{U_4} \land \overline{U_5} \land \overline{U_6} \land \overline{U_7} \land \overline{U_8} \).

If you have less than 4/8/16 signals, use the n1 parameter to handle the unconnected inputs. If you have only two input signals, consider using the AND_ block.

Inputs

\( U_1..U_{16} \) Logical inputs of the block \( \text{Bool} \)

Outputs

\( Y \) Result of the logical operation \( \text{Bool} \)
\( \overline{Y} \) Boolean complementation of Y \( \text{Bool} \)

Parameter

\( n1 \) List of signals to negate. The format of the list is e.g. 1, 3..5, 8. \( \text{Long} (132) \)

Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.
ATMT – Finite-state automaton

Function Description

The ATMT block implements a finite state machine with at most 16 states and 16 transition rules.

The current state of the machine $i$, $i = 0, 1, \ldots, 15$ is indicated by the binary outputs $Q_0, Q_1, \ldots, Q_{15}$. If the state $i$ is active, the corresponding output is set to $Q_i$=on. The current state is also indicated by the $k_{sa}$ output ($k_{sa} \in \{0, 1, \ldots, 15\}$).

The transition conditions $C_k$, $k = 0, 1, \ldots, 15$ are activated by the binary inputs $C_0$, $C_1$, $\ldots$, $C_{15}$. If $C_k$ = on the k-th transition condition is fulfilled. The transition cannot happen when $C_k$ = off.

The automat function is defined by the following table of transitions:

<table>
<thead>
<tr>
<th>$S_n$</th>
<th>$C_n$</th>
<th>$FS_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$C_1$</td>
<td>$FS_1$</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$C_2$</td>
<td>$FS_2$</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$S_n$</td>
<td>$C_n$</td>
<td>$FS_n$</td>
</tr>
</tbody>
</table>

Each row of this table represents one transition rule. For example the first row

$S_1$  $C_1$  $FS_1$

has the meaning

$\text{If (}S_1\text{ is the current state AND transition condition }C_1\text{ is fulfilled)}$

$\text{then proceed to the following state }FS_1.$

The above mentioned table can be easily constructed from the automat state diagram or SFC description (Sequential Function Charts, formerly Grafcet).

The $R_1$ = on input resets the automat to the initial state $S_0$. The SET input allows manual transition from the current state to the ns0 state when rising edge occurs. The
R1 input overpowers the SET input. The HLD = on input freezes the automaton activity, the automaton stays in the current state regardless of the Ci input signals and the tstep timer is not incremented. The TOUT output indicates that the machine remains in the given state longer than expected. The time limits TOi for individual states are defined by the touts array. There is no time limit for the given state when TOi is set to zero. The TOUT output is set to off whenever the automaton changes its state.

It is possible to allow more state transitions in one cycle by the morestps parameter. However, this option must be thoroughly considered and tested, namely when the TOUT output is used in transition conditions. In such a case it is strongly recommended to incorporate the ksa output in the transition conditions as well.

The development tools of the REXYGEN system include also the SFCEditor program. You can create SFC schemes graphically using this tool. Run this editor from REXYGEN Studio by clicking the Configure button in the parameter dialog of the ATMT block.

Inputs

R1 Reset signal, R1 = on brings the automaton to the initial state S0; the R1 input overpowers the SET input
ns0 This state is reached when rising edge occurs at the SET input
SET The rising edge of this signal forces the transition to the ns0 state
HLD The HLD = on freezes the automaton, no transitions occur regardless of the input signals, tstep is not increasing
C0...C15 The transition conditions; Ci = on means that the i-th condition was fulfilled and the corresponding transition rule can be executed

Outputs

Q0...Q15 Output signals indicating the current state of the automaton; the current state i is indicated by Qi = on
ksa Integer code of the active state
Tstep Time elapsed since the current state was reached; the timer is set to 0 whenever a state transition occurs
TOUT Flag indicating that the time limit for the current state was exceeded

Parameters

morestps Allow multiple transitions in one cycle of the automat
off ... Disabled
on .... Enabled
ntr Number of state transition table rows
sfcname Filename of block configurator data file (filename is generated by system if parameter is empty)
<table>
<thead>
<tr>
<th>STT</th>
<th>State transition table (matrix)</th>
<th>Byte (U8)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( [0 \ 0 \ 1; \ 1 \ 1 \ 2; \ 2 \ 2 \ 3; \ 3 \ 3 \ 0] )</td>
</tr>
<tr>
<td>touts</td>
<td>Vector of timeouts TO0, TO1, ..., TO15 for the states S0, S1, ..., S15</td>
<td>Double (F64)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \ 16] )</td>
</tr>
</tbody>
</table>
BDOCT, BDHEXD – Bitwise demultiplexers

Block Symbols

Licence: STANDARD

Function Description

Both BDOCT and BDHEXD are bitwise demultiplexers for easy decomposition of the input signal to individual bits. The only difference is the number of outputs, the BDOCT block has 8 Boolean outputs while the BDHEXD block offers 16-bit decomposition. The output signals \( y_i \) correspond with the individual bits of the input signal \( iu \), the \( Y0 \) output is the least significant bit.

Input

\( iu \)  
Input signal to be decomposed  
Long (I32)

Outputs

\( Y0 \ldots Y15 \)  
Individual bits of the input signal  
Bool

Parameter

\( \text{shift} \)  
Bit shift of the input signal  
\( \Downarrow 0 \uparrow 31 \)  
Long (I32)
BITOP – Bitwise operation

Block Symbol

Function Description
The BITOP block performs bitwise operation $i_1 \circ i_2$ on the signals $i_1$ and $i_2$, resulting in an integer output $n$. The type of operation is selected by the $iop$ parameter described below. In case of logical negation or 2’s complements the input $i_2$ is ignored (i.e. the operation is unary).

Inputs

<table>
<thead>
<tr>
<th>$i_1$</th>
<th>First integer input of the block</th>
<th>Long (I32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_2$</td>
<td>Second integer input of the block</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>

Output

| $n$ | Result of the bitwise operation $iop$ | Long (I32) |

Parameter

<table>
<thead>
<tr>
<th>$iop$</th>
<th>Bitwise operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bitwise negation (Bit NOT)</td>
</tr>
<tr>
<td>2</td>
<td>Bitwise logical sum (Bit OR)</td>
</tr>
<tr>
<td>3</td>
<td>Bitwise logical product (Bit AND)</td>
</tr>
<tr>
<td>4</td>
<td>Bitwise logical exclusive sum (Bit XOR)</td>
</tr>
<tr>
<td>5</td>
<td>Shift of the $i_1$ signal by $i_2$ bits to the left (Shift Left)</td>
</tr>
<tr>
<td>6</td>
<td>Shift of the $i_1$ signal by $i_2$ bits to the right (Shift Right)</td>
</tr>
<tr>
<td>7</td>
<td>2’s complement of the $i_1$ signal on 8 bits (2’s Complement - Byte)</td>
</tr>
<tr>
<td>8</td>
<td>2’s complement of the $i_1$ signal on 16 bits (2’s Complement - Word)</td>
</tr>
<tr>
<td>9</td>
<td>2’s complement of the $i_1$ signal on 32 bits (2’s Complement - Long)</td>
</tr>
</tbody>
</table>
BMOCT, BMHEXD – Bitwise multiplexers

Block Symbols

Function Description
Both BMOCT and BMHEXD are bitwise multiplexers for easy composition of the output signal from individual bits. The only difference is the number of inputs, the BMOCT block has 8 Boolean inputs while the BMHEXD block offers 16-bit composition. The input signals \( U_i \) correspond with the individual bits of the output signal \( iy \), the \( U_0 \) input is the least significant bit.

Inputs

\[ U_0 \ldots U_{15} \quad \text{Individual bits of the output signal} \quad \text{Bool} \]

Output

\[ iy \quad \text{Composed output signal} \quad \text{Long (132)} \]

Parameter

\[ \text{shift} \quad \text{Bit shift of the output signal} \quad \downarrow 0 \uparrow 31 \quad \text{Long (132)} \]
COUNT – Controlled counter

Block Symbol

Licence: STANDARD

Function Description

The COUNT block is designed for bidirectional pulse counting – more precisely, counting rising edges of the UP and DN input signals. When a rising edge occurs at the UP (DN) input, the \texttt{cnt} output is incremented (decremented) by 1. Simultaneous occurrence of rising edges at both inputs is indicated by the error output \texttt{E} set to \texttt{on}. The \texttt{R1} input resets the counter to 0 and no addition or subtraction is performed unless the \texttt{R1} input returns to \texttt{off} again. It is also possible to set the output \texttt{cnt} to the value \texttt{n0} by the SETH input. Again, no addition or subtraction is performed unless the SETH input returns to \texttt{off} again. The \texttt{R1} input has higher priority than the SETH input. The input HLD = \texttt{on} prevents both incrementing and decrementing. When the counter reaches the value \texttt{cnt} ≥ \texttt{nmax}, the \texttt{Q} output is set to \texttt{on}.

Inputs

- \texttt{R1} Block reset (\texttt{R1 = on}) : \texttt{Bool}
- \texttt{n0} Value to set the counter to (using the SETH input) : \texttt{Long (I32)}
- \texttt{SETH} Set the counter value to \texttt{n0} (SETH = \texttt{on}) : \texttt{Bool}
- \texttt{UP} Incrementing input signal : \texttt{Bool}
- \texttt{DN} Decrementing input signal : \texttt{Bool}
- \texttt{HLD} Counter freeze : \texttt{Bool}
  - \texttt{off} ... Counter is running
  - \texttt{on} ... Counter is locked
- \texttt{nmax} Counter target value : \texttt{Long (I32)}

Outputs

- \texttt{cnt} Total number of pulses : \texttt{Long (I32)}
- \texttt{SGN} Sign of the \texttt{cnt} output
  - \texttt{off} ... for \texttt{cnt} < 0
  - \texttt{on} ... for \texttt{cnt} ≥ 0
- \texttt{Q} Target value indicator
  - \texttt{off} ... for \texttt{cnt} < \texttt{nmax}
  - \texttt{on} ... for \texttt{cnt} ≥ \texttt{nmax}
E Indicator of simultaneous occurrence of rising edges at both Bool inputs UP and DN
EATMT – Extended finite-state automaton

Function Description

The EATMT block implements a finite automat with at most 256 states and 256 transition rules, thus it extends the possibilities of the ATMT block.

The current state of the automaton \( i, i = 0, 1, \ldots, 255 \) is indicated by individual bits of the integer outputs \( q_0, q_1, \ldots, q_{15} \). Only a single bit with index \( i \mod 16 \) of the \( q(i \div 16) \) output is set to 1. The remaining bits of that output and the other outputs are zero. The bits are numbered from zero, least significant bit first. Note that the DIV and MOD operators denote integer division and remainder after integer division respectively. The current state is also indicated by the ksa \( \in \{0, 1, \ldots, 255\} \) output.

The transition conditions \( C_k, k = 0, 1, \ldots, 255 \) are activated by individual bits of the inputs \( c_0, c_1, \ldots, c_{15} \). The \( k \)-th transition condition is fulfilled when the \( (k \mod 16) \)-th bit of the input \( c(k \div 16) \) is equal to 1. The transition cannot happen otherwise.

The BMHEXD or BMDCST bitwise multiplexers can be used for composition of the input signals \( c_0, c_1, \ldots, c_{15} \) from individual Boolean signals. Similarly the output signals \( q_0, q_1, \ldots, q_{15} \) can be decomposed using the BDHEXO or BDQCT bitwise demultiplexers.

The automat function is defined by the following table of transitions:

\[
\begin{align*}
S_1 & \quad C_1 & \quad FS_1 \\
S_2 & \quad C_2 & \quad FS_2 \\
& \quad \cdots \\
S_n & \quad C_n & \quad FS_n
\end{align*}
\]

Each row of this table represents one transition rule. For example the first row

\[
S_1 \quad C_1 \quad FS_1
\]

has the meaning

If \((S_1 \text{ is the current state AND transition condition } C_1 \text{ is fulfilled})\)

then proceed to the following state \( FS_1 \).
The above described meaning of the table row holds for $C1 < 1000$. Negation of the 
$(C1 - 1000)$-th transition condition is assumed for $C1 \geq 1000$.

The above mentioned table can be easily constructed from the automat state diagram
or SFC description (Sequential Function Charts, formerly Grafcet).

The $R1 = \text{on}$ input resets the automat to the initial state $S0$. The $\text{SET}$ input allows
manual transition from the current state to the $\text{ns0}$ state when rising edge occurs. The
$R1$ input overpowers the $\text{SET}$ input. The $\text{HLD} = \text{on}$ input freezes the automat activity,
the automat stays in the current state regardless of the $ci$ input signals and the $\text{tstep}$
timer is not incremented. The $\text{TOUT}$ output indicates that the machine remains in the
given state longer than expected. The time limits $TOi$ for individual states are defined
by the $\text{touts}$ array. There is no time limit for the given state when $TOi$ is set to zero.
The $\text{TOUT}$ output is set to $\text{off}$ whenever the automat changes its state.

It is possible to allow more state transitions in one cycle by the $\text{morestps}$ parameter.
However, this option must be thoroughly considered and tested, namely when the $\text{TOUT}$
output is used in transition conditions. In such a case it is strongly recommended to
incorporate the $\text{ksa}$ output in the transition conditions as well.

The development tools of the REXYGEN system include also the SFCEditor program.
You can create SFC schemes graphically using this tool. Run this editor from REXYGEN
Studio by clicking the $\text{Configure}$ button in the parameter dialog of the $\text{EATMT}$ block.

Inputs

- $R1$ : Reset signal, $R1 = \text{on}$ brings the automat to the initial state $S0$; $\text{Bool}$
  the $R1$ input overpowers the $\text{SET}$ input
- $\text{ns0}$ : This state is reached when rising edge occurs at the $\text{SET}$ input $\text{Long}$ (132)
- $\text{SET}$ : The rising edge of this signal forces the transition to the $\text{ns0}$ $\text{Bool}$
  state
- $\text{HLD}$ : The $\text{HLD} = \text{on}$ freezes the automat, no transitions occur $\text{Bool}$
  regardless of the input signals, $\text{tstep}$ is not increasing
- $c0...c15$ : Transition conditions, each input signal contains 16 transition
  conditions, see details above

Outputs

- $q0...q15$ : Output signals indicating the current state of the automat, see $\text{Long}$ (132)
  details above
- $\text{ksa}$ : Integer code of the active state $\text{Long}$ (132)
- $\text{tstep}$ : Time elapsed since the current state was reached; the timer is
  set to 0 whenever a state transition occurs $\text{Double}$ (P64)
- $\text{TOUT}$ : Flag indicating that the time limit for the current state was
  exceeded $\text{Bool}$
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>morestps</code></td>
<td>Allow multiple transitions in one cycle of the automat</td>
<td><code>Bool</code></td>
</tr>
<tr>
<td></td>
<td><strong>off</strong> ... Disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>on</strong> ... Enabled</td>
<td></td>
</tr>
<tr>
<td><code>ntr</code></td>
<td>Number of state transition table rows</td>
<td><code>Long (I32)</code></td>
</tr>
<tr>
<td><code>sfcname</code></td>
<td>Filename of block configurator data file (filename is generated by system if parameter is empty)</td>
<td><code>String</code></td>
</tr>
<tr>
<td><code>STT</code></td>
<td>State transition table (matrix)</td>
<td><code>Short (I16)</code></td>
</tr>
<tr>
<td></td>
<td>○[0 0 1; 1 1 2; 2 2 3; 3 3 0]</td>
<td></td>
</tr>
<tr>
<td><code>touts</code></td>
<td>Vector of timeouts T00, T01, ..., T0255 for the states S0, S1, ... , S255</td>
<td><code>Double (F64)</code></td>
</tr>
<tr>
<td></td>
<td>○[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16]</td>
<td></td>
</tr>
</tbody>
</table>
EDGE_ – Falling/rising edge detection in a binary signal

Block Symbol

```
\[ \text{EDGE}_n \]
```

Licence: STANDARD

Function Description

The EDGE_ block detects rising (off→on) and/or falling (on→off) edges in the binary input signal U. The type of edges to detect is determined by the iedge parameter. As long as the input signal remains constant, the output Y is off. In the case when an edge corresponding with the iedge parameter is detected, the output Y is set to on for one sampling period.

Input

\( U \) Logical input of the block \( \text{Bool} \)

Output

\( Y \) Logical output of the block \( \text{Bool} \)

Parameter

\( \text{iedge} \) Type of edges to detect

\( \begin{array}{c}
1 \quad \ldots \quad \text{Rising edge} \\
2 \quad \ldots \quad \text{Falling edge} \\
3 \quad \ldots \quad \text{Both edges}
\end{array} \)

\( \odot 1 \) Long (132)
EQ – Equivalence of two signals

Function Description
The block compares two input signals and \( Y=\text{on} \) is set if both signals have the same value. Both signals must be either of a numeric type or strings. A conversion between numeric types is performed: for example 2.0 (double) and 2 (long) are evaluated as equivalent. Comparison of matrices or other complex types is not supported.

Inputs
- \( u_1 \): Block input signal
- \( u_2 \): Block input signal

Output
- \( Y \): Output signal
INTSM – Integer number bit shift and mask

Block Symbol

Function Description
The INTSM block performs bit shift of input value \( i \) by \( \text{shift} \) bits right (if \( \text{shift} \) is positive) or left (if \( \text{shift} \) is negative). Free space resulting from shifting is filled with zeros.

Output value \( n \) is calculated as bitwise AND of shifted input \( i \) and bit mask \( \text{mask} \).

Typical application of this block is extraction of one or more adjacent bits from a given position in integer register which was read from some external system.

Input
\[
i \quad \text{Integer value to shift and mask} \quad \downarrow 9.22337E+18 \uparrow 9.22337E+18 \quad \text{Large (I64)}
\]

Parameters
\[
\begin{align*}
\text{shift} & \quad \text{Bit shift (negative=left, positive=right)} \quad \downarrow 63 \uparrow 63 \quad \text{Long (I32)} \\
\text{mask} & \quad \text{Bit mask (applied after bit shift)} \quad \downarrow 0 \uparrow 4294970000 \odot 4294967295 \quad \text{Large (I64)}
\end{align*}
\]

Output
\[
n \quad \text{Resulting integer value} \quad \text{Large (I64)}
\]
ISSW – Simple switch for integer signals

Block Symbol

Licence: STANDARD

Function Description

The ISSW block is a simple switch for integer input signals $i_1$ and $i_2$ whose decision variable is the binary input $SW$. If $SW$ is off, then the output $n$ is equal to the $i_1$ signal. If $SW$ is on, then the output $n$ is equal to the $i_2$ signal.

Inputs

<table>
<thead>
<tr>
<th>$i_1$</th>
<th>First integer input of the block</th>
<th>Long (I32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_2$</td>
<td>Second integer input of the block</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>$SW$</td>
<td>Signal selector</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... The $i_1$ signal is selected</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... The $i_2$ signal is selected</td>
<td></td>
</tr>
</tbody>
</table>

Output

| $n$  | Integer output of the block     | Long (I32) |
INTSM – Integer number bit shift and mask

Block Symbol

Licence: STANDARD

Function Description

The INTSM block performs bit shift of input value i by shift bits right (if shift is positive) or left (if shift is negative). Free space resulting from shifting is filled with zeros.

Output value n is calculated as bitwise AND of shifted input i and bit mask mask.

Typical application of this block is extraction of one or more adjacent bits from a given position in integer register which was read from some external system.

Input

\[ i \quad \text{Integer value to shift and mask} \quad \downarrow_{-9.22337E+18} \uparrow_{9.22337E+18} \quad \text{Large (I64)} \]

Parameters

\[ \text{shift} \quad \text{Bit shift (negative=left, positive=right)} \quad \downarrow_{-63} \uparrow_{63} \quad \text{Long (I32)} \]
\[ \text{mask} \quad \text{Bit mask (applied after bit shift)} \quad \downarrow_{0} \uparrow_{4294970000} \quad \text{Large (I64)} \]

Output

\[ n \quad \text{Resulting integer value} \quad \text{Large (I64)} \]
**ITOI – Transformation of integer and binary numbers**

Block Symbol

**Function Description**

The ITOI block transforms the input number \( k \), or the binary number \((U3 \ U2 \ U1 \ U0)_2\), from the set \{0, 1, 2, \ldots, 15\} to the output number \( nk \) and its binary representation \((Y3 \ Y2 \ Y1 \ Y0)_2\) from the same set. The transformation is described by the following table

\[
\begin{array}{c|ccccccc}
\text{k} & 0 & 1 & 2 & \ldots & 15 \\
\hline
\text{nk} & n0 & n1 & n2 & \ldots & n15
\end{array}
\]

where \( n0, \ldots, n15 \) are given by the mapping table target vector \( \text{fktab} \).

If \( \text{BINF} = \text{off} \), then the integer input \( k \) is active, while for \( \text{BINF} = \text{on} \) the input is defined by the binary inputs \((U3 \ U2 \ U1 \ U0)_2\).

**Inputs**

- \( k \): Integer input of the block
  - Long (I32)
- \( U0 \): Binary input digit, weight of 1
  - Bool
- \( U1 \): Binary input digit, weight of 2
  - Bool
- \( U2 \): Binary input digit, weight of 4
  - Bool
- \( U3 \): Binary input digit, weight of 8
  - Bool

**Outputs**

- \( nk \): Integer output of the block
  - Long (I32)
- \( Y0 \): Binary output digit, weight of 1
  - Bool
- \( Y1 \): Binary output digit, weight of 2
  - Bool
- \( Y2 \): Binary output digit, weight of 4
  - Bool
- \( Y3 \): Binary output digit, weight of 8
  - Bool

**Parameters**

- \( \text{BINF} \): Enable the binary selectors
  - \( \odot \text{on} \) bool
  - \( \text{off} \ldots \) Disabled (integer input \( k \))
  - \( \text{on} \ldots \) Enabled (binary input signals \( U3 \ldots U0 \))
- \( \text{fktab} \): Vector of mapping table target values
  - Byte (U8)
  - \( \odot [0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15] \)
**NOT** – Boolean complementation

Block Symbol

\[ \text{NOT} \]

Function Description

The **NOT** block negates the input signal.

Input

\[ U \]

Logical input of the block

\[ U \]

Bool

Output

\[ Y \]

Logical output of the block \(Y = \lnot U\)

\[ Y \]

Bool
ORₗ – Logical sum of two signals

Block Symbol

\[
\begin{array}{c}
\text{U1} \\
\text{U2} \\
\text{OR}
\end{array}
\]

Licence: STANDARD

Function Description

The OR block computes the logical sum of two input signals U₁ and U₂.

If you need to work with more input signals, use the ORGCT block.

Inputs

\begin{align*}
\text{U1} & \quad \text{First logical input of the block} & \text{U1} & \text{Boo1} \\
\text{U2} & \quad \text{Second logical input of the block} & \text{U2} & \text{Boo1}
\end{align*}

Outputs

\begin{align*}
\text{Y} & \quad \text{Logical output of the block } (\text{U1} \lor \text{U2}) & \text{Y} & \text{Boo1} \\
\text{NY} & \quad \text{Boolean complementation of } \text{Y} (\text{NY} = \neg \text{Y}) & \text{NY} & \text{Boo1}
\end{align*}
ORQUAD, OROCT, ORHEXD – Logical sum of multiple signals

Block Symbols

Licence: STANDARD

Function Description

The ORQUAD, OROCT and ORHEXD blocks compute the logical sum of up to sixteen input signals U1, U2, ..., U16. The signals listed in the n1 parameter are negated prior to computing the logical sum.

For an empty n1 parameter a simple logical sum \( Y = U_1 \lor U_2 \lor U_3 \lor U_4 \lor U_5 \lor U_6 \lor U_7 \lor ... \lor U_{16} \) is computed. For e.g. n1=1,3..5, the logical function is \( Y = \neg U_1 \lor U_2 \lor \neg U_3 \lor \neg U_4 \lor \neg U_5 \lor U_6 \lor ... \lor U_{16} \).

If you have only two input signals, consider using the OR_ block.

Inputs

\( U_1 \ldots U_{16} \) Logical inputs of the block  

Outputs

\( Y \) Result of the logical operation  
\( NY \) Boolean complementation of Y

Parameter

n1 List of signals to negate. The format of the list is e.g. 1,3..5,8.  

Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.
RS – Reset-set flip-flop circuit

Block Symbol

Function Description
The RS block is a flip-flop circuit, which sets its output permanently to on as soon as the input signal S is equal to on. The other input signal R1 resets the Q output to off even if the S input is on. The NQ output is simply the negation of the signal Q.

The block function is evident from the inner block structure depicted below.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Flip-flop set, sets the Q output to on</td>
<td>Bool</td>
</tr>
<tr>
<td>R1</td>
<td>Priority flip-flop reset, sets the Q output to off, overpowers the S input</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>Flip-flop circuit state</td>
<td>Bool</td>
</tr>
<tr>
<td>NQ</td>
<td>Boolean complementation of Q</td>
<td>Bool</td>
</tr>
</tbody>
</table>
SR – Set-reset flip-flop circuit

Function Description

The SR block is a flip-flop circuit, which sets its output permanently to on as soon as the input signal S1 is on. The other input signal R resets the Q output to off, but only if the S1 input is off. The NQ output is simply the negation of the signal Q.

The block function is evident from the inner block structure depicted below.

Inputs

- **S1**: Priority flip-flop set, sets the Q output to on, overpowers the R input
- **R**: Flip-flop reset, sets the Q output to off

Outputs

- **Q**: Flip-flop circuit state
- **NQ**: Boolean complementation of Q

Licence: STANDARD
**TIMER_ — Multipurpose timer**

Block Symbol

![Block Symbol](image)

Licence: **STANDARD**

**Function Description**

The `TIMER_` block either generates an output pulse of the given width `pt` (in seconds) or filters narrow pulses in the `U` input signal whose width is less than `pt` seconds. The operation mode is determined by the `mode` parameter.

The graph illustrates the behaviour of the block in individual modes for `pt = 3`:

![Behavior Graph](image)

The timer can be paused by the `HLD` input. The `R1` input resets the timer. The reset signal overpowers the `U` input.

**Inputs**

- `U` Trigger of the timer [Bool]
- `HLD` Timer hold [Bool]
- `R1` Block reset (R1 = on) [Bool]

**Outputs**

- `q` Timer output [Bool]
- `et` Elapsed time [s] [Double (F64)]
- `rt` Remaining time [s] [Double (F64)]
Parameters

**mode**  
Timer mode  
1 . . . . Pulse - an output pulse of the length \(pt\) is generated upon rising edge at the \(U\) input. All input pulses during the generation of the output pulse are ignored.

2 . . . . Delayed ON - the input signal \(U\) is copied to the \(Q\) output, but the start of the pulse is delayed by \(pt\) seconds. Any pulse shorter than \(pt\) is does not pass through the block.

3 . . . . Delayed OFF - the input signal \(U\) is copied to the \(Q\) output, but the end of the pulse is delayed by \(pt\) seconds. If the break between two pulses is shorter than \(pt\), the output remains on for the whole time.

4 . . . . Delayed change - the \(Q\) output is set to the value of the \(U\) input no sooner than the input remains unchanged for \(pt\) seconds

**pt**  
Timer interval [s]  
\(0.1\)  
Double (F64)
Chapter 9

TIME – Blocks for handling time

Contents

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE_</td>
<td>Current date</td>
<td>260</td>
</tr>
<tr>
<td>DATETIME</td>
<td>Get, set and convert time</td>
<td>261</td>
</tr>
<tr>
<td>TIME</td>
<td>Current time</td>
<td>262</td>
</tr>
<tr>
<td>WSCH</td>
<td>Weekly schedule</td>
<td>264</td>
</tr>
</tbody>
</table>
DATE – Current date

Block Symbol

Licence: STANDARD

Function Description

The outputs of the DATE function block correspond with the actual date of the operating system. Use the DATETIME block for advanced operations with date and time.

Outputs

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>Year</td>
</tr>
<tr>
<td>month</td>
<td>Month</td>
</tr>
<tr>
<td>day</td>
<td>Day</td>
</tr>
<tr>
<td>dow</td>
<td>Day of week, first day of week is Sunday (1)</td>
</tr>
</tbody>
</table>

Parameter

<table>
<thead>
<tr>
<th>tz</th>
<th>Timezone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Local time</td>
</tr>
<tr>
<td>2</td>
<td>UTC</td>
</tr>
</tbody>
</table>

©1 Long (I32)
DATETIME – Get, set and convert time

Block Symbol

Licence: STANDARD

Function Description

The DATETIME block is intended for advanced date/time operations in the REXYGEN system.

It allows synchronization of the operating system clock and the clock of the REXYGEN system. When the executive of the REXYGEN system is initialized, both clocks are the same. But during long-term operation the clocks may loose synchronization (e.g. due to daylight saving time). If re-synchronization is required, the rising edge (off→on) at the SET input adjusts the clock of the REXYGEN system according to the block inputs and parameters.

It is highly recommended not to adjust the REXYGEN system time when the controlled machine/process is in operation. Unexpected behavior might occur.

If date/time reading or conversion is required, the rising edge (off→on) at the GET input triggers the action and the block outputs are updated. The outputs starting with 't' denote the total number of respective units since January 1st, 2000 UTC.

Both reading and adjusting clock can be repeated periodically if set by getper and setper parameters.

If the difference of the two clocks is below the tolerance limit settol, the clock of the REXYGEN system is not adjusted at once, a gradual synchronization is used instead. In such a case, the timing of the REXYGEN system executive is negligibly altered and the clocks synchronization is achieved after some time. Afterwards the timing of the REXYGEN executive is reverted back to normal.

For simple date/time reading use the DATE_ and TIME function blocks.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>uyear</td>
<td>Input for setting year</td>
<td>0.0E+00</td>
<td>Long (132)</td>
</tr>
<tr>
<td>umonth</td>
<td>Input for setting month</td>
<td>0.0E+00</td>
<td>Long (132)</td>
</tr>
<tr>
<td>uday</td>
<td>Input for setting day</td>
<td>0.0E+00</td>
<td>Long (132)</td>
</tr>
<tr>
<td>uhour</td>
<td>Input for setting hours</td>
<td>0.0E+00</td>
<td>Long (132)</td>
</tr>
<tr>
<td>umin</td>
<td>Input for setting minutes</td>
<td>0.0E+00</td>
<td>Long (132)</td>
</tr>
</tbody>
</table>
CHAPTER 9. TIME - BLOCKS FOR HANDLING TIME

ussec  Input for setting seconds  0.00E+00 Long (I32)
unsec  Input for setting nanoseconds  Large (I64)
       ↓-9.22E+18 ↑9.22E+18 0.00E+00
SET     Trigger for setting time  Bool
GET     Trigger for getting time  Bool

Outputs

yyyear  Year  Long (I32)
ymonth  Month  Long (I32)
yday    Day  Long (I32)
yhour   Hours  Long (I32)
ymin    Minutes  Long (I32)
ysec    Seconds  Long (I32)
ynsec   Nanoseconds  Long (I32)
ydow    Day of week  Long (I32)
ywoy    Week of year  Long (I32)
tday    Total number of days  Long (I32)
tsec    Total number of seconds  Long (I32)
tnsec   Total number of nanoseconds  Large (I64)
dsec   Number of seconds since midnight  Long (I32)

Parameters

isetmode Source for setting time  1.00E+00 Long (I32)
       1 ...... OS clock
       2 ...... Block inputs
       3 ...... The unsec input
       4 ...... The ussec input
       5 ...... The unsec input, relative
igetmode Source for getting or converting time  6.00E+00 Long (I32)
       1 ...... OS clock
       2 ...... Block inputs
       3 ...... The unsec input
       4 ...... The ussec input
       5 ...... The uday input
       6 ...... REXYGEN clock
settol Tolerance for setting the REXYGEN clock [s]  1.0 Double (F64)
setper Period for setting time [s] (0=not periodic)  0.0 Double (F64)
getper Period for getting time [s] (0=not periodic)  0.001 Double (F64)
FDDW    First day of week is Sunday  Bool
       off ... Week starts on Monday
       on ... Week starts on Sunday
tz      Timezone  1.00E+00 Long (I32)
       1 ...... Local time
       2 ...... UTC
TIME – Current time

Block Symbol

Function Description
The outputs of the TIME function block correspond with the actual time of the operating system. Use the DATETIME block for advanced operations with date and time.

Outputs

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>hour</td>
<td>Hours</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>min</td>
<td>Minutes</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>sec</td>
<td>Seconds</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>

Parameter

<table>
<thead>
<tr>
<th>tz</th>
<th>Timezone</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Local time</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>UTC</td>
<td></td>
</tr>
</tbody>
</table>

Licence: STANDARD
WSCH – Weekly schedule

Block Symbol

Function Description

The WSCH function block is a weekly scheduler for e.g. heating (day, night, eco), ventilation (high, low, off), lighting, irrigation etc. Its outputs can be used for switching individual appliances on/off or adjusting the intensity or power of the connected devices.

During regular weekly schedule the outputs iy and y reflect the values from the wst table. This table contains triplets day-hour-value. E.g. the notation [2 6.5 21.5] states that on Tuesday, at 6:30 in the morning (24-hour format), the output y will be set to 21.5. The output iy will be set to 22 (rounding to nearest integer). The individual triplets are separated by semicolons.

The days in a week are numbered from 1 (Monday) to 7 (Sunday). Higher values can be used for special daily schedules, which can be forced using the fsch input or the specdays table. The active daily program is indicated by the isch output.

Alternatively it is possible to temporarily force a specific output value using the val input and a rising edge at the SET input (off→on). When a rising edge occurs at the SET input, the val input is copied to the y output and the isch output is set to 0. The forced value remains set until:

- the next interval as defined by the wst table, or
- another rising edge occurs at the SET input, or
- a different daily schedule is forced using the fsch input.

The list of special days (specdays) can be used for forcing a special daily schedule at given dates. E.g. you can force a Sunday daily schedule on holidays, Christmas, New Year, etc. The date is entered in the YYYYMMDD format. The notation [20160328 7] thus means that on March 28th, 2016, the Sunday daily schedule should be used. Individual pairs are separated by semicolons.

The trem and ynext outputs can be used for triggering specific actions in advance, before the y and iy are changed.

The iy output is meant for direct connection to function blocks with Boolean inputs (the conversion from type long to type bool is done automatically).

The nmax parameter defines memory allocation for the wst and specdays arrays. For nmax = 100 the wst list can contain up to 100 triplets day-hour-value. In typical applications there is no need to modify the nmax parameter.
Inputs

**SET**  
Trigger for setting the y and iy outputs  
**val**  
Temporary value to set the y and iy outputs to  
**fsch**  
Forced schedule

0 .... standard weekly schedule  
1 .... Monday  
2 .... Tuesday  
.....  
7 .... Sunday  
8 and above additional daily programs as defined by the wst table

Outputs

**iy**  
Integer output value  
**y**  
Output value  
**isch**  
Daily schedule identifier  
**trem**  
Time remaining in the current section (in seconds)  
**ynext**  
Output in the next section

Parameters

**tz**  
Timezone  
1 .... Local time  
2 .... UTC  
**nmax**  
Allocated size of arrays  
**wst**  
Weekly schedule table (list of triplets day-hour-value)  
**specdays**  
List of special days (list of pairs date-daily program)
Chapter 10

ARC – Data archiving

Contents

10.1 Functionality of the archiving subsystem .......................... 268
10.2 Generating alarms and events ................................. 269
   ALB, ALBE – Alarms for Boolean value .......................... 269
   ALN, ALNI – Alarms for numerical value ......................... 271
   ARS – Archive store value ...................................... 274
10.3 Trends recording ................................................ 276
   ACD – Archive compression using Delta criterion ............... 276
   TRND – Real-time trend recording ............................... 278
   TRNDV – Real-time trend recording with vector input .......... 281
10.4 Archive management ............................................. 282
   AFLUSH – Forced archive flushing .............................. 282

The RexCore executive of the REXYGEN system consists of various interconnected subsystems (real-time subsystem, diagnostic subsystem, drivers subsystem, etc.). One of these subsystems is the archiving subsystem.

The archiving subsystem takes care of recording the history of the control algorithm. The first chapter describes the functionality of the archiving subsystem while the subsequent chapters describe the function blocks of the REXYGEN system.

The function blocks can be divided into groups by their use:

- Blocks for generating alarms and events
- Blocks for recording trends
- Blocks for handling archives
10.1 Functionality of the archiving subsystem

The archive in the REXYGEN system stores the history of events, alarms and trends of selected signals. There can be up to 15 archives in each target device. The types or archives are listed below:

**RAM memory archive** – Suitable for short-term data storage. The data access rate is very high but the data is lost on reboot.

**Archive in a backed-up memory** – Similar to the RAM archive but the data is not lost on restart. Data can be accessed fast but the capacity is usually quite limited (depends on the target platform).

**Disk archive** The disk archives are files in a proprietary binary format. The files are easily transferrable among individual platforms and the main advantage is the size, which is limited only by the capacity of the storage medium. On the other hand, the drawback is the relatively slow data access.

Not all hardware platforms support all types of archives. The individual types which are supported by the platform can be displayed in the REXYGEN Diagnostics program or in REXYGEN Studio in the Diagnostics tree view panel after clicking on the name of the target device (IP address). The supported types are listed in the lower left part of the Target tab.
10.2 Generating alarms and events

ALB, ALBI – Alarms for Boolean value

Block Symbols

Function Description

The ALB and ALBI blocks generate alarms or events when a Boolean input signal U changes. The men parameter selects whether the rising or falling or both edges in the input signal should be indicated. The iack output shows the current alarm (event) code.

The ALBI block is an extension of the ALB block as the alarms (events) are indicated also by Boolean output signals HA, LA and NACK. The type of edges to watch is selected by the men input signal and the alarms are acknowledged by the iACK input signal instead of parameters with the same name and meaning.

The events and alarms are differentiated by the lv1 parameter in the REXYGEN system. The range $1 \leq lv1 \leq 127$ is reserved for alarms. All starts, ends and acknowledgments of the alarms are stored in the archive. On the contrary, the range $128 \leq lv1 \leq 255$ indicates events. Only the start (the time instant) of the event is stored in the archive.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>Logical input of the block whose changes are watched</td>
<td>Bool</td>
</tr>
<tr>
<td>men</td>
<td>Enable alarms</td>
<td>Long (132)</td>
</tr>
<tr>
<td></td>
<td>0 ....... All alarms disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 ....... Low-alarm enabled (LA) (falling edge in the input signal U)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ....... High-alarm enabled (HA)(rising edge in the input signal U)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 ....... All alarms enabled</td>
<td></td>
</tr>
<tr>
<td>iACK</td>
<td>Acknowledge alarm</td>
<td>Byte (08)</td>
</tr>
<tr>
<td></td>
<td>1 ....... Low-alarm acknowledge</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ....... High-alarm acknowledge</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 ....... Both alarms acknowledge</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Alarm is acknowledged on rising edge</td>
<td></td>
</tr>
</tbody>
</table>
Outputs

iac  Current alarm code
     0 . . . .  All alarms inactive
     1 . . . .  Low-alarm active (LA)
     2 . . . .  High-alarm active (HA)
     256 . .  Low-alarm not acknowledged (NACK)
     512 . .  High-alarm not acknowledged (NACK)

HA    High-alarm indicator
LA    Low-alarm indicator
NACK  Alarm-not-acknowledged indicator

Parameters

arc   List of archives to store the events. The format of the list is
e.g. 1, 3, 5, 8. The event will be stored in all listed archives (see
the ARC block for details on archives numbering). Third-party
programs (Simulink, OPC clients etc.) work with an integer
number, which is a binary mask, i.e. 157 (binary 10011101) in
the mentioned case.

id    Identification code of the alarm in the archive. This identifier
must be unique in the whole target device with the REXYGEN
control system (i.e. in all archiving blocks). Disabled for id = 0.

lvl   The level of the alarms (HA and LA) which differentiates alarms
from events and defines the severity of the alarm/event

Desc  Extended description of the alarm which is displayed by the
diagnostic tools of the REXYGEN system
ALN, ALNI – Alarms for numerical value

Block Symbols

Function Description

The ALN and ALNI blocks generate two-level alarms or events when a limit value is exceeded (or not reached). There are four limit values the input signal \( u \) is compared to, namely high-limits \( h \) and \( hh \) and low-limits \( l \) and \( l1 \). The \( iac \) output shows the current alarm (event) code.

The ALNI block is an extension of the ALN block as the alarms (events) are indicated also by Boolean output signals \( HHA \), \( HA \), \( LA \) and \( LLA \) and the variables of the alarm algorithm are given by the input signals \( hys \), \( hh \), \( h \), \( l \) and \( l1 \) instead of parameters with the same name and meaning.

The events and alarms are differentiated by the \( 1v1 \) parameter in the REXYGEN system. The range \( 1 \leq 1v1 \leq 127 \) is reserved for alarms. All starts, ends and acknowledgements of the alarms are stored in the archive. On the contrary, the range \( 128 \leq 1v1 \leq 255 \) indicates events. Only the start (the time instant) of the event is stored in the archive.

Inputs

- **u**: Analog input of the block which is checked to remain within the given limits
- **hys**: Alarm hysteresis for switching the alarm off \( \downarrow \text{e}-10 \uparrow \text{e}+10 \)
- **hh**: The second high-alarm limit, must be greater than \( h \)
- **h**: High-alarm limit, must be greater than \( l \)
- **l**: Low-alarm limit, must be greater than \( l1 \)
- **l1**: The second low-alarm limit
- **iac**: Alarm is acknowledged on rising edge of the individual bits of this input/parameter. E.g. value 15 acknowledges all alarms.

**Byte (U8)**

1 . . . . . Second low-alarm acknowledge
2 . . . . . Low-alarm acknowledge
4 . . . . . High-alarm acknowledge
8 . . . . . Second high-alarm acknowledge

In case a one-level alarm is required, it is sufficient to set \( 1v12=0 \) or set the \( hh \) and \( l1 \) limits to extreme values which can never be reached by the input signal.
CHAPTER 10. ARC - DATA ARCHIVING

Outputs

<table>
<thead>
<tr>
<th>iac</th>
<th>Current alarm code. Additional bitwise combinations of the codes may appear. E.g. 12 means both high alarms.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Signal within limits</td>
</tr>
<tr>
<td>1</td>
<td>Low-alarm active</td>
</tr>
<tr>
<td>2</td>
<td>High-alarm active</td>
</tr>
<tr>
<td>4</td>
<td>Second low-alarm active</td>
</tr>
<tr>
<td>8</td>
<td>Second high-alarm active</td>
</tr>
<tr>
<td>256</td>
<td>Low-alarm not acknowledged</td>
</tr>
<tr>
<td>512</td>
<td>High-alarm not acknowledged</td>
</tr>
<tr>
<td>1024</td>
<td>Second low-alarm not acknowledged</td>
</tr>
<tr>
<td>2048</td>
<td>Second high-alarm not acknowledged</td>
</tr>
<tr>
<td>-1</td>
<td>Invalid alarm limits</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E</th>
<th>Error flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>No error</td>
</tr>
<tr>
<td>on</td>
<td>An error occurred, alarm limits disordered</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HHA</th>
<th>The second high-alarm indicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>HA</td>
<td>High-alarm indicator</td>
</tr>
<tr>
<td>LA</td>
<td>Low-alarm indicator</td>
</tr>
<tr>
<td>LLA</td>
<td>The second low-alarm indicator</td>
</tr>
<tr>
<td>NACK</td>
<td>Alarm-not-acknowledged indicator</td>
</tr>
</tbody>
</table>

Parameters

| ac1s | Alarm class (data type to store)  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bool</td>
</tr>
<tr>
<td>2</td>
<td>Byte (U8)</td>
</tr>
<tr>
<td>3</td>
<td>Short (I16)</td>
</tr>
<tr>
<td>4</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>5</td>
<td>Word (U16)</td>
</tr>
<tr>
<td>6</td>
<td>DWord (U64)</td>
</tr>
<tr>
<td>7</td>
<td>Large (I64)</td>
</tr>
<tr>
<td>8</td>
<td>Float (F32)</td>
</tr>
<tr>
<td>9</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

| arc | List of archives to store the events. The format of the list is e.g. 1, 3, 5, 8. The event will be stored in all listed archives (see the ARC block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case. |

| id  | Identification code of the alarm in the archive. This identifier must be unique in the whole target device with the REXXYGEN control system (i.e. in all archiving blocks). Disabled for id = 0. |

| lv11 | The level of first high- and low-alarm alarms (HA and LA) which differentiates alarms from events and defines the severity of the alarm/event |

| lv12 | The level of second high- and low-alarm alarms (HHA and LLA) which differentiates alarms from events and defines the severity of the alarm/event |
10.2. GENERATING ALARMS AND EVENTS

Desc  Extended description of the alarm which is displayed by the diagnostic tools of the REXYGEN system
String
⊙Alarm Description
ARS – Archive store value

Block Symbol

Function Description

The block allow to store value into archive subsystem. Written value must be connected to the u input. Value could be simple like bool, int or float, string or matrix/vector. Type of value must be set by the type parameter. The the parameter codetype=13:Reference must be set for vector or matrix. There is one archive item for each column of the matrix. Data are stored only if the input RUN=on is set. The parameter subtype allow write alarm type that write other alarm blocks (for example L->H for bool alarm, HHHi for numeric alarm). the value of this parameter is in range 0 to 7 and is not used in vector/matrix items. This parameter is usually not needed.

Inputs

| u   | Value to store into archive | Unknown |
| RUN | Enable execution           | Bool    |

Parameters

| type | Type of all trend buffers | ○12 | Byte (U8) |
|      |                           |     |           |
| 1    | Bool                      |     |           |
| 2    | Byte (U8)                 |     |           |
| 3    | Short (I16)               |     |           |
| 4    | Long (I32)                |     |           |
| 5    | Word (U16)                |     |           |
| 6    | DWord (U32)               |     |           |
| 7    | Float (F32)               |     |           |
| 8    | Double (F64)              |     |           |
| 9    | Time                      |     |           |
| 10   | Large (I64)               |     |           |
| 11   | Error                     |     |           |
| 12   | String                    |     |           |
| 13   | Reference                 |     |           |

| arc | List of archives to write the events to | Word (U16) |
|     |                                          | ○1       |

| id  | Unique archive item ID | ○1 | Word (U16) |
|     |                        |     |            |

| lvl | Alarm level            | ○1 | Word (U16) |
|     |                        |     |            |

| Desc | Event description string | ○Value Description | String |
|      |                          |                 |       |

| subtype | alarm subtype (for special usage only) |                       |
|         |                                         |                     |
Output

\begin{tabular}{lc}
\textbf{iE} & Error code & Error \\
\end{tabular}
10.3 Trends recording

ACD – Archive compression using Delta criterion

Block Symbol

Function Description

The ACD block is meant for storing compressed analog signals to archives using archive events.

The main idea is to store the input signal \( u \) only when it changes significantly. The interval between two samples is in the range \( (t_{\text{min}}, t_{\text{max}}) \) seconds (rounded to the nearest multiple of the sampling period). A constant input signal is stored every \( t_{\text{max}} \) seconds while rapidly changing signal is stored every \( t_{\text{min}} \) seconds.

When the execution of the block is started, the first input value is stored. This value will be referred to as \( u_0 \) in the latter. The rules for storing the following samples are given by the delta and \( T \) input signals.

For \( T = \text{off} \) the condition \( |u - u_0| > \text{delta} \) is checked. If it holds and the last stored sample occurred more than \( t_{\text{min}} \) seconds ago, the value of input \( u \) is stored and \( u_0 = u \) is set. If the condition is fulfilled sooner than \( t_{\text{min}} \) seconds after the last stored value, the error output \( E \) is set to 1 and the first value following the \( t_{\text{min}} \) interval is stored. At that time the output \( E \) is set back to 0 and the whole procedure is repeated.

For \( T = \text{on} \) the input signal values are compared to a signal with compensated trend. The condition for storing the signal is the same as in the previous case.

The following figure shows the archiving process for both cases: a) \( T = \text{off} \), b) \( T = \text{on} \). The stored samples are marked by the symbol ×.

Inputs

- \( u \) Signal to compress and store
- \( \text{delta} \) Threshold for storing the signal

Output

- \( E \) Error output

Double (F64)
### Outputs

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>The last value stored in the archive</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>E</td>
<td>Error flag – indicates that a significant change in the input signal occurred sooner than the ( t_{\text{min}} ) interval passes</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... No error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... A error occurred</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>acls</td>
<td>Archive class determining the variable type to store</td>
<td>( \odot ) Byte (U8)</td>
</tr>
<tr>
<td></td>
<td>1 ...... Bool 5 ...... Word (U16) .....</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ...... Byte (U8) 6 ...... DWord (U32) ... Large (I64)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 ...... Short (I16) 4 ...... Float (F32)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 ...... Long (I32) 5 ...... Double (F64)</td>
<td></td>
</tr>
<tr>
<td>arc</td>
<td>List of archives to store the events. The format of the list is</td>
<td>Word (U16)</td>
</tr>
<tr>
<td></td>
<td>e.g. 1, 3 ... 5, 8. The event will be stored in all listed archives (see</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the ABC block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.</td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>Identification code of the event in the archive. This identifier must be</td>
<td>Word (U16)</td>
</tr>
<tr>
<td></td>
<td>unique in the whole target device with the REXYGEN control system (i.e. in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>all archiving blocks). Disabled for ( id = 0 ).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \odot ) 1</td>
<td></td>
</tr>
<tr>
<td>tmin</td>
<td>The shortest interval between two samples of the ( u ) input signal stored in the archive ([s])</td>
<td>Double (F64)</td>
</tr>
<tr>
<td></td>
<td>( \downarrow 0.001 \uparrow 1000000.0 \odot 1.0 )</td>
<td></td>
</tr>
<tr>
<td>tmax</td>
<td>The longest interval between two samples of the ( u ) input signal stored in the archive ([s])</td>
<td>Double (F64)</td>
</tr>
<tr>
<td></td>
<td>( \downarrow 1.0 \uparrow 1000000.0 \odot 1000.0 )</td>
<td></td>
</tr>
<tr>
<td>TR</td>
<td>Trend evaluation flag</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... The deviation of the input signal from the last stored value is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>evaluated</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... The deviation of the input signal from the last value’s trend is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>evaluated</td>
<td></td>
</tr>
<tr>
<td>Desc</td>
<td>Extended description of the event which is displayed by the diagnostic tools</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>of the REXYGEN system</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \odot ) Value Description</td>
<td></td>
</tr>
</tbody>
</table>
**TRND – Real-time trend recording**

Block Symbol  

[Diagram]

** Licence: STANDARD**

**Function Description**

The TRND block is designed for storing of up to 4 input signals (u1 to u4) in cyclic buffers in the memory of the target device. The main advantage of the TRND block is the synchronization with the real-time executive, which allows trending of even very fast signals (i.e. with very high sampling frequency). In contrary to asynchronous data storing in the higher level operator machine (host), there are no lost or multiply stored samples.

The number of stored signals is determined by the parameter \( n \). In case the trend buffer of length \( n \) samples gets full, the oldest samples are overwritten. Data can be stored once in pfac executions of the block (decimation) and the data can be further processed according to the ptype1 to ptype4 parameters. The other decimation factor afac can be used for storing data in archives.

The type of trend buffers can be specified in order to conserve memory of the target device. The memory requirements of the trend buffers are defined by the formula \( s \cdot n \cdot 1 \), where \( s \) is the size of the corresponding variable in bytes. The default type Double consumes 8 bytes per sample, thus for storing \( n = 4 \) trends of this type and length \( 1 = 1000 \), \( 8 \cdot 4 \cdot 1000 = 32000 \) bytes are required. In case the input signals come from 16-bit A/D converter the word type can be used and memory requirements drop to one quarter. Memory requirements and allowed ranges of individual types are summarized in table 1.1 on page 16 of this reference guide.

It can happen that the processed input value exceeds the representable limits when using different type of buffer than Double. In such a case the highest (lowest) representable number of the corresponding type is stored in the buffer and an error is binary encoded to the 1E output according to the following table (the unused bits are omitted):

<table>
<thead>
<tr>
<th>Input</th>
<th>Range underflow</th>
<th>Range overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit number</td>
<td>11 10 9 8</td>
<td>3 2 1 0</td>
</tr>
<tr>
<td>Bit weight</td>
<td>2048 1024 512 256</td>
<td>8 4 2 1</td>
</tr>
</tbody>
</table>

In case of simultaneous errors the resulting error code is given by the sum of the weights of individual errors. Note that underflow and overflow cannot happen simultaneously on a single input.
10.3. TRENDS RECORDING

It is possible to read, display and export the stored data by the REXYGEN Diagnostics diagnostic program.

Inputs

\( u_1 \ldots u_4 \) Analog inputs to be processed and stored in the trend
\( \text{RUN} \) Enable execution. The data are processed and stored if and only if \( \text{RUN} = \text{on} \).
\( R_1 \) Input for clearing the trend contents. The buffers are cleared when \( R_1 = \text{on} \). This flag overpowers the \( \text{RUN} \) input.

Outputs

\( y_1 \ldots y_4 \) Analog outputs of the block set once in pfac executions of the block to the last values stored in the trend buffers
\( \text{iE} \) Error code, see the table above

Parameters

\( n \) Number of signals to process and store in the trend buffers
\( l \) Number of samples reserved in memory for each trend buffer
\( \text{btype} \) Type of all \( n \) trend buffers
\( \text{ptype} \) The way the signal \( u_i, i = 1 \ldots 4 \), is processed. The last pfac samples are processed as selected and the result is stored in the \( i \)-th trend buffer.
\( \text{pfac} \) Multiple of the block execution period defining the period for storing the data in the trend buffers. Data are stored with the period of \( \text{pfac} \cdot T_S \) unless \( \text{RUN} = \text{off} \), where \( T_S \) is the block execution period in seconds.
\( \text{afac} \) Every afac-th sample stored in the trend buffer is also stored in the archives specified by the \( \text{arc} \) parameter. There are no data stored in the archives for \( \text{afac} = 0 \). Data are stored with the period of \( \text{afac} \cdot \text{pfac} \cdot T_S \), where \( T_S \) is the block execution period in seconds.
List of archives to store the trend data. The format of the list is e.g. 1,3..5,8. The data will be stored in all listed archives (see the ARC block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.

Identification code of the trend block. This identifier must be unique in the whole target device with the REXYGEN system (i.e. in all archiving blocks). Disabled for id = 0.

Title

Title of the trend to be displayed in the diagnostic tools of the REXYGEN system, e.g. in the REXYGEN Diagnostics program.

Source of timestamps. Each data sample in trend buffer is stored with a timestamp. For fast or short term trends where you are interested in sample-by-sample timing more than in absolute time, choose CORETIMER – REXYGEN internal technological time which is incremented by nominal period each base tick. For long running trends where you are interested mostly in absolute time shared with operating system (and possibly synchronized by NTP), choose RTC. Other values are intended for debug or special purposes.

1 . . . . CORETIMER – technological time – at current tick
2 . . . . CORETIMER-PRECISE – technological time – at block execution
3 . . . . RTC – real time clock (wallclock) from operating system – at current tick
4 . . . . RTC-PRECISE – real time clock (wallclock) from operating system – at block execution
5 . . . . PFC – raw high precision time (PerformanceCounter)
TRNDV – Real-time trend recording with vector input

Block Symbol

Function Description

The TRND block is designed for storing input signals which arrive at the uVec input in vector form. On the contrary to the TRND block it allows storing more than 4 signals. The signals are stored in cyclic buffers in the memory of the target device. The main advantage of the TRNDV block is the synchronization with the real-time executive, which allows trending of even very fast signals (i.e. with very high sampling frequency). In contrary to asynchronous data storing in the higher level operator machine (host), there are no samples lost or multiply stored.

The number of stored signals is determined by the parameter n. In case the trend buffer of length 1 samples gets full, the oldest samples are overwritten. Data can be stored once in pfac executions of the block (decimation). The other decimation factor afac can be used for storing data in archives.

The type of trend buffers can be specified in order to conserve memory of the target device. The memory requirements of the trend buffers are defined by the formula s·n·1, where s is the size of the corresponding variable in bytes. The default type Double consumes 8 bytes per sample, thus for storing e.g. n = 4 trends of this type and length 1 = 1000, 8·4·1000 = 32000 bytes are required. In case the input signals come from 16-bit A/D converter the Word type can be used and memory requirements drop to one quarter. Memory requirements and allowed ranges of individual types are summarized in table 1.1 on page 16 of this reference guide.

It is possible to read, display and export the stored data by the REXYGEN Diagnostics diagnostic program.

Inputs

<table>
<thead>
<tr>
<th>uVec</th>
<th>Vector signal to record</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLD</td>
<td>Input for freezing the cyclic buffers, no data is appended when HLD = on</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>iE</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>i .... REXYGEN general error</td>
</tr>
</tbody>
</table>

Licence: STANDARD
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Default Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Number of signals (trend buffers)</td>
<td>\ldots \uparrow 1 \downarrow 64 \circ 8          \uparrow 2 \downarrow 2n8435000 \circ 1000</td>
<td>Long (I32)</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>Number of samples per trend buffer</td>
<td>\ldots \downarrow 2 \uparrow 8 \circ 8 \circ 8</td>
<td>Long (I32)</td>
<td></td>
</tr>
<tr>
<td>btype</td>
<td>Type of all trend buffers</td>
<td>\circ 8</td>
<td>Long (I32)</td>
<td></td>
</tr>
<tr>
<td>pfac</td>
<td>Multiple of the block execution period defining the period for storing the data in the trend buffers. Data are stored with the period of pfac · Ts unless RUN = off, where Ts is the block execution period in seconds.</td>
<td>\downarrow 1 \uparrow 1000000 \circ 1          \uparrow 0 \downarrow 1000000</td>
<td>Long (I32)</td>
<td></td>
</tr>
<tr>
<td>afac</td>
<td>Every afac-th sample stored in the trend buffer is also stored in the archives specified by the arc parameter. There are no data stored in the archives for afac = 0. Data are stored with the period of afac · pfac · Ts, where Ts is the block execution period in seconds.</td>
<td>\uparrow 0 \downarrow 1000000</td>
<td>Long (I32)</td>
<td></td>
</tr>
<tr>
<td>arc</td>
<td>List of archives to store the trend data. The format of the list is e.g. 1, 3...5, 8. The data will be stored in all listed archives (see the ARC block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.</td>
<td>\circ 1</td>
<td>Word (U16)</td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>Identification code of the trend block. This identifier must be unique in the whole target device with the REXYGEN system (i.e. in all archiving blocks). Disabled for id = 0.</td>
<td>\circ 1</td>
<td>Word (U16)</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>Title of the trend to be displayed in the diagnostic tools of the REXYGEN system, e.g. in the REXYGEN Diagnostics program.</td>
<td>\circ Trend Title</td>
<td>String (U16)</td>
<td></td>
</tr>
</tbody>
</table>

10.4 Archive management

AFLUSH – Forced archive flushing

Block Symbol

\[\text{ licence: STANDARD} \]

Function Description

The AFLUSH block is intended for immediate storing of archive data to permanent memory (hard drive, flash disk, etc.). It is useful when power loss can be anticipated, e.g. emergency shutdown of the system following some failure. It forces the archive subsystem to write all archive data to avoid data loss. The write operation is initiated by a rising edge (off→on) at the FLUSH input regardless of the period parameter of the ARC block.
10.4. ARCHIVE MANAGEMENT

Input

**FLUSH**  Force archive flushing  **Bool**

Parameter

**arc**  List of archives to store the events. The format of the list is e.g. 1, 3..5, 8. The event will be stored in all listed archives (see the *ABC* block for details on archives numbering). Third-party programs (*Simulink, OPC clients etc.*) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.
Chapter 11

STRING – Blocks for string operations

Contents

<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNS – String constant</td>
<td>286</td>
</tr>
<tr>
<td>CONCAT – Concat string by pattern</td>
<td>287</td>
</tr>
<tr>
<td>FIND – Find a Substring</td>
<td>288</td>
</tr>
<tr>
<td>ITOS – Integer number to string conversion</td>
<td>289</td>
</tr>
<tr>
<td>LEN – String length</td>
<td>290</td>
</tr>
<tr>
<td>MID – Substring Extraction</td>
<td>291</td>
</tr>
<tr>
<td>PJR0CT – Parse JSON string (real output)</td>
<td>292</td>
</tr>
<tr>
<td>PJSOCT – Parse JSON string (string output)</td>
<td>293</td>
</tr>
<tr>
<td>REGEXP – Regular expression parser</td>
<td>294</td>
</tr>
<tr>
<td>REPLACE – Replace substring</td>
<td>297</td>
</tr>
<tr>
<td>RTOS – Real Number to String Conversion</td>
<td>298</td>
</tr>
<tr>
<td>SELSOCT – Selector switch for string signals</td>
<td>299</td>
</tr>
<tr>
<td>STUR – String to real number conversion</td>
<td>300</td>
</tr>
</tbody>
</table>
CNS – String constant

Block Symbol  Licence: STANDARD

Function Description

The CNS block is a simple string constant with maximal available size. A value of `scv` is always truncated to `nmax`.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>scv</td>
<td>String (constant) value</td>
<td>String</td>
</tr>
<tr>
<td>nmax</td>
<td>Allocated size of string [bytes]</td>
<td>Long (132)</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sy</td>
<td>String output value</td>
<td>String</td>
</tr>
</tbody>
</table>
CONCAT – Concat string by pattern

Block Symbol

Licence: STANDARD

Function Description
Concatenates up to 8 input strings su1 to su8 by pattern specified in ptrn parameter.

Inputs

| su1...8 | String input value | String |

Parameters

<table>
<thead>
<tr>
<th>ptrn</th>
<th>Concatenation pattern</th>
<th>%2%3%4</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmax</td>
<td>Allocated size of string [bytes]</td>
<td>0 → 65520</td>
<td>Long (132)</td>
</tr>
</tbody>
</table>

Output

| sy | String output value | String |


FIND – Find a Substring

Block Symbol

Licence: STANDARD

Function Description
The FIND block searches for the string su2 in the string su1 and returns a one-based index into su1 if a su2 is found or zero otherwise. Both su1 and su2 are truncated to nmax.

Inputs

su1 String input value String
su2 String input value String

Parameter

nmax Allocated size of string [bytes] \( \downarrow 0 \uparrow 65520 \) Long (I32)

Output

pos Position of substring Long (I32)
ITOS – Integer number to string conversion

Block Symbol

Function Description

The ITOS block is used for converting an integer into text. The \text{len} parameter specifies the minimum length of the output string. If the number has a smaller number of digits, zeroes or spaces will be added according to the \text{mode} parameter. The \text{radix} parameter specifies the numerical system in which the conversion is to be performed. The output string does not contain any identification of the numerical system used (e.g. the 0x prefix for the hexadecimal system).

Input

\text{n} \quad \text{Integer input of the block} \quad \text{Long (I32)}

Output

\text{sy} \quad \text{String output value} \quad \text{String}

Parameters

\text{len} \quad \text{Minimum length of output string} \quad \downarrow 0 \uparrow 30 \quad \text{Long (I32)}

\text{mode} \quad \text{Output string format} \quad \circ 1 \quad \text{Long (I32)}

1 \ldots \quad \text{Align right, fill with spaces}
2 \ldots \quad \text{Align right, fill with zeroes}
3 \ldots \quad \text{Align left, fill with spaces}

\text{radix} \quad \text{Radix} \quad \circ 10 \quad \text{Long (I32)}

2 \ldots \quad \text{Binary}
8 \ldots \quad \text{Octal}
10 \ldots \quad \text{Decimal}
16 \ldots \quad \text{Hexadecimal}
LEN – String length

Block Symbol

Function Description
The LEN block returns the actual length of the string in su in UTF-8 characters.

Input
su String input value

Parameter
nmax Allocated size of string [bytes] 

Output
len Length of input string
MID – Substring Extraction

Block Symbol

Function Description
The MID block extracts a substring sy from su. The parameters l and p specify position and length of the string being extracted in UTF-8 characters. The parameter p is one-based.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>su</td>
<td>String input value</td>
<td>String</td>
</tr>
<tr>
<td>l</td>
<td>Length of output string</td>
<td>Long (132)</td>
</tr>
<tr>
<td>p</td>
<td>Position of output string (one-based)</td>
<td>Long (132)</td>
</tr>
</tbody>
</table>

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmax</td>
<td>Allocated size of string [bytes]</td>
<td>↓0 ↑65520</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sy</td>
<td>String output value</td>
<td>String</td>
</tr>
</tbody>
</table>
PJROCT – Parse JSON string (real output)

Block Symbol

Licence: STANDARD

Function Description

 Parses input JSON string `jtxt` according to specified `name*` parameters when the input `RUN` is on. Output signals are real type.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>jtxt</td>
<td>JSON formatted string</td>
<td>String</td>
</tr>
<tr>
<td>RUN</td>
<td>Enable execution</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>name1.8</td>
<td>Name of JSON object</td>
<td>String</td>
</tr>
<tr>
<td>nmax</td>
<td>Allocated size of string [bytes]</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>yerr</td>
<td>Substitute value for an error case</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>y1.8</td>
<td>Block output signal</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>iE</td>
<td>Error code</td>
<td>Error</td>
</tr>
</tbody>
</table>
PJSOCT – Parse JSON string (string output)

Block Symbol

Licence: STANDARD

Function Description

Parses input JSON string jtxt according to specified name* parameters when the input RUN is on. Output signals are string type.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>jtxt</td>
<td>JSON formatted string</td>
<td>String</td>
</tr>
<tr>
<td>RUN</td>
<td>Enable execution</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>name1..8</td>
<td>Name of JSON object</td>
<td>String</td>
</tr>
<tr>
<td>mmx</td>
<td>Allocated size of string [bytes]</td>
<td>↓0 ↑65520 Long (I32)</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sy1..8</td>
<td>String output value</td>
<td>String</td>
</tr>
<tr>
<td>iE</td>
<td>Error code</td>
<td>Error</td>
</tr>
</tbody>
</table>
REGEXP – Regular expresion parser

Block Symbol Licence: ADVANCED

Function Description
This block implements a subset of Perl or C# or Unix command grep regular expression syntax.
Supported syntax is:
• (?i) . . . Must be at the beginning of the regex. Makes match case-insensitive
• ^ . . . Match beginning of a buffer
• $ . . . Match end of a buffer
• () . . . Grouping and substring capturing
• \s . . . Match whitespace
• \S . . . Match non-whitespace
• \d . . . Match decimal digit
• \n . . . Match new line character
• \r . . . Match line feed character
• \f . . . Match form feed character
• \v . . . Match vertical tab character
• \t . . . Match horizontal tab character
• \b . . . Match backspace character
• + . . . Match one or more times (greedy)
• +? . . . Match one or more times (non-greedy)
• * . . . Match zero or more times (greedy)
- `*?` ... Match zero or more times (non-greedy)
- `?` ... Match zero or once (non-greedy)
- `x | y` ... Match x or y (alternation operator)
- `\meta` ... Match one of the meta characters: `\$()[*]+`\`
- `\xHH` ... Match byte with hex value 0xHH, e.g. `\x4a`
- `[...]` ... Match any character from set. Ranges like `[a-z]` are supported.
- `[^...]` ... Match any character except the ones in set. Ranges like `[a-z]` are supported.

Examples

- `[0-9]*` ... Find first integer in input string (and put it into cap output)
- `[+]? [0-9]*` `\. [0-9]*` `([eE] [+] ? [0-9]+)` ... Find first real number in input string (and put it into cap output)
- `~s*(.*?)s*` ... Put trimmed input string into cap1 output
- `num\s+?s*([0-9]*` `\. [0-9]*` `)` ... Expect input string in JSON format; find integer parameter num, and put its value into cap1

Inputs

<table>
<thead>
<tr>
<th>text</th>
<th>String to parse</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>Enable execution</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>expr</th>
<th>Regular expression pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmax</td>
<td>Allocated size of string</td>
</tr>
<tr>
<td>bufmax</td>
<td>Parser internal buffer size (0 = autodetect)</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>MATCH</th>
<th>Pattern match flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>cap</td>
<td>Whole matching string</td>
</tr>
<tr>
<td>cap1</td>
<td>Captured string (string matched to 1st bracket)</td>
</tr>
<tr>
<td>cap2</td>
<td>Captured string (string matched to 2nd bracket)</td>
</tr>
<tr>
<td>cap3</td>
<td>Captured string (string matched to 3rd bracket)</td>
</tr>
<tr>
<td>cap4</td>
<td>Captured string (string matched to 4th bracket)</td>
</tr>
<tr>
<td>cap5</td>
<td>Captured string (string matched to 5th bracket)</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>cap6</td>
<td>Captured string (string matched to 6th bracket)</td>
</tr>
<tr>
<td>cap7</td>
<td>Captured string (string matched to 7th bracket)</td>
</tr>
<tr>
<td>cap8</td>
<td>Captured string (string matched to 8th bracket)</td>
</tr>
</tbody>
</table>
REPLACE – Replace substring

Block Symbol

Function Description

The REPLACE block replaces a substring from *su1* by the string *su2* and puts the result in *sy*. The parameters *l* and *p* specify position and length of the string being replaced in UTF-8 characters. The parameter *p* is one-based.

Inputs

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>su1</td>
<td>String input value</td>
<td>String</td>
</tr>
<tr>
<td>su2</td>
<td>String input value</td>
<td>String</td>
</tr>
<tr>
<td>l</td>
<td>Length of origin text</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>p</td>
<td>Position of origin text (one-based)</td>
<td>(0.00\times10^0) Long (I32)</td>
</tr>
</tbody>
</table>

Parameter

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mmax</td>
<td>Allocated size of string [bytes]</td>
<td>(\downarrow0 \uparrow65520) Long (I32)</td>
</tr>
</tbody>
</table>

Output

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sy</td>
<td>String output value</td>
<td>String</td>
</tr>
</tbody>
</table>
CHAPTER 11. STRING – BLOCKS FOR STRING OPERATIONS

RTOS – Real Number to String Conversion

Block Symbol

Function Description

The RTOS converts a real number in u into a string value in sy. Precision and format are specified by the prec and mode parameters.

Input

u    Analog input of the block

Output

sy   String output value

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>prec</td>
<td>Precision (number of digits)</td>
<td>↓0 ↑20 ○0.00E+00</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>mode</td>
<td>Output string format</td>
<td>01</td>
<td>Long (I32)</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Best fit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Normal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Exponential</td>
<td></td>
</tr>
</tbody>
</table>
SELSOCT – Selector switch for string signals

Block Symbol

Function Description

The SELSOCT block selects one of the input strings and copy it to the output string sy. The selection of the active signal u0..u15 is based on the iSW input or the binary inputs SW1..SW3. These two modes are distinguished by the BINF binary flag. The signal is selected according to the following table:

<table>
<thead>
<tr>
<th>iSW</th>
<th>SW1</th>
<th>SW2</th>
<th>SW3</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>u0</td>
</tr>
<tr>
<td>1</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>u1</td>
</tr>
<tr>
<td>2</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>u2</td>
</tr>
<tr>
<td>3</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>u3</td>
</tr>
<tr>
<td>4</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>u4</td>
</tr>
<tr>
<td>5</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>u5</td>
</tr>
<tr>
<td>6</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>u6</td>
</tr>
<tr>
<td>7</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>u7</td>
</tr>
</tbody>
</table>

Inputs

<table>
<thead>
<tr>
<th>su0..7</th>
<th>String input value</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>iSW</td>
<td>Active signal selector</td>
<td>Long (132)</td>
</tr>
<tr>
<td>SW1..3</td>
<td>Binary signal selector</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>BINF</th>
<th>Enable the binary selectors</th>
<th>Bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmax</td>
<td>Allocated size of string [bytes]</td>
<td>↓0 ↑65520 Long (132)</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>sy</th>
<th>The selected input signal</th>
<th>String</th>
</tr>
</thead>
</table>
STOR – String to real number conversion

Block Symbol

Function Description
The STOR converts a string in su into a real number in y. An error is signaled in E if unsuccessful.

Input
su String input value

Parameter
yerr Substitute value for an error case

Outputs
y Analog output of the block
E Error indicator

Licence: STANDARD
Chapter 12

PARAM – Blocks for parameter handling

Contents

GETPA – Block for remote array parameter acquirement .... 302
GETPR, GETPI, GETPB – Blocks for remote parameter acquirement 304
GETPS – * Block for remote string parameter acquirement .... 306
PARA – Block with input-defined array parameter ............... 307
PAPE – Block with input-defined enumeration parameter .... 308
PARR, PARI, PARB – Blocks with input-defined parameter .... 309
PARS – * Block with input-defined string parameter .......... 311
SETPA – Block for remote array parameter setting ............ 312
SETPR, SETPI, SETPB – Blocks for remote parameter setting .... 314
SETPS – * Block for remote string parameter setting ........... 316
SGSLP – Set, get, save and load parameters .................. 317
SILO – Save input value, load output value .................... 321
SILOS – Save input string, load output string ............... 323
GETPA – Block for remote array parameter acquisition

Function Description

The GETPA block is used for acquiring the array parameters of other blocks in the model remotely. The block operates in two modes, which are switched by the GETF parameter. For GETF = off the output arrRef is set to the value of the remote parameter at the start and every time when the remote parameter changes. If the GETF parameter is set to on, then the block works in single-shot read mode. In that case the remote parameter is read only when rising edge (off → on) occurs at the GET input.

The name of the remote parameter is determined by the string parameter sc (string connection), which has the form <block_path:parameter_name>. The path to the block whose parameter should be read can contain hierarchic levels separated by dots followed by the block name. The path can be either relative or absolute:

- Relative – starts at the level where the GETPA block is located. The string has to be prefixed with '.' in this case. Examples of relative paths: ".:CNDR:yp", ".Lights.ATMT:touts".
- Absolute – complete sequence of hierarchic levels down to the block. For referring to blocks located in the driver task (see the IOTASK block for details on configuration) the '"' followed by the driver’s name is used at the beginning of the absolute path. Examples of absolute paths: ":task1.inputs.ATMT:touts", ":EfaDrv.measurements.CNDR:yp".

The order and names of individual hierarchic levels are displayed in a tree structure in the REXYGEN Diagnostics program.

Input

GET Input for initiating one-shot parameter read Bool

Outputs

arrRef Array reference Reference
E Error flag Bool
### Parameters

<table>
<thead>
<tr>
<th>sc</th>
<th>String connection to the parameter</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETF</td>
<td>Get parameter only when forced to</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... Remote parameter is continuously read</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... One-shot mode, the remote parameter is read only</td>
<td></td>
</tr>
<tr>
<td></td>
<td>when forced to by the GET input (rising edge)</td>
<td></td>
</tr>
<tr>
<td>nmax</td>
<td>Maximum size of array</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Long (132)</td>
</tr>
</tbody>
</table>
CHAPTER 12. PARAM - BLOCKS FOR PARAMETER HANDLING

GETPR, GETPI, GETPB - Blocks for remote parameter acquirement

Block Symbols

Function Description

The GETPR, GETPI and GETPB blocks are used for acquiring the parameters of other blocks in the model remotely. The only difference among the three blocks is the type of parameter which they are acquiring. The GETPR block is used for obtaining real parameters, the GETPI block for integer parameters and the GETPB block for Boolean parameters.

The blocks operate in two modes, which are switched by the GETF parameter. For GETF = off the output \( y \) (or \( k \), \( Y \)) is set to the value of the remote parameter at the start and every time when the remote parameter changes. If the GETF parameter is set to on, then the blocks work in single-shot read mode. In that case the remote parameter is read only when rising edge \( \text{off} \to \text{on} \) occurs at the GET input.

The name of the remote parameter is determined by the string parameter sc (string connection), which has the form <block_path:parameter_name>. It is also possible to access individual items of array-type parameters (e.g. the tout parameter of the ATMT block). This can be achieved using the square brackets and item number, e.g. .ATMT:touts[2]. The items are numbered from zero, thus the string connection stated above refers to the third element of the array.

The path to the block whose parameter should be read can contain hierarchic levels separated by dots followed by the block name. The path can be either relative or absolute:

- **Relative** - starts at the level where the GETPR block (or GETPI, GETPB) is located. The string has to be prefixed with '.' in this case. Examples of relative paths:
  
  - ".GAIN:x", ".Motor1.Position:ycn"

- **Absolute** - complete sequence of hierarchic levels down to the block. For referring to blocks located in the driver task (see the IDTASK block for details on configuration) the '*' followed by the driver’s name is used at the beginning of the absolute path. Examples of absolute paths: "task1.inputs.lin1:u2", ".&EfaDrv.measurements.DER1:n"

The order and names of individual hierarchic levels are displayed in a tree structure in the REXYGEN Diagnostics program.

Input

\( \text{GET} \)  Input for initiating one-shot parameter read \( \text{off} \to \text{on} \)  Bool
### Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Parameter value, output of the GETPR block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>k</td>
<td>Parameter value, output of the GETPI block</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>Y</td>
<td>Parameter value, output of the GETPB block</td>
<td>Bool</td>
</tr>
<tr>
<td>E</td>
<td>Error flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>off ... No error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... An error occurred</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc</td>
<td>String connection to the remote parameter respecting the above mentioned notation</td>
<td>String</td>
</tr>
<tr>
<td>GETF</td>
<td>Continuous or one-shot mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td>off ... Remote parameter is continuously read</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... One-shot mode, the remote parameter is read only when forced to by the GET input (rising edge)</td>
<td>Bool</td>
</tr>
</tbody>
</table>
GETPS - Block for remote string parameter acquisition

Block Symbol

License: STANDARD

Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Input

GET Input for initiating one-shot parameter read Bool

Parameters

sc String connection to the parameter String

GETF Get parameter only when forced to Bool

off ... Remote parameter is continuously read
on .... One-shot mode

mmax Allocated size of string Long (I32)

Outputs

sy Parameter value String

E Error indicator Bool

off ... No error
on .... An error occurred
**PARA – Block with input-defined array parameter**

**Block Symbol**

Licence: STANDARD

**Function Description**

The PARA block allows, additionally to the standard way of parameter setting, changing one of its parameters by the input signal. The input-parameter pair is uRef and apar.

The Boolean input LOC (LOCal) determines whether the value of the apar parameter is read from the input uRef or is input-independent (LOC = on). In the local mode LOC = on the parameter apar contains the last value of input uRef entering the block right before LOC was set to on.

The output value is equivalent to the value of the parameter (yRef = apar).

**Inputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uRef</td>
<td>Array reference</td>
<td>Reference</td>
</tr>
<tr>
<td>LOC</td>
<td>Activation of local mode</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... The parameter follows the input</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... Local mode active</td>
<td></td>
</tr>
</tbody>
</table>

**Output**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yRef</td>
<td>Array reference</td>
<td>Reference</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETS</td>
<td>Set array size flag. Use this flag to adjust the size of array when setting the parameter.</td>
<td>Bool</td>
</tr>
<tr>
<td>apar</td>
<td>Internal value of the parameter</td>
<td>Double (F64)</td>
</tr>
<tr>
<td></td>
<td>⊙[0.0 1.0 2.0 3.0 4.0 5.0]</td>
<td></td>
</tr>
</tbody>
</table>
**PARE – Block with input-defined enumeration parameter**

Block Symbol

Licence: STANDARD

**Function Description**

The block is similar to the the PARI block with the additional option to assign texts to numeric values. The corresponding text is set on the output sy. The block has two modes and the active mode is selected by the LIST parameter. If LIST=off a corresponding text for the input value is set on the output sy. If LIST=on the input number is considered as a bitfield, texts are defined for each bit and the output sy is composed of the texts that correspond to bits which are set. The behavior for undefined values is determined by the SATF parameter. If SATF=off, undefined values are set to output iy and the output sy is set to empty text. Undefined values are ignored if SAT=on. The pupstr parameter has the same format as in the CNA block: <number1>: <description1>|<number2>: <description2>|<number3>: <description3> ...

**Inputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip</td>
<td>Parameter value</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>LOC</td>
<td>Activation of local mode</td>
<td>Bool</td>
</tr>
</tbody>
</table>

   - off ... The parameter follows the input
   - on ... Local mode active

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipar</td>
<td>Internal value of parameter</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>pupstr</td>
<td>Popup list definition</td>
<td>String</td>
</tr>
</tbody>
</table>

   - option A: option B: option C

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM</td>
<td>Number in string output</td>
<td>Bool</td>
</tr>
<tr>
<td>LIST</td>
<td>Bitfield mode</td>
<td>Bool</td>
</tr>
<tr>
<td>SATF</td>
<td>Saturation flag</td>
<td>Bool</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>iy</td>
<td>Integer output of the block</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>sy</td>
<td>String output value</td>
<td>String</td>
</tr>
</tbody>
</table>
PARR, PARI, PARB – Blocks with input-defined parameter

Block Symbols

```
PARR
LOC y
PARI
LOC k
PARB
LOC Y
```

Function Description

The PARR, PARI and PARB blocks allow, additionally to the standard way of parameters setting, changing one of their parameters by the input signal. The input-parameter pairs are p and par for the PARR block, ip and ipar for the PARI block and finally P and PAR for the PARB block.

The Boolean input LOC (LOCal) determines whether the value of the par (or ipar, PAR) parameter is read from the input p (or ip, P) or is input-independent (LOC = on). In the local mode LOC = on the parameter par (or ipar, PAR) contains the last value of input p (or ip, P) entering the block right before LOC was set to on. Afterwards it is possible to modify the value manually.

The output value is equivalent to the value of the parameter y = par (or k = ipar, Y = PAR). The output of the PARR and PARI blocks can be additionally constrained by the saturation limits (lolim, hilim). The saturation is active only when SATF = on.

See also the SHLD block, which can be used for storing a numeric value, similarly as in the PARR block.

Inputs

- **p** Parameter value (the PARR block) [Double (F64)]
- **ip** Parameter value (the PARI block) [Long (I32)]
- **P** Parameter value (the PARB block) [Bool]
- **LOC** Activation of local mode [Bool]
  - **off** ... The parameter follows the input
  - **on** ... Local mode active

Output

- **y** Logical output of the PARR block [Double (F64)]
- **k** Logical output of the PARI block [Long (I32)]
- **Y** Logical output of the PARB block [Bool]

Parameter

- **par** Initial value of the parameter (the PARR block) [1.0 Double (F64)]
- **ipar** Initial value of the parameter (the PARI block) [1 Long (I32)]
- **PAR** Initial value of the parameter (the PARB block) [on Bool]
SATF Activation of the saturation limits for the PARR and PARI blocks  Bool
   off .... Signal not limited
   on .... Saturation limits active
hilim Upper limit of the output signal (the PARR and PARI blocks)  Double (F64)
   ⊠ 1.0
lolim Lower limit of the output signal (the PARR and PARI blocks)  Double (F64)
   ⊠ -1.0
**PARS — * Block with input-defined string parameter**

**Block Symbol**

License: **STANDARD**

**Function Description**

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

**Inputs**

- **sp**   Parameter value                   String
- **LOC** Activation of local mode         Bool

**Parameters**

- **spar** Internal value of the parameter String
- **nmax** Allocated size of string         Long (I32)

**Output**

- **sy**   String output of the block       String
SETPA – Block for remote array parameter setting

Block Symbol

Licence: STANDARD

Function Description

The SETPA block is used for setting the array parameters of other blocks in the model remotely. The block operates in two modes, which are switched by the SETF parameter. For SETF = off the remote parameter cs is set to the value of the input vector signal arrRef at the start and every time when the input signal changes. If the SETF parameter is set to on, then the block works in one-shot write mode. In that case the remote parameter is set only when rising edge (off → on) occurs at the SET input.

The name of the remote parameter is determined by the string parameter sc (string connection), which has the form <block_path:parameter_name>. The path to the block whose parameter should be read can contain hierarchic levels separated by dots followed by the block name. The path can be either relative or absolute:

- Relative – starts at the level where the SETPA block is located. The string has to be prefixed with ‘.’ in this case. Examples of relative paths: ".CNDR:yp", ".Lights.ATMT:touts".
- Absolute – complete sequence of hierarchic levels down to the block. For referring to blocks located in the driver task (see the IOTASK block for details on configuration) the ‘&’ followed by the driver’s name is used at the beginning of the absolute path. Examples of absolute paths: ":task1.inputs.ATMT:touts", ":&EfaDrv.measurements.CNDR:yp".

The order and names of individual hierarchic levels are displayed in a tree structure in the REXYGEN Diagnostics program.

Inputs

arrRef  Array reference  Reference
SET     Input for initiating one-shot parameter write  Bool

Output

E       Error flag  Bool
### Parameters

| sc | String connection to the parameter | String |
| SETF | Continuous or one-shot mode | Bool |
| | off ... Remote parameter is continuously updated | |
| | on .... One-shot mode, the remote parameter is updated only when forced to by the SET input (rising edge) | |
| SETS | Set array size flag. Use this flag to adjust the size of array when setting the parameter. | Bool |
SETPR, SETPI, SETPB – Blocks for remote parameter setting

Block Symbols

Function Description

The SETPR, SETPI, and SETPB blocks are used for setting the parameters of other blocks in the model remotely. The only difference among the three blocks is the type of parameter which they are setting. The SETPR block is used for setting real parameters, the SETPI block for integer parameters and the SETPB block for Boolean parameters.

The blocks operate in two modes, which are switched by the SETF parameter. For SETF = off the remote parameter sc is set to the value of the input signal p (or ip, P) at the start and every time when the input changes. If the SETF parameter is set to on, then the blocks work in one-shot write mode. In that case the remote parameter is set only when rising edge (off → on) occurs at the SET input. Successful modification of the remote parameter is indicated by zero error output E = off and the output y (or k, Y) is set to the value of the modified parameter. The error output is set to E = on in case of write error.

The name of the remote parameter is determined by the string parameter sc (string connection), which has the form <block_path:parameter_name>. It is also possible to access individual items of array-type parameters (e.g. the tout parameter of the ATMT block). This can be achieved using the square brackets and item number, e.g. .ATMT:touts[2]. The items are numbered from zero, thus the string connection stated above refers to the third element of the array.

The path to the block whose parameter should be set can contain hierarchic levels separated by dots followed by the block name. The path can be either relative or absolute:

- Relative – starts at the level where the SETPR block (or SETPI, SETPB) is located. The string has to be prefixed with "." in this case. Examples of relative paths: ".GAIN:k", ".Motor1.Position:ycn".

- Absolute – complete sequence of hierarchic levels down to the block. For referring to blocks located in the driver task (see the IOBLOCK block for details on configuration) the "&" followed by the driver’s name is used at the beginning of the absolute path. Examples of absolute paths: ".task1.inputs.lin1:u2", ".&EfaDrv.measurements.DER1:n".

The order and names of individual hierarchic levels are displayed in a tree structure in the REXYGEN Diagnostics program.
**Inputs**

<table>
<thead>
<tr>
<th></th>
<th>Desired parameter value at the SETPR block input</th>
<th>Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Desired parameter value at the SETPI block input</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>ip</td>
<td>Desired parameter value at the SETPB block input</td>
<td>Bool</td>
</tr>
<tr>
<td>P</td>
<td>Input for initiating one-shot parameter write (off→on)</td>
<td>Bool</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th></th>
<th>Parameter value (the SETPR block)</th>
<th>Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Parameter value (the SETPI block)</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>k</td>
<td>Parameter value (the SETPB block)</td>
<td>Bool</td>
</tr>
<tr>
<td>Y</td>
<td>Error flag</td>
<td>Bool</td>
</tr>
<tr>
<td>E</td>
<td>off ... No error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... An error occurred</td>
<td></td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>String connection to the remote parameter respecting the above mentioned notation</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc</td>
<td>Continuous or one-shot mode</td>
<td>Bool</td>
</tr>
<tr>
<td>SETF</td>
<td>off ... Remote parameter is continuously updated</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... One-shot mode, the remote parameter is updated only when forced to by the SET input (rising edge)</td>
<td></td>
</tr>
</tbody>
</table>
SETPS – * Block for remote string parameter setting

Block Symbol

Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp</td>
<td>Desired parameter value</td>
<td>String</td>
</tr>
<tr>
<td>SET</td>
<td>Input for initiating one-shot parameter write</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc</td>
<td>String connection to the parameter</td>
<td>String</td>
</tr>
<tr>
<td>SETF</td>
<td>Set parameter only when forced to</td>
<td>Bool</td>
</tr>
<tr>
<td>nmax</td>
<td>Allocated size of string</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sy</td>
<td>Parameter value</td>
<td>String</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
SGSLP – Set, get, save and load parameters

Block Symbol

Function Description

The SGSLP block is a special function block for manipulation with parameters of other function blocks in the REXYGEN system configuration. It works also in the Matlab-Simulink system but its scope is limited to the .mdl file it is included in.

The block can manage up to 16 parameter sets, which are numbered from 0 to 15. The number of parameter sets is given by the nps parameter and the active set is defined by the ips input. If the ips input remains unconnected, the active parameter set is ips = 0. Each set contains up to 16 different parameters defined by the string parameters sc0 to sc15. Thus the SGSLP block can work with a maximum of 256 parameters of the REXYGEN system. An empty sci string means that no parameter is specified, otherwise one of the following syntaxes is used:

1. \texttt{<block>:<param>} – Specifies one function block named \texttt{block} and its parameter \texttt{param}. The same block and parameter are used for all nps parameter sets in this case.

2. \texttt{<block>:<param><sep><block>:<param>} – This syntax allows the parameters to differ among the parameter sets. In general, each sci string can contain up to 16 items in the form \texttt{<block>:<param>} separated by comma or semi-colon. E.g. the third item of these is active for ips = 2. There should be exactly nps items in each non-empty sci string. If there is less items than nps none of the below described operations can be executed on the incomplete parameter set.

It is recommended not to use both syntaxes in one SGSLP block, all 16 sci strings should have the same form. The first syntax is for example used when producing nps types of goods, where many parameters must be changed for each type of production. The second syntax is usually used for saving user-defined parameters to disk (see the
SAVE operation below). In that case it is desirable to arrange automated switching of the
ips input (e.g. using the AMT block from the LOGIC library).

The broot parameter is suitable when all blocks whose parameters are to be con-
trolled by the SGSLP block reside in the same subsystem or deeper in the hierarchy. It
is inserted in front of each <block> substring in the sci parameters. The "." character
stands for the subsystem where the SGSLP block is located. No quotation marks are
used to define the parameter, they are used here solely to highlight a single character.
If the broot parameter is an empty string, all <block> items must contain full path.
For example, to create a connection to the CNR block and its parameter ycn located in
the same subsystem as the SGSLP block, broot = . and sc0 = CNR:ycn must be set. Or
it is possible to leave the broot parameter empty and put the "." character to the sc0
string. See the GETPR or SETPR blocks description for more details about full paths in the
REXYGEN system.

The SGSLP block executes one of the below described operations when a rising edge
(off→on) occurs at the input of the same name. The operations are:

SET – Sets the parameters of the corresponding parameter set ips to the values of the
input signals ui. In case the parameter is successfully set, the same value is also
sent to the yi output.

GET – Gets the parameters of the corresponding parameter set ips. In case the parameter
is successfully read, its value is sent to the yi output.

SAVE – Saves the parameters of the corresponding parameter set ips to a file on the
target platform. The parameters of the procedure and the format of the resulting
file are described below.

LOAD – Loads the parameters of the corresponding parameter set ips from a file on
the target platform. This operation is executed also during the initialization of the
block but only when 0 ≤ ips0 ≤ ips – 1. The parameters of the procedure and
the format of the file are described below.

The LOAD and SAVE operations work with a file on the target platform. The name of
the file is given by the fname parameter and the following rules:

• If no extension is specified in the fname parameter, the .rxs (ReX Status file)
extension is added.

• A backup file is created when overwriting the file. The file name is preserved, only
the extension is modified by adding the "" character right after the "." (e.g. when
no extension is specified, the backup file has a . rxs extension.

• The path is relative to the folder where the archives of the REXYGEN system are
stored. The file should be located on a media which is not erased by system restart
(flash drive or hard drive, not RAM).
The **SAVE** operation stores the data in a text file. Two lines are added for each parameter **sci**, \( i = 0, \ldots, m \), where \( m < 16 \) defines the nonempty **scm** string with the highest number. The lines have the form:

"<block>:<param>",...,"<block>:<param>"

<value>,...,<value>

There are \( nps \) individual items "<block>:<param>" which are separated by commas. The second line contains the same number of <value> items which contain the value of the parameter at the same position in the line above. Note that the format of the file remains the same even for **sci** containing only one <block>:<param> item (see the syntax no. 1 above). The "<block>:<param>" item is always listed \( nps \)-times in the file, which allows seamless switching of the **sci** parameters syntax without modifying the file.

Consider using the **SIL6** block if working with only a few values.

### Inputs

- **ui**: \( i \)-th analog input signal, \( i = 0, \ldots, 15 \)  
  - Double (F64)
- **ips**: Parameter set index (numbered from zero)  
  - Long (I32)
- **SET**: Set the parameters of the **ips** parameter set according to the values of the **ui** inputs. The values can be found at the **yi** outputs after a successful operation.
- **GET**: Get the parameters of the **ips** parameter set. The values can be found at the **yi** outputs after a successful operation.
- **SAVE**: Save the **ips** parameter set to a file on the target device  
  - Bool
- **LOAD**: Load the **ips** parameter set from a file on the target device  
  - Bool

### Outputs

- **yi**: \( i \)-th analog output signal, \( i = 0, \ldots, 15 \)  
  - Double (F64)
- **E**: Error flag
  - Off ... No error
  - On .... An error occurred (see **iE**)
  - Bool
CHAPTER 12. PARAM – BLOCKS FOR PARAMETER HANDLING

<table>
<thead>
<tr>
<th>iE</th>
<th>Error or warning code of the last operation</th>
<th>Long (I32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Operation successful</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Fatal error of the Matlab system (only in Simulink), the block is no longer executed</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Error opening the file for reading (LOAD operation)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Error opening the file for writing (SAVE operation)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Incorrect file format</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>The ips parameter set not found in the file</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Parameter not found in the configuration, name mismatch (LOAD operation)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Unexpected end of file</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Error writing to file (disk full?)</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Parameter syntax error (the ‘:’ character not found)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Only whitespace in the parameter name</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Error creating the backup file</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Error obtaining the parameter value by the GET operation (non-existing parameter?)</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Error setting the parameter value by the SET operation (non-existing parameter?)</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Timeout during obtaining/settting the parameter</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>The specified parameter is read-only</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>The ips parameter is out of range</td>
<td></td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>nps</th>
<th>Number of parameter sets</th>
<th>Long (I32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ips0</td>
<td>Index of parameter set to load and set during the block initialization. No set is read for ips0 &lt; 0 or ips0 &gt;= nps</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>iprec</td>
<td>Precision (number of digits) for storing the values of double type in a file</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>icolw</td>
<td>Requested column width in the status file. Spaces are appended to the parameter value when necessary.</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>fname</td>
<td>Name of the file the SAVE and LOAD operations work with</td>
<td>String</td>
</tr>
<tr>
<td>status</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>broot</td>
<td>Root block in hierarchy, inserted at the beginning of all sci parameters, see the description above</td>
<td>String</td>
</tr>
<tr>
<td>sci</td>
<td>Strings defining the connection of ui inputs and yi outputs to the parameters, i = 0, ..., 15, see details above</td>
<td>String</td>
</tr>
</tbody>
</table>
SIL0 – Save input value, load output value

Block Symbol

Licence: STANDARD

Function Description

The SIL0 block can be used to export or import a single value to/from a file. The value is saved when a rising edge (off→on) occurs at the SAVE input and the value is also set to the y output. The value is loaded at startup and when a rising edge (off→on) occurs at the LOAD input.

The outputs E and lastErr indicate an error during disk operation. The E indicator is reset on falling edge at the SAVE or LOAD input while the lastErr output holds the value until another disk operation is invoked. If the error occurs during the LOAD operation, a substitute value yerr is set to the y output.

Alternatively it is possible to write or read the value continuously if the corresponding flag (CSF, CLF) is set to on. The disk operation is then performed when the corresponding input is set to on. Beware, in that case the disk operation is executed in each cycle, which can cause excessive use of the storage medium. Thus it is necessary to use this feature with caution.

The fname parameter defines the location of the file on the target platform. The path is relative to the data folder of the RexCore runtime module.

Use the SGSLP function block for advanced and complex operations.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>Input signal</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>SAVE</td>
<td>Save value to file</td>
<td>Bool</td>
</tr>
<tr>
<td>LOAD</td>
<td>Load value from file</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>fname</td>
<td>Name of persistent storage file</td>
<td>String</td>
</tr>
<tr>
<td>CSF</td>
<td>Flag for continuous saving</td>
<td>Bool</td>
</tr>
<tr>
<td>CLF</td>
<td>Flag for continuous loading</td>
<td>Bool</td>
</tr>
<tr>
<td>yerr</td>
<td>Substitute value for an error case</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Output signal</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>E</td>
<td>Error flag</td>
<td>Bool</td>
</tr>
</tbody>
</table>
lastErr  Result of last operation  Long (I32)
SILOs – Save input string, load output string

Block Symbol

Licence: STANDARD

Function Description

The SILOs block can be used to export or import a string to/from a file. The string is saved when a rising edge (off→on) occurs at the SAVE input and the string is also set to the sy output. The string is loaded at startup and when a rising edge (off→on) occurs at the LOAD input.

If a logical true (on) is brought to the APPEND input, the input string is added to the end of the file when it is saved. This mode is useful for logging events into text files. This input signal has no effect on loading from the file.

The LL0 parameter is intended for choosing whether to load the entire file (off) or its last line only (on).

The outputs E and lastErr indicate an error during disk operation. The E indicator is reset on falling edge at the SAVE or LOAD input while the lastErr output holds the value until another disk operation is invoked.

Alternatively it is possible to write or read the string continuously if the corresponding flag (CSF, CLF) is set to on. The disk operation is then performed when the corresponding input is set to on. Beware, in that case the disk operation is executed in each cycle, which can cause excessive use of the storage medium. Thus it is necessary to use this feature with caution.

The fname parameter defines the location of the file on the target platform. The path is relative to the data folder of the RexCore runtime module.

Inputs

<table>
<thead>
<tr>
<th>sy</th>
<th>String input of the block</th>
<th>○0</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAVE</td>
<td>Save string to file</td>
<td></td>
<td>Bool</td>
</tr>
<tr>
<td>LOAD</td>
<td>Load string from file</td>
<td></td>
<td>Bool</td>
</tr>
<tr>
<td>APPEND</td>
<td>Append saved string to file</td>
<td></td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>sy</th>
<th>String output of the block</th>
<th></th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Error indicator</td>
<td></td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>off ... No error</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... An error occurred</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lastErr</td>
<td>Result of last operation</td>
<td></td>
<td>Long (132)</td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fname</code></td>
<td>Name of persistent storage file</td>
<td>String</td>
</tr>
<tr>
<td><code>CSF</code></td>
<td>Continuous saving</td>
<td>Bool</td>
</tr>
<tr>
<td><code>CLF</code></td>
<td>Continuous loading</td>
<td>Bool</td>
</tr>
<tr>
<td><code>LL0</code></td>
<td>Last line only loading</td>
<td>Bool</td>
</tr>
<tr>
<td><code>nmax</code></td>
<td>Allocated size of string</td>
<td>$0 \leq \text{65520}$ Long (I32)</td>
</tr>
</tbody>
</table>
Chapter 13

MODEL – Dynamic systems simulation

Contents

CDELSSM – Continuous state space model of a linear system with time delay ............................................ 326
CSSM – Continuous state space model of a linear system ........ 329
DDELSSM – Discrete state space model of a linear system with time delay ............................................. 331
DSSM – Discrete state space model of a linear system ........ 333
FOPDT – First order plus dead-time model .................. 335
MDL – Process model ........................................... 336
MDLI – Process model with input-defined parameters ........ 337
MVD – Motorized valve drive .................................. 338
SOPDT – Second order plus dead-time model ............... 339

325
CDELSSM – Continuous state space model of a linear system with time delay

Block Symbol

Function Description

The CDELSSM block (Continuous State Space Model with time DELay) simulates behavior of a linear system with time delay \( \text{del} \)

\[
\frac{dx(t)}{dt} = A_c x(t) + B_c u(t - \text{del}), \quad x(0) = x_0
\]
\[
y(t) = C_c x(t) + D_c u(t),
\]

where \( x(t) \in \mathbb{R}^n \) is the state vector, \( x_0 \in \mathbb{R}^n \) is the initial value of the state vector, \( u(t) \in \mathbb{R}^m \) is the input vector, \( y(t) \in \mathbb{R}^p \) is the output vector. The matrix \( A_c \in \mathbb{R}^{n \times n} \) is the system dynamics matrix, \( B_c \in \mathbb{R}^{n \times m} \) is the input matrix, \( C_c \in \mathbb{R}^{p \times n} \) is the output matrix and \( D_c \in \mathbb{R}^{p \times m} \) is the direct transmission (feedthrough) matrix.

All matrices are specified in the same format as in Matlab, i.e. the whole matrix is placed in brackets, elements are entered by rows, elements of a row are separated by spaces (blanks), rows are separated by semicolons. The \( x_0 \) vector is a column, therefore the elements are separated by semicolons (each element is in a separate row).

The simulated system is first converted to the discrete (discretized) state space model

\[
x((k+1)T) = A_dx(kT) + B_{d1}u((k-d)T) + B_{d2}u((k-d+1)T), \quad x(0) = x_0
\]
\[
y(kT) = C_c x(kT) + D_c u(kT),
\]

where \( k \in \{1, 2, \ldots\} \) is the simulation step, \( T \) is the execution period of the block in seconds and \( d \) is a delay in simulation step such that \( (d-1)T < \text{del} \leq dT \). The period \( T \) is not entered in the block, it is determined automatically as a period of the task (\text{TASK}, \text{QTASK} nebo \text{IOTASK}) containing the block.

If the input \( u(t) \) is changed only in the moments of sampling and between two consecutive sampling instants is constant, i.e. \( u(t) = u(kT) \) for \( t \in [kT, (k+1)T) \), then the
matrices $A_d$, $B_{d1}$ and $B_{d2}$ are determined by

\[
A_d = e^{A_c T} \\
B_{d1} = e^{A_c (T-\Delta)} \int_0^\Delta e^{A_c \tau} B_c d\tau \\
B_{d2} = \int_{T-\Delta}^T e^{A_c \tau} B_c d\tau,
\]

where $\Delta = del - (d - 1)T$.

Computation of discrete matrices $A_d$, $B_{d1}$ and $B_{d2}$ is based on a method described in [3], which uses Padé approximations of matrix exponential and its integral and scaling technique.

During the real-time simulation, single simulation step of the above discrete state space model is computed in each execution time instant.

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
</tr>
<tr>
<td>$\text{HLD}$</td>
</tr>
<tr>
<td>$u\ldots u_{16}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$iE$</td>
</tr>
<tr>
<td>0 \ldots O.K., the simulation runs correctly</td>
</tr>
<tr>
<td>$y_1\ldots y_{16}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{UD}$</td>
</tr>
<tr>
<td>$del$</td>
</tr>
<tr>
<td>$\text{is}$</td>
</tr>
<tr>
<td>$\text{eps}$</td>
</tr>
<tr>
<td>Symbol</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>Ac</td>
</tr>
<tr>
<td>Bc</td>
</tr>
<tr>
<td>Cc</td>
</tr>
<tr>
<td>Dc</td>
</tr>
<tr>
<td>x0</td>
</tr>
</tbody>
</table>
CSSM – Continuous state space model of a linear system

Block Symbol

Licence: ADVANCED

Function Description

The CSSM block (Continuous State Space Model) simulates behavior of a linear system

\[
\begin{align*}
\frac{dx(t)}{dt} &= A_c x(t) + B_c u(t), \quad x(0) = x_0 \\
y(t) &= C_c x(t) + D_c u(t),
\end{align*}
\]

where \( x(t) \in \mathbb{R}^n \) is the state vector, \( x_0 \in \mathbb{R}^n \) is the initial value of the state vector, \( u(t) \in \mathbb{R}^m \) is the input vector, \( y(t) \in \mathbb{R}^p \) is the output vector. The matrix \( A_c \in \mathbb{R}^{n \times n} \) is the system dynamics matrix, \( B_c \in \mathbb{R}^{n \times m} \) is the input matrix, \( C_c \in \mathbb{R}^{p \times n} \) is the output matrix and \( D_c \in \mathbb{R}^{p \times m} \) is the direct transmission (feedthrough) matrix.

All matrices are specified in the same format as in Matlab, i.e. the whole matrix is placed in brackets, elements are entered by rows, elements of a row are separated by spaces (blanks), rows are separated by semicolons. The \( x_0 \) vector is a column, therefore the elements are separated by semicolons (each element is in a separate row).

The simulated system is first converted to the discrete (discretized) state space model

\[
\begin{align*}
x((k+1)T) &= A_d x(kT) + B_d u(kT), \quad x(0) = x_0 \\
y(kT) &= C_d x(kT) + D_d u(kT),
\end{align*}
\]

where \( k \in \{1, 2, \ldots\} \) is the simulation step, \( T \) is the execution period of the block in seconds. The period \( T \) is not entered in the block, it is determined automatically as a period of the task (\textsc{task}, \textsc{qtask} nebo \textsc{iotask}) containing the block.

If the input \( u(t) \) is changed only in the moments of sampling and between two consecutive sampling instants is constant, i.e. \( u(t) = u(kT) \) for \( t \in [kT, (k+1)T) \), then the matrices \( A_d \) and \( B_d \) are determined by

\[
\begin{align*}
A_d &= e^{A_c T} \\
B_d &= \int_0^T e^{A_c \tau} B_c d\tau
\end{align*}
\]
Computation of discrete matrices $A_d$ and $B_d$ is based on a method described in [5], which uses Padé approximations of matrix exponential and its integral and scaling technique.

During the real-time simulation, single simulation step of the above discrete state space model is computed in each execution time instant.

**Inputs**

| R1         | Reset signal. When R1 = on, the state vector $x$ is set to its initial value $x_0$. The simulation continues on the falling edge of R1 (on$ightarrow$off). | Bool         |
|------------|----------------------------------------------------------------------------------|--------------|
| HLD        | Simulation output holds its value if HLD=on.                                    | Bool         |
| u1..u16    | Simulated system inputs. First $m$ simulation inputs are used where $m$ is the number of columns of the matrix Bc. | Double (F64) |

**Outputs**

<table>
<thead>
<tr>
<th>iE</th>
<th>Block error code</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>O.K., the simulation runs correctly</td>
<td></td>
</tr>
<tr>
<td>-213</td>
<td>incompatibility of the state space model matrices dimensions</td>
<td></td>
</tr>
<tr>
<td>-510</td>
<td>the model is badly conditioned (some working matrix is singular or nearly singular)</td>
<td></td>
</tr>
<tr>
<td>xxx</td>
<td>error code xxx of REXGEN, see appendix C for details</td>
<td></td>
</tr>
<tr>
<td>y1..y16</td>
<td>Simulated system outputs. First $p$ simulation outputs are used where $p$ is the number of rows of the matrix Cc.</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

**Parameters**

| UD         | Matrix Dc usage flag. If UD=off then the Dc matrix is not used for simulation (simulations behaves as if the Dc matrix is zero). | Bool         |
| is         | Order of the Padé approximation of the matrix exponential for the computation of the discretized system matrices. | Long (I32)   |
| eps        | Required accuracy of the Padé approximation.                                    | Double (F64) |
| Ac         | Matrix $(n \times n)$ of the continuous linear system dynamics.                  | Double (F64) |
| Bc         | Input matrix $(n \times m)$ of the continuous linear system.                    | Double (F64) |
| Cc         | Output matrix $(p \times n)$ of the continuous linear system.                   | Double (F64) |
| Dc         | Direct transmission (feedthrough) matrix $(p \times m)$ of the continuous linear system. The matrix is used only if the parameter UD=on. If UD=off, the dimensions of the Dc matrix are not checked. | Double (F64) |
| x0         | Initial value of the state vector (of dimension $n$) of the continuous linear system. | Double (F64) |
**DDELSSM – Discrete state space model of a linear system with time delay**

Block Symbol

![Block Symbol Image]

**Licence:** ADVANCED

**Function Description**

The **DDELSSM** block (Discrete State Space Model with time DELay) simulates behavior of a linear system with time delay $d$

$$
x(k+1) = A_dx(k) + B_du(k-d), \quad x(0) = x0
$$

$$
y(k) = C_dx(k) + D_du(k),
$$

where $k$ is the simulation step, $x(k) \in \mathbb{R}^n$ is the state vector, $x0 \in \mathbb{R}^n$ is the initial value of the state vector, $u(k) \in \mathbb{R}^m$ is the input vector, $y(k) \in \mathbb{R}^p$ is the output vector. The matrix $A_d \in \mathbb{R}^{n\times n}$ is the system dynamics matrix, $B_d \in \mathbb{R}^{n\times m}$ is the input matrix, $C_d \in \mathbb{R}^{p\times n}$ is the output matrix and $D_d \in \mathbb{R}^{p\times m}$ is the direct transmission (feedthrough) matrix. Number of steps of the delay $d$ is the largest integer such that $dT \leq d$, where $T$ is the block execution period.

All matrices are specified in the same format as in Matlab, i.e. the whole matrix is placed in brackets, elements are entered by rows, elements of a row are separated by spaces (blanks), rows are separated by semicolons. The $x0$ vector is a column, therefore the elements are separated by semicolons (each element is in a separate row).

During the real-time simulation, single simulation step of the above discrete state space model is computed in each execution time instant.

**Inputs**

- **R1** Reset signal. When R1 = on, the state vector $x$ is set to its initial value $x0$. The simulation continues on the falling edge of R1 (on→off).
- **HLD** Simulation output holds its value if HLD=on.
- **u1...u16** Simulated system inputs. First $m$ simulation inputs are used where $m$ is the number of columns of the matrix $B_d$.
CHAPTER 13. MODEL - DYNAMIC SYSTEMS SIMULATION

Outputs

\texttt{iE} \quad \text{Block error code}

\begin{align*}
0 \quad & \text{O.K., the simulation runs correctly} \\
-213 \quad & \text{incompatibility of the state space model matrices dimensions} \\
\text{xxx} \quad & \text{error code xxx of REXYGEN, see appendix C for details}
\end{align*}

\texttt{y1..y16} \quad \text{Simulated system outputs. First } p \text{ simulation outputs are used where } p \text{ is the number of rows of the matrix } C_d.

Parameters

\texttt{UD} \quad \text{Matrix } D_d \text{ usage flag. If } UD=\text{off then the } D_d \text{ matrix is not used for simulation (simulation behaves as if the } D_d \text{ matrix is zero).} \quad \text{Bool}

\texttt{del} \quad \text{Model time delay [s].} \quad \downarrow 0.0 \odot 0.0 \quad \text{Double (F64)}

\texttt{Ad} \quad \text{Matrix } (n \times n) \text{ of the discrete linear system dynamics.} \quad \text{Double (F64)}

\texttt{Bd} \quad \text{Input matrix } (n \times m) \text{ of the discrete linear system.} \quad \text{Double (F64)}

\texttt{Cd} \quad \text{Output matrix } (p \times n) \text{ of the discrete linear system.} \quad \text{Double (F64)}

\texttt{Dd} \quad \text{Direct transmission (feedthrough) matrix } (p \times m) \text{ of the discrete linear system. The matrix is used only if the parameter } UD=\text{on. If } UD=\text{off, the dimensions of the } D_d \text{ matrix are not checked.} \quad \text{Double (F64)}

\texttt{x0} \quad \text{Initial value of the state vector (of dimension } n) \text{ of the discrete linear system.} \quad \text{Double (F64)}
**DSSM – Discrete state space model of a linear system**

**Block Symbol**

Function Description

The DSSM block (Discrete State Space Model) simulates behavior of a linear system

\[
x(k + 1) = A_d x(k) + B_d u(k), \quad x(0) = x_0
\]

\[
y(k) = C_d x(k) + D_d u(k),
\]

where \( k \) is the simulation step, \( x(k) \in \mathbb{R}^n \) is the state vector, \( x_0 \in \mathbb{R}^n \) is the initial value of the state vector, \( u(k) \in \mathbb{R}^m \) is the input vector, \( y(k) \in \mathbb{R}^p \) is the output vector. The matrix \( A_d \in \mathbb{R}^{n \times n} \) is the system dynamics matrix, \( B_d \in \mathbb{R}^{n \times m} \) is the input matrix, \( C_d \in \mathbb{R}^{p \times n} \) is the output matrix and \( D_d \in \mathbb{R}^{p \times m} \) is the direct transmission (feedthrough) matrix.

All matrices are specified in the same format as in Matlab, i.e. the whole matrix is placed in brackets, elements are entered by rows, elements of a row are separated by spaces (blanks), rows are separated by semicolons. The \( x_0 \) vector is a column, therefore the elements are separated by semicolons (each element is in a separate row).

During the real-time simulation, single simulation step of the above discrete state space model is computed in each execution time instant.

**Inputs**

- **R1**
  - Reset signal. When \( R1 = \text{on} \), the state vector \( x \) is set to its initial value \( x_0 \). The simulation continues on the falling edge of \( R1 \) (on→off).
  - **Bool**

- **HLD**
  - Simulation output holds its value if \( \text{HLD=} \text{on} \).
  - **Bool**

- **u1..u16**
  - Simulated system inputs. First \( m \) simulation inputs are used where \( m \) is the number of columns of the matrix \( B_d \). **Double (F64)**

\( \odot 0.0 \)
CHAPTER 13. MODEL – DYNAMIC SYSTEMS SIMULATION

Outputs

<table>
<thead>
<tr>
<th>iE</th>
<th>Block error code</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>O.K., the simulation runs correctly.</td>
<td></td>
</tr>
<tr>
<td>-213</td>
<td>Incompatibility of the state space model matrices dimensions</td>
<td></td>
</tr>
<tr>
<td>xxx</td>
<td>Error code xxx of REXYGEN, see appendix C for details</td>
<td></td>
</tr>
</tbody>
</table>

| y₁..y₁₆ | Simulated system outputs. First p simulation outputs are used where p is the number of rows of the matrix Cₐ. | Double (F64) |

Parameters

| Uᵋ | Matrix Dᵋ usage flag. If Uᵋ=off then the Dᵋ matrix is not used for simulation (simulation behaves as if the Dᵋ matrix is zero). | Bool |
| Aᵋ | Matrix (n × n) of the discrete linear system dynamics. | Double (F64) |
| Bᵋ | Input matrix (n × m) of the discrete linear system. | Double (F64) |
| Cᵋ | Output matrix (p × n) of the discrete linear system. | Double (F64) |
| Dᵋ | Direct transmission (feedthrough) matrix (p × m) of the discrete linear system. The matrix is used only if the parameter Uᵋ=on. If Uᵋ=off, the dimensions of the Dᵋ matrix are not checked. | Double (F64) |
| x₀ | Initial value of the state vector (of dimension n) of the discrete linear system. | Double (F64) |
FOPDT – First order plus dead-time model

Block Symbol

Function Description
The FOPDT block is a discrete simulator of a first order continuous-time system with time delay, which can be described by the transfer function below:

\[ P(s) = \frac{k_0}{(\tau s + 1)} \cdot e^{-\delta t s} \]

The exact discretization at the sampling instants is used for discretization of the \( P(s) \) transfer function. The sampling period used for discretization is equivalent to the execution period of the FOPDT block.

Input
\( u \) Analog input of the block \( 0.0 \) Double (F64)

Output
\( y \) Analog output of the block Double (F64)

Parameters
\( k_0 \) Static gain \( 1.0 \) Double (F64)
\( \delta t \) Dead time [s] \( 0.0 \) Double (F64)
\( \tau \) Time constant \( 1.0 \) Double (F64)
\( n_{max} \) Size of delay buffer (number of samples) for the time delay \( \delta t \). Used for internal memory allocation. \( 1.0 \) Double (F64)
MDL – Process model

Block Symbol

License: STANDARD

Function Description

The MDL block is a discrete simulator of continuous-time system with transfer function

\[ F(s) = \frac{K_0 e^{-Ds}}{(\tau_1 s + 1)(\tau_2 s + 1)}, \]

where \( K_0 > 0 \) is the static gain \( k_0 \), \( D \geq 0 \) is the time-delay \( del \) and \( \tau_1, \tau_2 > 0 \) are the system time-constants \( tau1 \) and \( tau2 \).

Input

\( u \)  
Analog input of the block  
\( \odot 0.0 \)  
Double (F64)

Output

\( y \)  
Analog output of the block  
Double (F64)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k_0 )</td>
<td>Static gain</td>
<td>( \odot 1.0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>( del )</td>
<td>Dead time [s]</td>
<td>( \odot 0.0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>( tau1 )</td>
<td>The first time constant</td>
<td>( \odot 1.0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>( tau2 )</td>
<td>The second time constant</td>
<td>( \odot 2.0 )</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>( nmax )</td>
<td>Size of delay buffer (number of samples) for the time delay ( del ). Used for internal memory allocation.</td>
<td>( \downarrow 10 \uparrow 10000000 \odot 1.00E+03 )</td>
<td></td>
</tr>
</tbody>
</table>
MDLI – Process model with input-defined parameters

Block Symbol

Licence: STANDARD

Function Description
The MDLI block is a discrete simulator of continuous-time system with transfer function

\[ F(s) = \frac{K_0 e^{-Ds}}{(\tau_1 s + 1)(\tau_2 s + 1)}, \]

where \( K_0 > 0 \) is the static gain, \( D \geq 0 \) is the time-delay \( \text{del} \) and \( \tau_1, \tau_2 > 0 \) are the system time-constants \( \text{tau1} \) and \( \text{tau2} \). In contrary to the MDL block the system is time variant. The system parameters are determined by the input signals.

Inputs

- \( u \) Analog input of the block \( \circ 0.0 \) Double (F64)
- \( k0 \) Static gain \( \circ 0.0 \) Double (F64)
- \( \text{del} \) Dead time [s] \( \circ 0.0 \) Double (F64)
- \( \text{tau1} \) The first time constant \( \circ 0.0 \) Double (F64)
- \( \text{tau2} \) The second time constant \( \circ 0.0 \) Double (F64)

Output

- \( y \) Analog output of the block Double (F64)

Parameters

- \( nmax \) Size of delay buffer (number of samples) for the time delay \( \text{del} \). Long (I32)
  Used for internal memory allocation. \( \downarrow 10 \uparrow 10000000 \circ 1.00E+03 \)
MVD – Motorized valve drive

Block Symbol

Function Description

The MVD block simulates a servo valve. The UP (DN) input is a binary command for opening (closing) the valve at a constant speed $1/tv$, where $tv$ is a parameter of the block. The opening (closing) continues for UP = on (DN = on) until the full open $y = h\text{lim}$ (full closed $y = l\text{lim}$) position is reached. The full open (full closed) position is signalized by the end switch HS (LS). The initial position at start-up is $y = y_0$. If UP = DN = on or UP = DN = off, then the position of the valve remains unchanged (neither opening nor closing).

Inputs

<table>
<thead>
<tr>
<th>UP</th>
<th>Open</th>
<th>Bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>DN</td>
<td>Close</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>y</th>
<th>Valve position</th>
<th>Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HS</td>
<td>Upper end switch</td>
<td>Bool</td>
</tr>
<tr>
<td>LS</td>
<td>Lower end switch</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>y0</th>
<th>Initial valve position</th>
<th>0.0</th>
<th>Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tv</td>
<td>Time required for transition between $y = 0$ and $y = 1 , [s]$</td>
<td>10.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>h\text{lim}</td>
<td>Upper limit position (open)</td>
<td>1.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>l\text{lim}</td>
<td>Lower limit position (closed)</td>
<td>0.0</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
SOPDT – Second order plus dead-time model

Block Symbol

Function Description

The SOPDT block is a discrete simulator of a second order continuous-time system with time delay, which can be described by one of the transfer functions below. The type of the model is selected by the itf parameter.

\[
\begin{align*}
\text{itf} = 1 : & \quad P(s) = \frac{pb1 \cdot s + pb0}{s^2 + pa1 \cdot s + pa0} \cdot e^{-del \cdot s} \\
\text{itf} = 2 : & \quad P(s) = \frac{k0 \cdot (\tau1 \cdot s + 1)}{(\tau1 \cdot s + 1)(\tau2 \cdot s + 1)} \cdot e^{-del \cdot s} \\
\text{itf} = 3 : & \quad P(s) = \frac{k0 \cdot om^2 \cdot (\tau2 \cdot om \cdot s + 1)}{(s^2 + 2 \cdot \xi \cdot om \cdot s + om^2)} \cdot e^{-del \cdot s} \\
\text{itf} = 4 : & \quad P(s) = \frac{k0 \cdot (\tau1 \cdot s + 1)}{(\tau1 \cdot s + 1) \cdot s} \cdot e^{-del \cdot s}
\end{align*}
\]

For simulation of first order plus dead time systems (FOPDT) use the LLC block with parameter a set to zero.

The exact discretization at the sampling instants is used for discretization of the \( P(s) \) transfer function. The sampling period used for discretization is equivalent to the execution period of the SOPDT block.

Input

\( u \) \quad Analog input of the block \quad 0.0 \quad Double (F64)

Output

\( y \) \quad Analog output of the block \quad Double (F64)

Parameters

\( \text{itf} \) \quad Transfer function form \quad 1.00E+00 \quad Long (I32)

1 \quad A general form
2 \quad A form with real poles
3 \quad A form with complex poles
4 \quad A form with integrator
### CHAPTER 13. MODEL—DYNAMIC SYSTEMS SIMULATION

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>k0</td>
<td>Static gain</td>
<td>1.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>tau</td>
<td>Numerator time constant</td>
<td>0.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>tau1</td>
<td>The first time constant</td>
<td>1.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>tau2</td>
<td>The second time constant</td>
<td>1.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>om</td>
<td>Natural frequency</td>
<td>1.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>xi</td>
<td>Relative damping coefficient</td>
<td>1.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>pb0</td>
<td>Numerator coefficient: $s^0$</td>
<td>1.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>pb1</td>
<td>Numerator coefficient: $s^1$</td>
<td>1.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>pa0</td>
<td>Denominator coefficient: $s^0$</td>
<td>1.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>pa1</td>
<td>Denominator coefficient: $s^1$</td>
<td>1.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>del</td>
<td>Dead time [s]</td>
<td>0.0</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>nmax</td>
<td>Size of delay buffer (number of samples) for the time delay del.</td>
<td>1000000</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>

Used for internal memory allocation.

\[ \downarrow 10 \uparrow 1000000 \circ 1.0 \times 10^3 \]
Chapter 14

MATRIX – Blocks for matrix and vector operations

Contents

CMA – Array (vector/matrix) constant ........................................ 344
MB_DASUM – Sum of the absolute values ................................. 345
MB_DAXPY – Performs $y := a \cdot x + y$ for vectors $x, y$ .............. 346
MB_DCPY – Copies vector $x$ to vector $y$ .................................. 348
MB-DDOT – Dot product of two vectors ................................. 350
MB_DGEMM – Performs $C := \alpha \cdot op(A) \cdot op(B) + \beta \cdot C$, where
            $op(X) = X$ or $op(X) = X^\top$ ............................... 352
MB_DGEMV – Performs $y := \alpha \cdot A \cdot x + \beta \cdot y$ or $y := \alpha \cdot A^\top \cdot x$
            $+ \beta \cdot y$ ................................................. 354
MB_DGER – Performs $A := \alpha \cdot x \cdot y^\top + A$ ...................... 356
MB_DNRM2 – Euclidean norm of a vector .................................. 358
MB_DROT – Plain rotation of a vector ..................................... 359
MB_DSCAL – Scales a vector by a constant ............................... 361
MB_DSROT – Interchanges two vectors ..................................... 362
MB_DTRMM – Performs $B := \alpha \cdot op(A) \cdot B$ or $B := \alpha \cdot B \cdot op(A)$,
            where $op(X) = X$ or $op(X) = X^\top$ for triangular matrix $A$ 364
MB_DTRMV – Performs $x := A \cdot x$ or $x := A^\top \cdot x$ for triangular
            matrix $A$ .................................................... 366
MB_DTRS V – Solves one of the system of equations $A \cdot x = b$
            or $A^\top \cdot x = b$ for triangular matrix $A$ ...................... 369
ML_DGEBAK – Backward transformation to ML_DGEBAL of left or right
            eigenvectors .................................................. 372
ML_DGEBAL – Balancing of a general real matrix ...................... 374
ML_DGEBRD – Reduces a general real matrix to bidiagonal form by
            an orthogonal transformation ................................ 376

341
CHAPTER 14. MATRIX – BLOCKS FOR MATRIX AND VECTOR OPERATIONS

ML_DGECDFN – Estimates the reciprocal of the condition number of a general real matrix ............................................. 378
ML_DGEEES – Computes the eigenvalues, the Schur form, and, optionally, the matrix of Schur vectors ......................... 380
ML_DGEEV – Computes the eigenvalues and, optionally, the left and/or right eigenvectors ............................................. 382
ML_DGEEHRD – Reduces a real general matrix A to upper Hessenberg form ................................................................. 384
ML_DGELQF – Computes an LQ factorization of a real M-by-N matrix A ................................................................. 386
ML_DGELSD – Computes the minimum-norm solution to a real linear least squares problem ............................................. 388
ML_DGQR – Computes an QR factorization of a real M-by-N matrix A ................................................................. 390
ML_DGESDD – Computes the singular value decomposition (SVD) of a real M-by-N matrix A ............................................. 392
ML_DLACPY – Copies all or part of one matrix to another matrix ....................................................................................... 394
ML_DLANGE – Computes one of the matrix norms of a general matrix ............................................................................. 396
ML_DLASET – Initializes the off-diagonal elements and the diagonal elements of a matrix to given values .................. 398
ML_DTRSYL – Solves the real Sylvester matrix equation for quasi-triangular matrices A and B ............................................. 400
MX_AT – Get Matrix/Vector element ......................................................................................................................... 402
MX_ATSET – Set Matrix/Vector element ...................................................................................................................... 403
MX_CNADD – Add scalar to each Matrix/Vector element ............................................................................................ 404
MX_CNMUL – Multiply a Matrix/Vector by a scalar ...................................................................................................... 405
MX_CTODPA – Discretizes continuous model given by (A,B) to (Ad,Bd) using Pade approximations ................................ 406
MX_DIM – Matrix/Vector dimensions ....................................................................................................................... 408
MX_DIMSET – Set Matrix/Vector dimensions ............................................................................................................. 409
MX_DSAGET – Set subarray of A into B ....................................................................................................................... 411
MX_DSAREF – Set reference to subarray of A into B ..................................................................................................... 413
MX_DSASET – Set A into subarray of B ....................................................................................................................... 415
MX_DTRNSP – General matrix transposition: B := alpha*A’*T .................................................................................. 417
MX_DTRNSQ – Square matrix in-place transposition: A := alpha*A’*T ........................................................................ 419
MX_FILL – Fill real matrix or vector ........................................................................................................................ 420
MX_MAT – Matrix data storage block ........................................................................................................................ 421
MX_RAND – Randomly generated matrix or vector .................................................................................................. 422
MX_REFCOPY – Copies input references of matrices A and B to their output references ............................................. 424
MX_SLFS – Save or load a Matrix/Vector into file or string ....................................................................................... 425
MX_VEC – Vector data storage block ....................................................................................................................... 427
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MX_WRITE</strong></td>
<td>Write a Matrix/Vector to the console/system log</td>
<td>428</td>
</tr>
<tr>
<td><strong>RTOV</strong></td>
<td>Vector multiplexer</td>
<td>430</td>
</tr>
<tr>
<td><strong>SwVMR</strong></td>
<td>Vector/matrix/reference signal switch</td>
<td>431</td>
</tr>
<tr>
<td><strong>VTOR</strong></td>
<td>Vector demultiplexer</td>
<td>432</td>
</tr>
</tbody>
</table>
CNA – Array (vector/matrix) constant

Block Symbol Licence: STANDARD

Function Description

The block CNA allocates memory for nmax elements of the type etype of the vector/matrix referenced by the output vec and initializes all elements to data stored in the parameter acn.

If the string parameter filename is not empty then it loads initialization data from the filename file on the host computer in CSV format. Column separator can be comma or semicolon or space (but the same in the whole file), decimal separator have to be dot, row separator is new line. Empty lines are skipped.

If the parameter TRN = on then the output reference vec contains transposed data.

Note: In case of etype = Large (I64), values loaded from parameter acn are converted to double-precision float due to implementation reasons, so you can loose precision for very large values. If this could be a problem, use external file for initialization which does not have this issue.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>CSV data file</td>
<td>String</td>
</tr>
<tr>
<td>TRN</td>
<td>Transpose loaded matrix</td>
<td>Bool</td>
</tr>
<tr>
<td>nmax</td>
<td>Allocated size of output matrix (total number of items)</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>etype</td>
<td>Type of elements</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 ...... Bool</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ...... Byte (U8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 ...... Short (I16)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 ...... Long (I32)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 ...... Word (U16)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6 ...... DWord (U32)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7 ...... Float (F32)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 ...... Double (F64)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10 ...... Large (I64)</td>
<td></td>
</tr>
<tr>
<td>acn</td>
<td>Initial array value</td>
<td></td>
</tr>
<tr>
<td></td>
<td>344.00E+00</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vec</td>
<td>Reference to vector/matrix data</td>
</tr>
</tbody>
</table>

Reference
MB_DASUM – Sum of the absolute values

Block Symbol

Function Description

The output reference yX is always set to the input reference uX. If HLD = on then nothing is computed otherwise the BLAS function DASUM is called internally:

\[ \text{value} = \text{DASUM}(N, uX, \text{INCX}); \]

where the values N and INCX are set in the following way:

- If the input \( n > 0 \) then \( N \) is set to \( n \) else \( N \) is set to the current number of the vector or matrix elements \( \text{CNT} \) referenced by \( uX \).
- If the input \( \text{INCX} > 0 \) then \( \text{INCX} \) is set to \( \text{INCX} \) else \( \text{INCX} \) is set to 1.

The error flag \( E \) is set to on if:

- the reference \( uX \) is not defined (i.e. input \( uX \) is not connected),
- \( n < 0 \) or \( \text{INCX} < 0 \),
- \( (N - 1) \times \text{INCX} + 1 > \text{CNT} \).


Inputs

- \( uX \) : Input reference to vector x
- \( n \) : Number of processed vector elements
- \( \text{INCX} \) : Index increment of vector x
- \( \text{HLD} \) : Hold

Outputs

- \( yX \) : Output reference to vector x
- \( \text{value} \) : Return value of the function
- \( E \) : Error flag

Licence: STANDARD
MB_DAXPY – Performs $y := a^* x + y$ for vectors $x, y$

Block Symbol

Licence: STANDARD

Function Description

The output references $yX$ and $yY$ are always set to the corresponding input references $uX$ and $uY$. If HLD = on then nothing is computed otherwise the BLAS function DAXPY is called internally:

$$\text{DAXPY}(N, a, uX, \text{INCX}, uY, \text{INCY});$$

where the values $N$, INCX and INCY are set in the following way:

- If the input $n > 0$ then $N$ is set to $n$ else $N$ is set to the current number of the vector or matrix elements CNTX referenced by $uY$.
- If the input incx ≠ 0 then INCX is set to incx else INCX is set to 1.
- If the input incy ≠ 0 then INCY is set to incy else INCY is set to 1.

The error flag $E$ is set to on if:

- the reference $uX$ or $uY$ is not defined (i.e. input $uX$ or $uY$ is not connected),
- $n < 0$,
- $(N - 1) \times |\text{INCX}| + 1 > \text{CNTX}$, where $\text{CNTX}$ is a number of the vector or matrix elements referenced by $uX$,
- $(N - 1) \times |\text{INCY}| + 1 > \text{CNTY}$.


Inputs

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$uX$</td>
<td>Input reference to vector $x$</td>
<td>Reference</td>
<td>$uY$</td>
<td>Input reference to vector $y$</td>
<td>Reference</td>
</tr>
<tr>
<td>$a$</td>
<td>Scalar coefficient $a$</td>
<td>Double (F64)</td>
<td>$n$</td>
<td>Number of processed vector elements</td>
<td>0.00E+00 Long (I32)</td>
</tr>
</tbody>
</table>
incx    Index increment of vector x    Long (132)
in cy    Index increment of vector y    Long (132)
HLD     Hold                          Bool

Outputs
yX      Output reference to vector x  Reference
yY      Output reference to vector y  Reference
E       Error indicator              Bool
348 CHAPTER 14. MATRIX – BLOCKS FOR MATRIX AND VECTOR OPERATIONS

MB_DCOPY – Copies vector x to vector y

Block Symbol

Licence: STANDARD

Function Description

The output references yX and yY are always set to the corresponding input references uX and uY. If HLD = on then nothing is computed otherwise the BLAS function DCOPY is called internally:

\[ \text{DCOPY}(N, uX, \text{INCX}, uY, \text{INCY}); \]

where the values N, INCX and INCY are set in the following way:

• If the input n > 0 then N is set to n else N is set to the current number of the vector or matrix elements CNTX referenced by uX.

• If the input incx ≠ 0 then INCX is set to incx else INCX is set to 1.

• If the input incy ≠ 0 then INCY is set to incy else INCY is set to 1.

The error flag E is set to on if:

• the reference uX or uY is not defined (i.e. input uX or uY is not connected),

• n < 0,

• \((N - 1) \times |\text{INCX}| + 1 > \text{CNTX},\)

• \((N - 1) \times |\text{INCY}| + 1 > \text{CNTY},\) where CNTY is a number of the vector or matrix elements referenced by uY.


Inputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uX</td>
<td>Input reference to vector x</td>
<td>Reference</td>
</tr>
<tr>
<td>uY</td>
<td>Input reference to vector y</td>
<td>Reference</td>
</tr>
<tr>
<td>n</td>
<td>Number of processed vector elements</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>incx</td>
<td>Index increment of vector x</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>incy</td>
<td>Index increment of vector y</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>
## Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yX</td>
<td>Output reference to vector x</td>
<td>Reference</td>
</tr>
<tr>
<td>yY</td>
<td>Output reference to vector y</td>
<td>Reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
MB_DDOT – Dot product of two vectors

Block Symbol

Function Description

The output references yX and yY are always set to the corresponding input references uX and uY. If HLD = on then nothing is computed otherwise the BLAS function DDOT is called internally:

\[ \text{DDOT}(N, uX, INCX, uY, INCY); \]

where the values N, INCX and INCY are set in the following way:

- If the input \( n > 0 \) then \( N \) is set to \( n \) else \( N \) is set to the current number of the vector or matrix elements \( \text{CNTX} \) referenced by \( uX \).
- If the input \( \text{INCX} \neq 0 \) then \( \text{INCX} \) is set to \( \text{INCX} \) else \( \text{INCX} \) is set to 1.
- If the input \( \text{INCY} \neq 0 \) then \( \text{INCY} \) is set to \( \text{INCY} \) else \( \text{INCY} \) is set to 1.

The error flag \( E \) is set to on if:

- the reference \( uX \) or \( uY \) is not defined (i.e. input \( uX \) or \( uY \) is not connected),
- \( n < 0 \),
- \( (N - 1) * |\text{INCX}| + 1 > \text{CNTX} \),
- \( (N - 1) * |\text{INCY}| + 1 > \text{CNTY} \), where \( \text{CNTY} \) is a number of the vector or matrix elements referenced by \( uY \).


Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uX</td>
<td>Input reference to vector x</td>
<td>Reference</td>
</tr>
<tr>
<td>uY</td>
<td>Input reference to vector y</td>
<td>Reference</td>
</tr>
<tr>
<td>n</td>
<td>Number of processed vector elements</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>INCX</td>
<td>Index increment of vector x</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>INCY</td>
<td>Index increment of vector y</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yX</td>
<td>Output reference to vector x</td>
<td>Reference</td>
</tr>
<tr>
<td>yY</td>
<td>Output reference to vector y</td>
<td>Reference</td>
</tr>
<tr>
<td>value</td>
<td>Return value of the function</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
**MB_DGEMM** — Performs 

\[ C := \alpha \cdot \text{op}(A) \cdot \text{op}(B) + \beta \cdot C, \]

where \( \text{op}(X) = X \) or \( \text{op}(X) = X^T \)

Block Symbol

License: **STANDARD**

**Function Description**

The output references \( y_A, y_B \) and \( y_C \) are always set to the corresponding input references \( u_A, u_B \) and \( u_C \). If \( \text{HLD} = \text{on} \) then nothing is computed otherwise the BLAS function **DGEMM** is called internally:

\[
\text{DGEMM}(\text{stransA}, \text{stransB}, M, N, K, \alpha, u_A, LDA, u_B, LDB, \beta, u_C, LDC);
\]

where parameters of **DGEMM** are set in the following way:

- Integer inputs \( \text{transa} \) and \( \text{transb} \) are mapped to strings \( \text{stransA} \) and \( \text{stransB} \):
  \[
  \{0, 1\} \rightarrow "N", \{2\} \rightarrow "T" \text{ and } \{3\} \rightarrow "C".
  \]
- \( M \) is number of rows of the matrix referenced by \( u_C \).
- \( N \) is number of columns of the matrix referenced by \( u_C \).
- If the input \( \text{transa} \) is equal to 0 or 1 then \( K \) is number of columns else \( K \) is number of rows of the matrix referenced by \( u_A \).
- \( LDA, LDB \) and \( LDC \) are leading dimensions of matrices referenced by \( u_A, u_B \) and \( u_C \).

The error flag \( E \) is set to \text{on} if:

- the reference \( u_A \) or \( u_B \) or \( u_C \) is not defined (i.e. input \( u_A \) or \( u_B \) or \( u_C \) is not connected),
- \( \text{transa} \) or \( \text{transb} \) is less than 0 or greater than 3
- \( K \neq K_B \); if the input \( \text{transb} \) is equal to 0 or 1 then \( K_B \) is number of rows else \( K_B \) is number of columns of the matrix referenced by \( u_B \) (i.e. matrices \( \text{op}(A) \) and \( \text{op}(B) \) have to be multiplicable).
- the call of the function **DGEMM** returns error using the function **XERBLA**, see the system log.

## Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>uB</td>
<td>Input reference to matrix B</td>
<td>Reference</td>
</tr>
<tr>
<td>uC</td>
<td>Input reference to matrix C</td>
<td>Reference</td>
</tr>
<tr>
<td>transa</td>
<td>Transposition of matrix A</td>
<td>Long (132)</td>
</tr>
<tr>
<td>transb</td>
<td>Transposition of matrix B</td>
<td>Long (132)</td>
</tr>
<tr>
<td>alpha</td>
<td>Scalar coefficient alpha</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>beta</td>
<td>Scalar coefficient beta</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>yB</td>
<td>Output reference to matrix B</td>
<td>Reference</td>
</tr>
<tr>
<td>yC</td>
<td>Output reference to matrix C</td>
<td>Reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
MB_DGEMV – Performs $y := \alpha A^T x + \beta y$ or $y := \alpha A^x + \beta x$

Block Symbol

License: STANDARD

Function Description

The output references $yA$, $x$ and $y$ are always set to the corresponding input references $uA$, $uX$ and $uY$. If $HLD = \text{on}$ then nothing is computed otherwise the BLAS function DGEMV is called internally:

$$\text{DGEMV}(\text{sTRANS, M, N, alpha, uA, LDA, uX, INCX, beta, uY, INCY});$$

where parameters of DGEMV are set in the following way:

- Integer input $\text{trans}$ is mapped to the string $\text{sTRANS}$: $\{0, 1\} \rightarrow \text{"N"}, \{2\} \rightarrow \text{"T"}$ and $\{3\} \rightarrow \text{"C"}$.
- $M$ is number of rows of the matrix referenced by $uA$.
- $N$ is number of columns of the matrix referenced by $uA$.
- $LDA$ is the leading dimension of matrix referenced by $uA$.
- If the input $\text{incx} \neq 0$ then $\text{INCX}$ is set to $\text{incx}$ else $\text{INCX}$ is set to 1.
- If the input $\text{incy} \neq 0$ then $\text{INCY}$ is set to $\text{incy}$ else $\text{INCY}$ is set to 1.

The error flag $E$ is set to on if:

- the reference $uA$ or $uX$ or $uY$ is not defined (i.e. input $uA$ or $uX$ or $uY$ is not connected),
- $\text{trans}$ is less than 0 or greater than 3
- the call of the function DGEMV returns error using the function XERBLA, see the system log.

### Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(u_A)</td>
<td>Input reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>(u_X)</td>
<td>Input reference to vector x</td>
<td>Reference</td>
</tr>
<tr>
<td>(u_Y)</td>
<td>Input reference to vector y</td>
<td>Reference</td>
</tr>
<tr>
<td>(\text{trans})</td>
<td>Transposition of the input matrix</td>
<td>(\downarrow 0 \uparrow 3\times 0.00E+00) Long (I32)</td>
</tr>
<tr>
<td>(\text{incx})</td>
<td>Index increment of vector x</td>
<td>(0.00E+00) Long (I32)</td>
</tr>
<tr>
<td>(\text{incy})</td>
<td>Index increment of vector y</td>
<td>(0.00E+00) Long (I32)</td>
</tr>
<tr>
<td>(\alpha)</td>
<td>Scalar coefficient alpha</td>
<td>(0.0) Double (F64)</td>
</tr>
<tr>
<td>(\beta)</td>
<td>Scalar coefficient beta</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>(\text{HLD})</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(y_A)</td>
<td>Output reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>(y_X)</td>
<td>Output reference to vector x</td>
<td>Reference</td>
</tr>
<tr>
<td>(y_Y)</td>
<td>Output reference to vector y</td>
<td>Reference</td>
</tr>
<tr>
<td>(\text{E})</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
MB_DGER – Performs $A := \alpha x^Ty^T + A$

Block Symbol

License: STANDARD

Function Description

The output references $yX$, $yY$ and $yA$ are always set to the corresponding input references $uX$, $uY$ and $uA$. If $HLD = \text{on}$ then nothing is computed otherwise the BLAS function DGER is called internally:

\[
\text{DGER}(M, N, \alpha, uX, uY, INCX, INCY, uA, LDA);
\]

where parameters of DGER are set in the following way:

- $M$ is number of rows of the matrix referenced by $uA$.
- $N$ is number of columns of the matrix referenced by $uA$.
- If the input $\text{incx} \neq 0$ then $\text{INcx}$ is set to $\text{incx}$ else $\text{INcx}$ is set to 1.
- If the input $\text{incy} \neq 0$ then $\text{INcy}$ is set to $\text{incy}$ else $\text{INcy}$ is set to 1.
- $LDA$ is the leading dimension of matrix referenced by $uA$.

The error flag $E$ is set to on if:

- the reference $uX$ or $uY$ or $uA$ is not defined (i.e. input $uX$ or $uY$ or $uA$ is not connected),
- the call of the function DGER returns error using the function XERBLA, see the system log.


Inputs

- $uX$: Input reference to vector $x$
- $uY$: Input reference to vector $y$
- $uA$: Input reference to matrix $A$
- $\text{incx}$: Index increment of vector $x$ [0.00E+00; Long (I32)]
- $\text{incy}$: Index increment of vector $y$ [0.00E+00; Long (I32)]
- $\alpha$: Scalar coefficient $\alpha$ [0.0; Double (F64)]
- $HLD$: Hold [Bool]
### Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yX</td>
<td>Output reference to vector x</td>
<td>Reference</td>
</tr>
<tr>
<td>yY</td>
<td>Output reference to vector y</td>
<td>Reference</td>
</tr>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
MB_DNRM2 – Euclidean norm of a vector

Block Symbol

Function Description

The output reference yX is always set to the input reference uX. If HLD = on then nothing is computed otherwise the BLAS function DNRM2 is called internally:

\[ \text{value} = \text{DNRM2}(N, \text{uX}, \text{INCX}); \]

where the values N and INCX are set in the following way:

- If the input n \( > 0 \) then N is set to n else N is set to the current number of the vector or matrix elements CNT referenced by uX.
- If the input incx \( > 0 \) then INCX is set to incx else INCX is set to 1.

The error flag E is set to on if:

- the reference uX is not defined (i.e. input uX is not connected),
- n \( < 0 \) or incx \( < 0 \),
- \((N - 1) * |\text{INCX}| + 1 \) > CNT.

See BLAS documentation \[6\] for more details.

Use the block MB_DLANG for computation of various norms of a matrix.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uX</td>
<td>Input reference to vector x</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>Number of processed vector elements</td>
<td>0.00E+00</td>
</tr>
<tr>
<td>incx</td>
<td>Index increment of vector x</td>
<td>0.00E+00</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>yX</td>
<td>Output reference to vector x</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>Return value of the function</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
MB_DROT – Plain rotation of a vector

Block Symbol

Function Description

The output references yX and yY are always set to the corresponding input references uX and uY. If HLD = on then nothing is computed otherwise the BLAS function DROT is called internally:

\[
\text{DROT}(N, uX, INCX, uY, INCY, c, s);
\]

where parameters of DROT are set in the following way:

- If the input \( n > 0 \) then \( N \) is set to \( n \) else \( N \) is set to the current number of the vector or matrix elements CNTX referenced by \( uX \).

- If the input \( \text{incx} \neq 0 \) then \( \text{INCX} \) is set to \( \text{incx} \) else \( \text{INCX} \) is set to 1.

- If the input \( \text{incy} \neq 0 \) then \( \text{INCY} \) is set to \( \text{incy} \) else \( \text{INCY} \) is set to 1.

The error flag E is set to on if:

- the reference \( uX \) or \( uY \) is not defined (i.e. input \( uX \) or \( uY \) is not connected),

- \( n < 0 \),

- \( (N - 1) * |\text{INCX}| + 1 > \text{CNTX} \),

- \( (N - 1) * |\text{INCY}| + 1 > \text{CNTY} \), where \( \text{CNTY} \) is a number of the vector or matrix elements referenced by \( uY \).


Inputs

| \( \text{uX} \) | Input reference to vector x | Reference |
| \( \text{uY} \) | Input reference to vector y | Reference |
| \( n \) | Number of processed vector elements | \( \pm 0.000E+00 \) Long (I32) |
| \( \text{incx} \) | Index increment of vector x | \( \pm 0.000E+00 \) Long (I32) |
### CHAPTER 14. MATRIX - BLOCKS FOR MATRIX AND VECTOR OPERATIONS

<table>
<thead>
<tr>
<th>Incy</th>
<th>Index increment of vector y</th>
<th>0.00E+00 Long (I32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Scalar coefficient c</td>
<td>0.0 Double (F64)</td>
</tr>
<tr>
<td>s</td>
<td>Scalar coefficient s</td>
<td>0.0 Double (F64)</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>yX</th>
<th>Output reference to vector x</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>yY</td>
<td>Output reference to vector y</td>
<td>Reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
MB_DSCAL – Scales a vector by a constant

Block Symbol

Licence: STANDARD

Function Description

The output references yX is always set to the corresponding input reference uX. If HLD = on then nothing is computed otherwise the BLAS function DSCAL is called internally:

DSCAL(N, a, uX, INCX);

where parameters of DSCAL are set in the following way:

- If the input n > 0 then N is set to n else N is set to the current number of the vector or matrix elements CNT referenced by uX.
- If the input incx ≠ 0 then INCX is set to incx else INCX is set to 1.

The error flag E is set to on if:

- the reference uX is not defined (i.e. input uX is not connected),
- n < 0 or incx < 0,
- (N − 1) * INCX + 1 > CNT.


Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uX</td>
<td>Input reference to vector x</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>Scalar coefficient a</td>
<td>⊙0.0</td>
</tr>
<tr>
<td>n</td>
<td>Number of processed vector elements</td>
<td>⊙0.0E+00</td>
</tr>
<tr>
<td>incx</td>
<td>Index increment of vector x</td>
<td>⊙0.0E+00</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td></td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>yX</td>
<td>Output reference to vector x</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
CHAPTER 14. MATRIX - BLOCKS FOR MATRIX AND VECTOR OPERATIONS

MB_DSWAP  -  Interchanges two vectors

Block Symbol

Function Description

The output references yX and yY are always set to the corresponding input references uX and uY. If HLD = on then nothing is computed otherwise the BLAS function 

\texttt{DSWAP(N, uX, INCX, uY, INCY)};

where the values N, INCX and INCY are set in the following way:

- If the input \( n > 0 \) then N is set to n else N is set to the current number of the vector or matrix elements CNTX referenced by uX.

- If the input incx \( \neq 0 \) then INCX is set to incx else INCX is set to 1.

- If the input incy \( \neq 0 \) then INCY is set to incy else INCY is set to 1.

The error flag E is set to on if:

- the reference uX or uY is not defined (i.e. input uX or uY is not connected),

- \( n < 0 \),

- \( (N - 1) * |INCX| + 1 > CNTX, \)

- \( (N - 1) * |INCY| + 1 > CNTY, \) where CNTY is a number of the vector or matrix elements referenced by uY.


Inputs

\begin{tabular}{lll}
\textbf{uX} & Input reference to vector x & Reference \\
\textbf{uY} & Input reference to vector y & Reference \\
\textbf{n} & Number of processed vector elements & \( \circ 0.00E+00 \) Long (I32) \\
\textbf{incx} & Index increment of vector x & \( \circ 0.00E+00 \) Long (I32) \\
\textbf{incy} & Index increment of vector y & \( \circ 0.00E+00 \) Long (I32) \\
\textbf{HLD} & Hold & Bool \\
\end{tabular}
## Outputs

<table>
<thead>
<tr>
<th>yX</th>
<th>Output reference to vector x</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>yY</td>
<td>Output reference to vector y</td>
<td>Reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
**MB_DTRMM** — Performs $B := \alpha \cdot \text{op}(A) \cdot B$ or $B := \alpha \cdot B \cdot \text{op}(A)$, where $\text{op}(X) = X$ or $\text{op}(X) = X^\top$ for triangular matrix $A$.

**Block Symbol**

**Function Description**

The output references $y_A$ and $y_B$ are always set to the corresponding input references $u_A$ and $u_B$. If `HLD` = on then nothing is computed otherwise the BLAS function `DTRMM` is called internally:

$$\text{DTRMM}(\text{RSIDE}, \text{LUPLO}, \text{TRANSA}, \text{DIAG}, M, N, \alpha, u_A, LDA, u_B, LDB);$$

where parameters of `DTRMM` are set in the following way:

- If `RSIDE` = on then the string `RSIDE` is set to "R" else it is set to "L".
- If `LUPLO` = on then the string `LUPLO` is set to "L" else it is set to "U".
- Integer input `TRANSA` is mapped to the string `TRANSA`: $\{0, 1\} \rightarrow "N", \{2\} \rightarrow "T"$ and $\{3\} \rightarrow "C"$.
- If `DIAG` = on then the string `DIAG` is set to "N" else it is set to "U".
- $M$ is the number of rows of the matrix referenced by $u_B$.
- $N$ is the number of columns of the matrix referenced by $u_B$.
- `LDA` and `LDB` are leading dimensions of matrices referenced by $u_A$ and $u_B$.

The error flag $E$ is set to on if:

- the reference $u_A$ or $u_B$ is not defined (i.e. input $u_A$ or $u_B$ is not connected),
- `TRANSA` is less than 0 or greater than 3,
- matrix referenced by $u_A$ is not square or is not compatible with the matrix referenced by $u_B$,
- the call of the function `DTRMM` returns error using the function `XERBLA`, see the system log.

**Inputs**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>uB</td>
<td>Input reference to matrix B</td>
<td>Reference</td>
</tr>
<tr>
<td>RSIDE</td>
<td>Operation is applied from right side</td>
<td>Bool</td>
</tr>
<tr>
<td>LUPLO</td>
<td>Matrix A is a lower triangular matrix</td>
<td>Bool</td>
</tr>
<tr>
<td>transa</td>
<td>Transposition of matrix A</td>
<td>( \downarrow0 \uparrow3 \circ0.00E+00 ) Long (I32)</td>
</tr>
<tr>
<td>NDIAG</td>
<td>Matrix A is not assumed to be unit triangular</td>
<td>Bool</td>
</tr>
<tr>
<td>alpha</td>
<td>Scalar coefficient alpha</td>
<td>( \circ0.0 ) Double (F64)</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

**Outputs**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>yB</td>
<td>Output reference to matrix B</td>
<td>Reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
MB_DTRMV – Performs \( x := A \cdot x \) or \( x := A^{\top} \cdot x \) for triangular matrix \( A \)

Block Symbol

<table>
<thead>
<tr>
<th>yA</th>
<th>uX</th>
<th>yX</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td>uA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INCX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HLD</td>
</tr>
<tr>
<td>MB_DTRMV</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Licence: STANDARD

Function Description

The output references \( yA \) and \( yX \) are always set to the corresponding input references \( uA \) and \( uX \). If \( HLD = \text{on} \) then nothing is computed otherwise the BLAS function DTRMV is called internally:

\[
\text{DTRMV}(\text{sLUPLO}, \text{sTRANS}, \text{sNDIAG}, N, uA, LDA, uX, INCX);
\]

where parameters of DTRMV are set in the following way:

- If \( \text{LUPLO} = \text{on} \) then the string \( \text{sLUPLO} \) is set to "L" else it is set to "U".
- Integer input \( \text{trans} \) is mapped to the string \( \text{sTRANS} \): \( \{0, 1\} \rightarrow "N", \{2\} \rightarrow "T" \) and \( \{3\} \rightarrow "C". \)
- If \( \text{NDIAG} = \text{on} \) then the string \( \text{sNDIAG} \) is set to "N" else it is set to "U".
- \( N \) is number of rows and columns of the square matrix referenced by \( uA \).
- \( LDA \) is the leading dimension of matrix referenced by \( uA \).
- If the input \( \text{incx} \neq 0 \) then \( \text{INCX} \) is set to \( \text{incx} \) else \( \text{INCX} \) is set to 1.

The error flag \( E \) is set to \( \text{on} \) if:

- the reference \( uA \) or \( uX \) is not defined (i.e. input \( uA \) or \( uX \) is not connected),
- \( \text{trans} \) is less than 0 or greater than 3,
- matrix referenced by \( uA \) is not square,
- \((N - 1) \cdot |\text{INCX}| + 1 > \text{CNTX}\), where \( \text{CNTX} \) is a number of the vector or matrix elements referenced by \( uX \).
- the call of the function DTRMV returns error using the function XERBLA, see the system log.
### Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>uX</td>
<td>Input reference to vector x</td>
<td>Reference</td>
</tr>
<tr>
<td>LUPL0</td>
<td>Matrix A is a lower triangular matrix</td>
<td>Bool</td>
</tr>
<tr>
<td>trans</td>
<td>Transposition of the input matrix</td>
<td>▼0 ↑3</td>
</tr>
<tr>
<td>NDIAG</td>
<td>Matrix A is not assumed to be unit triangular</td>
<td>Bool</td>
</tr>
<tr>
<td>incx</td>
<td>Index increment of vector x</td>
<td>▽0.00E+00 Long (I32)</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>yX</td>
<td>Output reference to vector x</td>
<td>Reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
MB_DTRSV – Solves one of the system of equations $A^t x = b$ or $A^T x = b$ for triangular matrix $A$

Block Symbol

Licence: STANDARD

Function Description

The output references $y_A$ and $y_X$ are always set to the corresponding input references $u_A$ and $u_X$. If $HLD = \text{on}$ then nothing is computed otherwise the BLAS function DTRSV is called internally:

$$DTRSV(sLUPLO, sTRANS, sNDIAG, N, uA, LDA, uX, INCX);$$

where parameters of DTRSV are set in the following way:

- If $LUPLO = \text{on}$ then the string $sLUPLO$ is set to "L" else it is set to "U".
- Integer input $sTRANS$ is mapped to the string $sTRANS$: $\{0, 1\} \rightarrow \text{"N"}, \{2\} \rightarrow \text{"T"}$ and $\{3\} \rightarrow \text{"C"}$.
- If $NDIAG = \text{on}$ then the string $sNDIAG$ is set to "Y" else it is set to "U".
- $N$ is number of rows and columns of the square matrix referenced by $u_A$.
- $LDA$ is the leading dimension of matrix referenced by $u_A$.
- If the input $\text{incx} \neq 0$ then $INX$ is set to $\text{incx}$ else $INX$ is set to 1.

The error flag $E$ is set to $\text{on}$ if:

- the reference $u_A$ or $u_X$ is not defined (i.e. input $u_A$ or $u_X$ is not connected),
- $sTRANS$ is less than 0 or greater than 3,
- matrix referenced by $u_A$ is not square,
- $(N - 1) * |\text{INCX}| + 1 > CNTX$, where $CNTX$ is a number of the vector or matrix elements referenced by $u_X$.
- the call of the function DTRSV returns error using the function XERBLA, see the system log.
## Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>uX</td>
<td>Input reference to vector x</td>
<td>Reference</td>
</tr>
<tr>
<td>LUPL0</td>
<td>Matrix A is a lower triangular matrix</td>
<td>Bool</td>
</tr>
<tr>
<td>trans</td>
<td>Transposition of the input matrix</td>
<td>↓0 ↑3</td>
</tr>
<tr>
<td>NDIA0</td>
<td>Matrix A is not assumed to be unit triangular</td>
<td>Bool</td>
</tr>
<tr>
<td>incx</td>
<td>Index increment of vector x</td>
<td>○0.00E+00</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>yX</td>
<td>Output reference to vector x</td>
<td>Reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
ML_DGEBAK — Backward transformation to ML_DGEBAL of left or right eigenvectors

Block Symbol

Function Description

The output references ySCALE and yV are always set to the corresponding input references uSCALE and uV. If HLD = on then nothing is computed otherwise the LAPACK function DGEBAK is called internally:

\[
\text{DGEBAK}(\text{SJOB}, \text{RSIDE}, N, \text{iLO}, \text{IHI}, \text{uSCALE}, M, \text{uV}, \text{LDV}, \text{info});
\]

where parameters of DGEBAK are set in the following way:

- Integer input job is mapped to the string sJOB: \{0, 1\} → "N", \{2\} → "P", \{3\} → "S" and \{4\} → "B".
- If RSIDE = on then the string sRSIDE is set to "R" else it is set to "L".
- \(N\) is number of elements of the vector referenced by uSCALE.
- If the input ihi ≠ 0 then IHI is set to ihi else IHI is set to \(N - 1\).
- \(M\) is number of columns of the matrix referenced by uV.
- LDV is the leading dimension of the matrix referenced by uV.
- info is return code from the function DGEBAK.

The error flag E is set to on if:

- the reference uSCALE or uV is not defined (i.e. input uSCALE or uV is not connected),
- the call of the function DGEBAK returns error using the function XERBLA, see the return code info and system log.

Emphasize that the indices iLO and IHI start from zero unlike FORTRAN version where they start from one. See LAPACK documentation [7] for more details.
<table>
<thead>
<tr>
<th><strong>Inputs</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>uSCALE</td>
<td>Input reference to vector SCALE</td>
</tr>
<tr>
<td>uV</td>
<td>Reference to matrix of right or left eigenvectors to be transformed</td>
</tr>
<tr>
<td>job</td>
<td>Type of backward transformation required</td>
</tr>
<tr>
<td>RSIDE</td>
<td>Operation is applied from right side</td>
</tr>
<tr>
<td>ilo</td>
<td>Zero based low row and column index of working submatrix</td>
</tr>
<tr>
<td>ihi</td>
<td>Zero based high row and column index of working submatrix</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Outputs</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ysCALE</td>
<td>Output reference to vector SCALE</td>
</tr>
<tr>
<td>yV</td>
<td>Reference to matrix of transformed right or left eigenvectors</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
</tr>
<tr>
<td>info</td>
<td>LAPACK function result info. If info = -i, the i-th argument had an illegal value</td>
</tr>
</tbody>
</table>
ML_DGEBAL — Balancing of a general real matrix

Block Symbol

Function Description

The output references yA and ySCALE are always set to the corresponding input references uA and uSCALE. If HLD = on then nothing is computed otherwise the LAPACK function DGEBAL is called internally:

\[ \text{DGEBAL} \left( \text{JOB}, N, uA, LDA, ilo, ihi, uSCALE, info \right) \]

where parameters of DGEBAL are set in the following way:

- Integer input JOB is mapped to the string sJOB: \{0, 1\} \rightarrow "N", \{2\} \rightarrow "P", \{3\} \rightarrow "S" and \{4\} \rightarrow "B".
- \( N \) is number of columns of the square matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.
- ilo and ihi are returned low and high row and column indices of the balanced submatrix of the matrix referenced by uA.
- info is return code from the function DGEBAL.

The error flag E is set to on if:

- the reference uA or uSCALE is not defined (i.e. input uA or uSCALE is not connected),
- matrix referenced by uA is not square,
- number of elements of the vector referenced by uSCALE is less than \( N \).
- the call of the function DGEBAL returns error using the function XERBLA, see the return code info and system log.

Emphasize that the indices ilo and ihi start from zero unlike FORTRAN version where they start from one. See LAPACK documentation [7] for more details.
Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>uSCALE</td>
<td>Input reference to vector SCALE</td>
<td>Reference</td>
</tr>
<tr>
<td>job</td>
<td>Specifies the operations to be performed on matrix A</td>
<td>Long (132)</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>ySCALE</td>
<td>Output reference to vector SCALE</td>
<td>Reference</td>
</tr>
<tr>
<td>ilo</td>
<td>Zero based low row and column index of working submatrix</td>
<td>Long (132)</td>
</tr>
<tr>
<td>ihi</td>
<td>Zero based high row and column index of working submatrix</td>
<td>Long (132)</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
<tr>
<td>info</td>
<td>LAPACK function result info. If info = -i, the i-th argument had an illegal value</td>
<td>Long (132)</td>
</tr>
</tbody>
</table>
ML_DGEBRD – Reduces a general real matrix to bidiagonal form by an orthogonal transformation

Block Symbol

Licence: MATRIX

Function Description

The output references yA, yD, yE, yTAUQ, yTAUP and yWORK are always set to the corresponding input references uA, uD, uE, uTAUQ, uTAUP and uWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGEBRD is called internally:

DGEBRD(M, N, uA, LDA, uD, uE, uTAUQ, uTAUP, uWORK, info);

where parameters of DGEBRD are set in the following way:

- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.
- info is return code from the function DGEBRD.

The error flag E is set to on if:

- the reference uA or uD or uE or uTAUQ or uTAUP or uWORK is not defined (i.e. input uA or uD or uE or uTAUQ or uTAUP or uWORK is not connected),
- number of elements of any vector referenced by uD, uTAUQ and uTAUP is less than MINMN, where MINMN is minimum from M and N,
- number of elements of the vector referenced by uE is less than MINMN – 1,
- the call of the function DGEBRD returns error using the function XERBLA, see the return code info and system log.

Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>uD</td>
<td>Diagonal elements of the bidiagonal matrix B</td>
<td>Reference</td>
</tr>
<tr>
<td>uE</td>
<td>Off-diagonal elements of the bidiagonal matrix B</td>
<td>Reference</td>
</tr>
<tr>
<td>uTAUQ</td>
<td>Reference to a vector of scalar factors of the elementary reflectors which represent the orthogonal matrix Q</td>
<td>Reference</td>
</tr>
<tr>
<td>uTAUP</td>
<td>Reference to a vector of scalar factors of the elementary reflectors which represent the orthogonal matrix P</td>
<td>Reference</td>
</tr>
<tr>
<td>uWORK</td>
<td>Input reference to working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>yD</td>
<td>Output reference to D</td>
<td>Reference</td>
</tr>
<tr>
<td>yE</td>
<td>Output reference to E</td>
<td>Reference</td>
</tr>
<tr>
<td>yTAUQ</td>
<td>Output reference to TAUQ</td>
<td>Reference</td>
</tr>
<tr>
<td>yTAUP</td>
<td>Output reference to TAUP</td>
<td>Reference</td>
</tr>
<tr>
<td>yWORK</td>
<td>Output reference to working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
| info     | LAPACK function result info. If info = -i, the i-th argument had an illegal value | Long (132)
ML_DGECN – Estimates the reciprocal of the condition number of a general real matrix

Block Symbol

Function Description

The output references yA, yWORK and yIWORK are always set to the corresponding input references uA, uWORK and uIWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGECN is called internally:

\[ \text{DGECN(sINORM, N, uA, LDA, anorm, rcond, uWORK, uIWORK, info);} \]

where parameters of DGECN are set in the following way:

- If INORM = on then the string sINORM is set to "I" else it is set to "1".
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.
- rcond is returned reciprocal value of the condition number of the matrix referenced by uA.
- info is return code from the function DGECN.

The error flag E is set to on if:

- the reference uA or uWORK or uIWORK is not defined (i.e. input uA or uWORK or uIWORK is not connected),
- the matrix referenced by uA is not square,
- number of elements of the vector referenced by uWORK is less than 4 * N,
- number of elements of the integer vector referenced by uIWORK is less than N,
- the call of the function DGECN returns error using the function XERBLA, see the return code info and system log.

### Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>uWORK</td>
<td>Input reference to working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>uIWORK</td>
<td>Input reference to integer working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>INORM</td>
<td>Use Infinity-norm</td>
<td>Bool</td>
</tr>
<tr>
<td>anorm</td>
<td>Norm of the original matrix A</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>yWORK</td>
<td>Output reference to working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>yIWORK</td>
<td>Output reference to integer working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>rcond</td>
<td>The reciprocal of the condition number of the matrix A</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
<tr>
<td>info</td>
<td>LAPACK function result info. If info = -i, the i-th argument had an illegal value</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>
ML_DGEES – Computes the eigenvalues, the Schur form, and, optionally, the matrix of Schur vectors

Block Symbol

Licence: MATRIX

Function Description

The output references yA, yWR, yWI, yVS, yWORK and yBWORK are always set to the corresponding input references uA, uWR, uWI, uVS, uWORK and uBWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGEES is called internally:

\[
\text{DGEES}(\text{sJOBVS}, \text{sSORT}, \text{SELECT}, \text{N}, \text{uA}, \text{LDA}, \text{sdim}, \text{uWR}, \text{uWI}, \text{uVS}, \text{LDVS}, \text{uWORK}, \text{LWORK}, \text{uBWORK}, \text{info});
\]

where parameters of DGEES are set in the following way:

- If JOBVS = on then the string sJOBVS is set to "V" else it is set to "N".
- If SORT = on then the string sSORT is set to "S" else it is set to "N".
- SELECT is the reference to Boolean eigenvalues sorting function which in this function block returns always true (i.e. on).
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.
- sdim is returned number of eigenvalues for which the function SELECT is true.
- LDVS is the leading dimension of the matrix referenced by uVS.
- LWORK is number of elements in the vector referenced by uWORK.
- info is return code from the function DGEES.

The error flag E is set to on if:

- the reference uA or uWR or uWI or uVS or uWORK or uBWORK is not defined (i.e. input uA or uWR or uWI or uVS or uWORK or uBWORK is not connected),
• the matrix referenced by $uA$ is not square,

• number of elements of any vector referenced by $uWR$, $uWI$ and $uBWORK$ is less than $N$,

• number of columns of the matrix referenced by $uVS$ is not equal to $N$,

• the call of the function DGEES returns error using the function XERBLA, see the return code info and system log.

See LAPACK documentation [?] for more details.

Inputs

<table>
<thead>
<tr>
<th>$uA$</th>
<th>Input reference to matrix A</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$uWR$</td>
<td>Input reference to vector of real parts of eigenvalues</td>
<td>Reference</td>
</tr>
<tr>
<td>$uWI$</td>
<td>Input reference to vector of imaginary parts of eigenvalues</td>
<td>Reference</td>
</tr>
<tr>
<td>$uVS$</td>
<td>Input reference to orthogonal matrix of Schur vectors</td>
<td>Reference</td>
</tr>
<tr>
<td>$uBWORK$</td>
<td>Input reference to working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>$JOBVS$</td>
<td>If true then Schur vectors are computed</td>
<td>Bool</td>
</tr>
<tr>
<td>$SORT$</td>
<td>If true then eigenvalues are sorted</td>
<td>Bool</td>
</tr>
<tr>
<td>$HLD$</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>$yA$</th>
<th>Output reference to matrix A</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$yWR$</td>
<td>Output reference to vector of real parts of eigenvalues</td>
<td>Reference</td>
</tr>
<tr>
<td>$yWI$</td>
<td>Output reference to vector of imaginary parts of eigenvalues</td>
<td>Reference</td>
</tr>
<tr>
<td>$yVS$</td>
<td>Output reference to VS</td>
<td>Reference</td>
</tr>
<tr>
<td>$yBWORK$</td>
<td>Output reference to working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>$sdim$</td>
<td>If SORT then number of eigenvalues for which SELECT is true else 0</td>
<td>Long (132)</td>
</tr>
<tr>
<td>$E$</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
<tr>
<td>$info$</td>
<td>LAPACK function result info. If info = -i, the i-th argument had an illegal value</td>
<td>Long (132)</td>
</tr>
</tbody>
</table>
ML_DGEEV – Computes the eigenvalues and, optionally, the left and/or right eigenvectors

Block Symbol

Licence: MATRIX

Function Description

The output references yA, yWR, yWI, yVL, yVR and yWORK are always set to the corresponding input references uA, uWR, uWI, uVL, uVR and uWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGEEV is called internally:

```
DGEEV(sJOBVL, sJOBVR, N, uA, LDA, uWR, uWI, uVL, LDVL, uVR, LDVR, uWORK, LWORK, info);
```

where parameters of DGEEV are set in the following way:

- If JOBVL = on then the string sJOBVL is set to "V" else it is set to "N".
- If JOBVR = on then the string sJOBVR is set to "V" else it is set to "N".
- N is number of columns of the matrix referenced by uA.
- LDA, LDVL and LDVR are leading dimensions of the matrices referenced by uA, uVL and uVR.
- LWORK is number of elements of the vector referenced by uWORK.
- info is return code from the function DGEEV.

The error flag E is set to on if:

- the reference uA or uWR or uWI or uVL or uVR or uWORK is not defined (i.e. input uA or uWR or uWI or uVL or uVR or uWORK is not connected),
- the matrix referenced by uA is not square,
- number of elements of vectors referenced by uWR or uWI is less than N,
- number of columns of matrices referenced by uVL or uVR is not equal to N,
- the call of the function DGEEV returns error using the function XERBLA, see the return code `info` and system log.

See LAPACK documentation [?] for more details.

### Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uA</code></td>
<td>Input reference to matrix A</td>
<td></td>
</tr>
<tr>
<td><code>uWR</code></td>
<td>Input reference to vector of real parts of eigenvalues</td>
<td></td>
</tr>
<tr>
<td><code>uWI</code></td>
<td>Input reference to vector of imaginary parts of eigenvalues</td>
<td></td>
</tr>
<tr>
<td><code>uVL</code></td>
<td>Input reference to matrix of left eigenvectors</td>
<td></td>
</tr>
<tr>
<td><code>uVR</code></td>
<td>Input reference to matrix of right eigenvectors</td>
<td></td>
</tr>
<tr>
<td><code>uWORK</code></td>
<td>Input reference to working vector WORK</td>
<td></td>
</tr>
<tr>
<td><code>JOBVL</code></td>
<td>If true then left eigenvectors are computed</td>
<td></td>
</tr>
<tr>
<td><code>JOBVR</code></td>
<td>If true then right eigenvectors are computed</td>
<td></td>
</tr>
<tr>
<td><code>HLD</code></td>
<td>Hold</td>
<td></td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>yA</code></td>
<td>Output reference to matrix A</td>
<td></td>
</tr>
<tr>
<td><code>yWR</code></td>
<td>Output reference to vector of real parts of eigenvalues</td>
<td></td>
</tr>
<tr>
<td><code>yWI</code></td>
<td>Output reference to vector of imaginary parts of eigenvalues</td>
<td></td>
</tr>
<tr>
<td><code>yVL</code></td>
<td>Output reference to VL</td>
<td></td>
</tr>
<tr>
<td><code>yVR</code></td>
<td>Output reference to VR</td>
<td></td>
</tr>
<tr>
<td><code>yWORK</code></td>
<td>Output reference to working vector WORK</td>
<td></td>
</tr>
<tr>
<td><code>E</code></td>
<td>Error indicator</td>
<td></td>
</tr>
<tr>
<td><code>info</code></td>
<td>LAPACK function result info. If info = -i, the i-th argument had an illegal value</td>
<td>Long (132)</td>
</tr>
</tbody>
</table>

`Reference`
**ML_DGEHRD** – Reduces a real general matrix A to upper Hessenberg form

**Block Symbol**

```
  yA       yTAU       yWORK
  uA       uTAU       uWORK
  ihi      lo        info
  HLD      E
```

**Function Description**

The output references yA, yTAU and yWORK are always set to the corresponding input references uA, uTAU and uWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGEHRD is called internally:

```
DGEHRD(N, ilo, IHI, uA, LDA, uTAU, uWORK, LWORK, info);
```

where parameters of DGEHRD are set in the following way:

- **N** is number of columns of the square matrix referenced by uA.
- If the input ihi ≠ 0 then IHI is set to ihi else IHI is set to N – 1.
- **LDA** is the leading dimension of the matrix referenced by uA.
- **LWORK** is number of elements of the vector referenced by uWORK.
- **info** is return code from the function DGEHRD.

The error flag E is set to on if:

- the reference uA or uTAU or uWORK is not defined (i.e. input uA or uTAU or uWORK is not connected),
- matrix referenced by uA is not square,
- number of elements of the vector referenced by uTAU is less than N – 1.
- the call of the function DGEHRD returns error using the function XERBLA, see the return code info and system log.

Emphasize that the indices ilo and ihi start from zero unlike FORTRAN version where they start from one. See LAPACK documentation [7] for more details.
Inputs

\begin{itemize}
  \item \texttt{uA} \quad Input reference to matrix A
  \item \texttt{uTAU} \quad Input reference to vector of scalar factors of the elementary reflectors
  \item \texttt{uWORK} \quad Input reference to working vector WORK
  \item \texttt{ilo} \quad Zero based low row and column index of working submatrix
  \item \texttt{ihi} \quad Zero based high row and column index of working submatrix
  \item \texttt{HLD} \quad Hold
\end{itemize}

Outputs

\begin{itemize}
  \item \texttt{yA} \quad Output reference to matrix A
  \item \texttt{yTAU} \quad Output reference to vector of scalar factors of the elementary reflectors
  \item \texttt{yWORK} \quad Output reference to working vector WORK
  \item \texttt{E} \quad Error indicator
  \item \texttt{info} \quad LAPACK function result info. If info = -i, the i-th argument had an illegal value
\end{itemize}
ML_DGELQF - Computes an LQ factorization of a real M-by-N matrix A

Block Symbol

Function Description
The output references yA, yTAU and yWORK are always set to the corresponding input references uA, uTAU and uWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGELQF is called internally:

DGELQF(M, N, uA, LDA, uTAU, uWORK, LWORK, info);

where parameters of DGELQF are set in the following way:

- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.
- LWORK is number of elements of the vector referenced by uWORK.
- info is return code from the function DGELQF.

The error flag E is set to on if:

- the reference uA or uTAU or uWORK is not defined (i.e. input uA or uTAU or uWORK is not connected),
- number of elements of the vector referenced by uTAU is less than the minimum of number of rows and number of columns of the matrix referenced by uA.
- the call of the function DGELQF returns error using the function XERBLA, see the return code info and system log.


Inputs

| uA | Input reference to matrix A | Reference |
Inputs

- **uTAU**: Input reference to vector of scalar factors of the elementary reflectors
- **uWORK**: Input reference to working vector WORK
- **HLD**: Hold

Outputs

- **yA**: Output reference to matrix A
- **yTAU**: Output reference to vector of scalar factors of the elementary reflectors
- **yWORK**: Output reference to working vector WORK
- **E**: Error indicator
- **info**: LAPACK function result info. If info = -i, the i-th argument had an illegal value

*References*

- **Reference**
- **Bool**

*Data Types*

- **Long (132)**
ML_DGELSD  – Computes the minimum-norm solution to a real linear least squares problem

Block Symbol

Licence: MATRIX

Function Description

The output references yA, yB, yS, yWORK and yIWORK are always set to the corresponding input references uA, uB, uS, uWORK and uIWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGELSD is called internally:

\[
\text{DGELSD}(M, N, NRHS, uA, LDA, uB, LDB, uS, rcond, irank, uWORK, LWORK, uIWORK, info);
\]

where parameters of DGELSD are set in the following way:

- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- NRHS is number of columns of the matrix referenced by uB.
- LDA and LDB are leading dimensions of the matrices referenced by uA and uB.
- irank is returned effective rank of the matrix referenced by uA.
- LWORK is number of elements in the vector referenced by uWORK.
- info is return code from the function DGELSD.

The error flag E is set to on if:

- the reference uA or uB or uS or uWORK or uIWORK is not defined (i.e. input uA or uB or uS or uWORK or uIWORK is not connected),
- the number of rows of the matrix referenced by uB is not equal to M,
- number of elements of any vector referenced by uS is less than the minimum of M and N,
- number of elements of the integer vector referenced by uWORK is not sufficient (see details in the LAPACK documentation of the function DGELSD),

- the call of the function DGELSD returns error using the function XERBLA, see the return code info and system log.

See LAPACK documentation [?] for more details.

**Inputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>uB</td>
<td>Input reference to matrix B</td>
<td>Reference</td>
</tr>
<tr>
<td>uS</td>
<td>Input reference to vector of singular values</td>
<td>Reference</td>
</tr>
<tr>
<td>uWORK</td>
<td>Input reference to working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>uIWORK</td>
<td>Input reference to integer working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>rcond</td>
<td>Used to determine the effective rank of A</td>
<td>$\geq 0.0$ Double (F64)</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>yB</td>
<td>Output reference to matrix B</td>
<td>Reference</td>
</tr>
<tr>
<td>yS</td>
<td>Output reference to vector of singular values</td>
<td>Reference</td>
</tr>
<tr>
<td>yWORK</td>
<td>Output reference to working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>yIWORK</td>
<td>Output reference to integer working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>irank</td>
<td>Effective rank of A</td>
<td>Long (132)</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
<tr>
<td>info</td>
<td>LAPACK function result info. If info = -i, the i-th argument had an illegal value</td>
<td>Long (132)</td>
</tr>
</tbody>
</table>
ML_DGEQRF – Computes an QR factorization of a real M-by-N matrix A

Block Symbol

Licence: MATRIX

Function Description
The output references yA, yTAU and yWORK are always set to the corresponding input references uA, uTAU and uWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGEQRF is called internally:

\[
\text{DGEQRF}(M, N, \text{uA}, \text{LDA}, \text{uTAU}, \text{uWORK}, \text{LWORK}, \text{info});
\]

where parameters of DGEQRF are set in the following way:

- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.
- LWORK is number of elements of the vector referenced by uWORK.
- info is return code from the function DGEQRF.

The error flag E is set to on if:

- the reference uA or uTAU or uWORK is not defined (i.e. input uA or uTAU or uWORK is not connected),
- number of elements of the vector referenced by uTAU is less than the minimum of number of rows and number of columns of the matrix referenced by uA.
- the call of the function DGEQRF returns error using the function XERBLA, see the return code info and system log.


Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
</tr>
</tbody>
</table>
Inputs

uTAU  
Input reference to vector of scalar factors of the elementary reflectors

uWORK  
Input reference to working vector WORK

HLD  
Hold

Outputs

yA  
Output reference to matrix A

yTAU  
Output reference to vector of scalar factors of the elementary reflectors

yWORK  
Output reference to working vector WORK

E  
Error indicator

info  
LAPACK function result info. If info = -i, the i-th argument had an illegal value

Reference

Reference

Bool

Long (132)
ML_DGESDD – Computes the singular value decomposition (SVD) of a real M-by-N matrix A

Block Symbol

Function Description

The output references yA, yS, yU, yVT, yWORK and yIWORK are always set to the corresponding input references uA, uS, uU, uVT, uWORK and uIWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGESDD is called internally:

DGESDD(sJOBZ, M, N, uA, LDA, uS, uU, LDU, uVT, LDVT, uWORK, LWORK, uIWORK, info);

where parameters of DGESDD are set in the following way:

- Integer input jobz is mapped to the string sJOBZ: \{0, 1\} → "A", \{2\} → "S", \{3\} → "O" and \{4\} → "N".
- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- LDA, LDU and LDVT are leading dimensions of the matrices referenced by uA, uU and uVT.
- LWORK is number of elements of the vector referenced by uWORK.
- info is return code from the function DGESDD.

The error flag E is set to on if:

- the reference uA or uS or uU or uVT or uWORK or uIWORK is not defined (i.e. input uA or uS or uU or uVT or uWORK or uIWORK is not connected),
- number of elements of the vector referenced by uS is less than MINM, the minimum of number of rows and number of columns of the matrix referenced by uA,
- number of elements of the integer vector referenced by uIWORK is less than 8*MINM,
- the call of the function DGESDD returns error using the function XERBLA, see the return code info and system log.

See LAPACK documentation \[?] for more details.

### Inputs

<table>
<thead>
<tr>
<th>uA</th>
<th>Input reference to matrix A</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uS</td>
<td>Input reference to vector of singular values</td>
<td>Reference</td>
</tr>
<tr>
<td>uU</td>
<td>Input reference to matrix containing left singular vectors of A</td>
<td>Reference</td>
</tr>
<tr>
<td>uVT</td>
<td>Input reference to matrix containing right singular vectors of A</td>
<td>Reference</td>
</tr>
<tr>
<td>uWORK</td>
<td>Input reference to working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>uIWORK</td>
<td>Input reference to integer working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>jobz</td>
<td>Specifies options for computing</td>
<td>(0.00E+00) Long (132)</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>yA</th>
<th>Output reference to matrix A</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>yS</td>
<td>Output reference to vector of singular values</td>
<td>Reference</td>
</tr>
<tr>
<td>yU</td>
<td>Output reference to matrix containing left singular vectors of A</td>
<td>Reference</td>
</tr>
<tr>
<td>yVT</td>
<td>Output reference to matrix containing right singular vectors of A</td>
<td>Reference</td>
</tr>
<tr>
<td>yWORK</td>
<td>Output reference to working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>yIWORK</td>
<td>Output reference to integer working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
<tr>
<td>info</td>
<td>LAPACK function result info. If info = -i, the i-th argument had an illegal value</td>
<td>Long (132)</td>
</tr>
</tbody>
</table>
ML_DLACPY – Copies all or part of one matrix to another matrix

Block Symbol

Function Description
The output references yA and yB are always set to the corresponding input references uA and uB. If HLD = on then nothing is computed otherwise the LAPACK function DLACPY is called internally:

\[
\text{DLACPY(sUPLO, } M, N, \text{ uA, LDA, uB, LDA);}
\]

where parameters of DLACPY are set in the following way:

- Integer input uplo is mapped to the string sUPLO: \{0, 1\} → "A", \{2\} → "U" and \{3\} → "L".
- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.

The number of rows of the matrix referenced by uB is set to M and the leading dimension of the matrix referenced by uB is set to LDA.

The error flag E is set to on if:

- the reference uA or uB is not defined (i.e. input uA or uB is not connected),
- the allocated number of elements of the matrix referenced by uA is different from the allocated number of elements of the matrix referenced by uB.


Inputs

<table>
<thead>
<tr>
<th>uA</th>
<th>Input reference to matrix A</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uB</td>
<td>Input reference to matrix B</td>
<td>Reference</td>
</tr>
<tr>
<td>uplo</td>
<td>Part of the matrix to be copied</td>
<td>0.00E+00 Long (I32)</td>
</tr>
<tr>
<td></td>
<td>0 . . . . All</td>
<td>0.00E+00 Long (I32)</td>
</tr>
<tr>
<td></td>
<td>1 . . . . All</td>
<td>0.00E+00 Long (I32)</td>
</tr>
<tr>
<td></td>
<td>2 . . . . Upper</td>
<td>0.00E+00 Long (I32)</td>
</tr>
<tr>
<td></td>
<td>3 . . . . Lower</td>
<td>0.00E+00 Long (I32)</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>
## Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>yB</td>
<td>Output reference to matrix B</td>
<td>Reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
ML_DLANGE – Computes one of the matrix norms of a general matrix

Block Symbol Licence: STANDARD

Function Description
The output references yA and yWORK are always set to the corresponding input references uA and uWORK. If HLD = on then nothing is computed otherwise the LAPACK function DLANGE is called internally:

value = DLANGE(sNORM, M, N, uA, LDA, uWORK);

where parameters of DLACPY are set in the following way:

- Integer input norm is mapped to the string sNORM: \{0, 1\} → "F" (Frobenius norm), \{2\} → "M" (max(abs(A(i,j))), \{3\} → "$1$" (one norm) and \{4\} → "$\infty$" (infinity norm).

- M is number of rows of the matrix referenced by uA.

- N is number of columns of the matrix referenced by uA.

- LDA is the leading dimension of the matrix referenced by uA.

- uWORK is the working vector of dimension LWORK ≥ M. uWORK is used only for infinity norm, otherwise it is not referenced.

The error flag E is set to on if:

- the reference uA is not defined (i.e. input uA is not connected),

- the reference uWORK is not defined for norm = 4 (i.e. input uWORK is not connected).


Use the block MB_DNRM2 for computation of Frobenius norm of a vector.

Inputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>uWORK</td>
<td>Input reference to working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>norm</td>
<td>The selected matrix norm</td>
<td>↓0 ↑4 ○0.00E+00</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Long (132)</td>
</tr>
</tbody>
</table>
## Outputs

<table>
<thead>
<tr>
<th>yA</th>
<th>Output reference to matrix A</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>yWORK</td>
<td>Output reference to working vector WORK</td>
<td>Reference</td>
</tr>
<tr>
<td>value</td>
<td>Return value of the function</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
**ML_DLASET** — Initializes the off-diagonal elements and the diagonal elements of a matrix to given values

Block Symbol

![Block Symbol](image)

**Function Description**

The output reference yA is always set to the corresponding input references uA. If HLD = on then nothing is computed otherwise the LAPACK function DLASET is called internally:

```
DLASET(sUPLO, M, N, alpha, beta, uA, LDA);
```

where parameters of DLACPY are set in the following way:

- Integer input uplo is mapped to the string sUPLO: \{0, 1\} → "A", \{2\} → "U" and \{3\} → "L".
- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.

The error flag E is set to on if:

- the reference uA is not defined (i.e. input uA is not connected),


**Inputs**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>0.00E+00 Long (I32)</td>
</tr>
<tr>
<td>uplo</td>
<td>Part of the matrix to be set</td>
<td>0 ...... All 1 ...... All 2 ...... Upper 3 ...... Lower</td>
</tr>
<tr>
<td>alpha</td>
<td>Scalar coefficient alpha</td>
<td>0.0 Double (F64)</td>
</tr>
<tr>
<td>beta</td>
<td>Scalar coefficient beta</td>
<td>0.0 Double (F64)</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>yA</th>
<th>Output reference to matrix A</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
ML_DTRSYL – Solves the real Sylvester matrix equation for quasi-triangular matrices A and B

Block Symbol

<table>
<thead>
<tr>
<th>uA</th>
<th>yA</th>
</tr>
</thead>
<tbody>
<tr>
<td>uB</td>
<td>yB</td>
</tr>
<tr>
<td>uC</td>
<td>yC</td>
</tr>
<tr>
<td>tran</td>
<td>yC</td>
</tr>
<tr>
<td>tran</td>
<td>scale</td>
</tr>
<tr>
<td>align</td>
<td>info</td>
</tr>
<tr>
<td>ML_DTRSYL</td>
<td></td>
</tr>
</tbody>
</table>

Licence: MATRIX

Function Description

The output references yA, yB and yC are always set to the corresponding input references uA, uB and uC. If HLD = on then nothing is computed otherwise the LAPACK function DTRSYL is called internally:

\[
\text{DTRSYL} (\text{strana}, \text{stranb}, M, N, uA, LDA, uB, LDB, uC, LDC, \text{scale}, \text{info})
\]

where parameters of DTRSYL are set in the following way:

- Integer inputs \( \text{trana} \) and \( \text{tranb} \) are mapped to strings \( \text{strana} \) and \( \text{stranb} \): \( \{0, 1\} \rightarrow "N" \), \( \{2\} \rightarrow "T" \) and \( \{3\} \rightarrow "C" \).
- \( M \) is number of rows of the matrix referenced by \( uA \).
- \( N \) is number of columns of the matrix referenced by \( uB \).
- \( LDA, LDB \) and \( LDC \) are leading dimensions of matrices referenced by \( uA, uB \) and \( uC \).
- \( \text{scale} \) is returned scaling factor to avoid overflow.
- \( \text{info} \) is return code from the function DTRSYL.

The error flag \( E \) is set to on if:

- the reference \( uA \) or \( uB \) or \( uC \) is not defined (i.e. input \( uA \) or \( uB \) or \( uC \) is not connected),
- \( \text{trana} \) or \( \text{tranb} \) is less than 0 or greater than 3
- number of columns of the matrix referenced by \( uA \) is not equal to \( M \)
- number of rows of the matrix referenced by \( uB \) is not equal to \( N \)
- number of rows of the matrix referenced by \( uC \) is not equal to \( N \) or number of columns of this matrix is not equal to \( M \),
• the call of the function DTRSYL returns error using the function XERBLA, see the system log.

See LAPACK documentation [?] for more details.

Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>uB</td>
<td>Input reference to matrix B</td>
<td>Reference</td>
</tr>
<tr>
<td>uC</td>
<td>Input reference to matrix C</td>
<td>Reference</td>
</tr>
<tr>
<td>tranA</td>
<td>Transposition of matrix A</td>
<td>( \downarrow 0 \uparrow 3 \circ 0.00E+00 ) Long (I32)</td>
</tr>
<tr>
<td>tranB</td>
<td>Transposition of matrix B</td>
<td>( \downarrow 0 \uparrow 3 \circ 0.00E+00 ) Long (I32)</td>
</tr>
<tr>
<td>isgn</td>
<td>Sign in the equation (1 or -1)</td>
<td>( \downarrow -1 \uparrow 1 \circ 0.00E+00 ) Long (I32)</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>yB</td>
<td>Output reference to matrix B</td>
<td>Reference</td>
</tr>
<tr>
<td>yC</td>
<td>Output reference to matrix C</td>
<td>Reference</td>
</tr>
<tr>
<td>scale</td>
<td>Scale</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
<tr>
<td>info</td>
<td>LAPACK function result info. If info = -i, the i-th argument had an illegal value</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>
MX_AT – Get Matrix/Vector element

Block Symbol

Function Description

The function block MX_AT returns the value (output \texttt{value}) of the matrix element at the i-th row and j-th column or the i-th vector element.

The output reference \texttt{yMV} is always set to the corresponding input reference \texttt{uMV} to the connected matrix/vector.

The error flag \texttt{E} is set to \texttt{on} if:

- the reference \texttt{uMV} is not defined (i.e. input \texttt{uMV} is not connected),
- the zero based row index $i < 0$ or $i \geq m$, where $m$ is the number of rows,
- the zero based column index $j < 0$ or $j \geq n$, where $n$ is the number of columns.

Note that $j$ must be 0 for a vector.

Inputs

\begin{itemize}
\item \texttt{uMV} \quad \text{Input reference to a matrix or vector}
\item \texttt{i} \quad \text{Row index of the element} \quad \text{Reference} \quad \texttt{\_0 \ 0.0E+00} \quad \text{Long (32)}
\item \texttt{j} \quad \text{Column index of the element} \quad \text{Reference} \quad \texttt{\_0 \ 0.0E+00} \quad \text{Long (32)}
\end{itemize}

Outputs

\begin{itemize}
\item \texttt{yMV} \quad \text{Output reference to a matrix or vector}
\item \texttt{value} \quad \text{Value of element at position \{(i,j)\}} \quad \text{Reference} \quad \text{Long (32)}
\item \texttt{E} \quad \text{Error indicator} \quad \text{Reference} \quad \text{Bool}
\end{itemize}
**MX_ATSET – Set Matrix/Vector element**

**Block Symbol**

![Block Symbol](image)

**Licence:** STANDARD

**Function Description**

The function block `MX_ATSET` sets the value (input `value`) to the matrix element at the i-th row and j-th column or to the i-th vector element.

The output reference `yMV` is always set to the corresponding input reference `uMV` to the connected matrix/vector.

The error flag `E` is set to **on** if:

- the reference `uMV` is not defined (i.e. input `uMV` is not connected),
- the zero based row index `i < 0` or `i ≥ m`, where `m` is the number of rows,
- the zero based column index `j < 0` or `j ≥ n`, where `n` is the number of columns.

Note that `j` must be 0 for a vector.

**Inputs**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uMV</code></td>
<td>Input reference to a matrix or vector</td>
<td>Reference</td>
</tr>
<tr>
<td><code>i</code></td>
<td>Row index of the element</td>
<td>↓0 oo 0.00E+00</td>
</tr>
<tr>
<td><code>j</code></td>
<td>Column index of the element</td>
<td>↓0 oo 0.00E+00</td>
</tr>
<tr>
<td><code>value</code></td>
<td>Value which should be set to the element at position (i,j)</td>
<td>Long (132)</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>yMV</code></td>
<td>Output reference to a matrix or vector</td>
<td>Reference</td>
</tr>
<tr>
<td><code>E</code></td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
MX_CNADD – Add scalar to each Matrix/Vector element

Block Symbol

Licence: STANDARD

Function Description

The function block MX_CNADD adds the value of the input alpha to each matrix/vector element referenced by uAX and the result is stored to the matrix/vector referenced by uBY. If HLD = on then nothing is computed.

The output references yAX and yBY are always set to the corresponding input references uAX and uBY. The dimensions of the matrix/vector referenced by uBY are set to the dimensions of the matrix/vector referenced by uAX if they are different.

The error flag E is set to on if:

- the reference uAX of uBY is not defined (i.e. input uAX or uBY is not connected),
- the count of allocated elements of the matrix/vector referenced by uAX is different from the count of allocated elements of the matrix/vector referenced by uBY.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAX</td>
<td>Input reference to the matrix A or vector X</td>
<td>Reference</td>
</tr>
<tr>
<td>uBY</td>
<td>Input reference to the matrix B or vector Y</td>
<td>Reference</td>
</tr>
<tr>
<td>alpha</td>
<td>Scalar coefficient alpha</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAX</td>
<td>Output reference to the matrix A or vector X</td>
<td>Reference</td>
</tr>
<tr>
<td>yBY</td>
<td>Output reference to the matrix B or vector Y</td>
<td>Reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
**MX_CNMUL – Multiply a Matrix/Vector by a scalar**

**Block Symbol**

<table>
<thead>
<tr>
<th>uAX</th>
<th>yAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>uBY</td>
<td>yBY</td>
</tr>
<tr>
<td>alpha</td>
<td>HLD</td>
</tr>
<tr>
<td>E</td>
<td>MX_CNMUL</td>
</tr>
</tbody>
</table>

**Licence:** STANDARD

**Function Description**

The function block MX_CNMUL multiplies each matrix/vector element referenced by uAX by the value of the input alpha and the result is stored to the matrix/vector referenced by uBY. If HLD = on then nothing is computed.

The output references yAX and yBY are always set to the corresponding input references uAX and uBY. The dimensions of the matrix/vector referenced by uBY are set to the dimensions of the matrix/vector referenced by uAX if they are different.

The error flag E is set to on if:

- the reference uAX of uBY is not defined (i.e. input uAX or uBY is not connected),
- the count of allocated elements of the matrix/vector referenced by uAX is different from the count of allocated elements of the matrix/vector referenced by uBY.

**Inputs**

- **uAX**: Input reference to the matrix A or vector X
- **uBY**: Input reference to the matrix B or vector Y
- **alpha**: Scalar coefficient alpha
- **HLD**: Hold

**Outputs**

- **yAX**: Output reference to the matrix A or vector X
- **yBY**: Output reference to the matrix B or vector Y
- **E**: Error indicator

**Reference**

Double (F64), Bool
MX_CTODPA – Discretizes continuous model given by (A,B) to (Ad,Bd) using Pade approximations

Block Symbol

Function Description

This function block discretizes a continuous state space model using Padé approximations of matrix exponential and its integral and scaling technique ([8]). The used technique is similar to method 3 Scaling and squaring described in [8].

The output references $y_A$, $y_B$, $y_{Ad}$, $y_{Bd}$, $y_P$, $y_Q$ and $y_R$ are always set to the corresponding input references $u_A$, $u_B$, $u_{Ad}$, $u_{Bd}$, $u_P$, $u_Q$ and $u_R$. If $HLD = on$ then nothing is computed otherwise the function mCtoD is called internally:

$$ mCtoD(nRes, u_{Ad}, u_{Bd}, u_A, u_B, N, M, is, Ts, eps, u_P, u_Q, u_R); $$

where parameters of mCtoD are set in the following way:

- $nRes$ is return code from the function mCtoD.
- $N$ is number of rows of the square system matrix referenced by $u_A$.
- $M$ is number of columns of the input matrix referenced by $u_B$.
- $Ts$ is sampling period for the discretization, which is equal to sampling period of the task containing this function block.

The error flag $E$ is set to on if:

- the reference $u_A$ or $u_B$ or $u_{Ad}$ or $u_{Bd}$ or $u_P$ or $u_Q$ or $u_R$ is not defined (i.e. input $u_A$ or $u_B$ or $u_{Ad}$ or $u_{Bd}$ or $u_P$ or $u_Q$ or $u_R$ is not connected),
- number of columns of the matrix referenced by $u_A$ is not equal to $N$,
- number of rows of the matrix referenced by $u_B$ is not equal to $N$,
- number of elements of any matrix referenced by $u_{Ad}$, $u_P$, $u_Q$ or $u_R$ is less than $N \times N$,
- number of elements of the matrix referenced by $u_{Bd}$ is less than $N \times M$. 
the return code nRes of the function MCont is not equal to zero.

Inputs

- uA: Input reference to matrix A
- uB: Input reference to matrix B
- uAd: Input reference to discretized matrix A
- uBd: Input reference to discretized matrix B
- uP: Input reference to a helper matrix
- uQ: Input reference to a helper matrix
- uR: Input reference to a helper matrix
- HLD: Hold

Parameters

- is: Pade approximation order
- eps: Approximation accuracy

Outputs

- yA: Output reference to matrix A
- yB: Output reference to matrix B
- yAd: Output reference to discretized matrix A
- yBd: Output reference to discretized matrix B
- yP: Output reference to a helper matrix
- yQ: Output reference to a helper matrix
- yR: Output reference to a helper matrix
- E: Error indicator
**MX_DIM** – Matrix/Vector dimensions

**Block Symbol**

![Block Symbol](image)

**Licence:** STANDARD

**Function Description**

The function block **MX_DIM** sets its outputs to the dimensions of the matrix or vector referenced by **uMV**.

The output reference **yMV** is always set to the corresponding input reference **uMV**. The error flag **E** is set to **on** if the reference **uMV** is not defined (i.e. input **uMV** is not connected).

**Input**

<table>
<thead>
<tr>
<th><strong>Input</strong></th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uMV</strong></td>
<td>Input reference to a matrix or vector</td>
<td></td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th><strong>Output</strong></th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>yMV</strong></td>
<td>Output reference to a matrix or vector</td>
<td></td>
</tr>
<tr>
<td><strong>m</strong></td>
<td>Number of matrix rows</td>
<td>Long (I32)</td>
</tr>
<tr>
<td><strong>n</strong></td>
<td>Number of matrix columns</td>
<td>Long (I32)</td>
</tr>
<tr>
<td><strong>ld</strong></td>
<td>Leading dimension ((\geq) number of rows)</td>
<td>Long (I32)</td>
</tr>
<tr>
<td><strong>cnt</strong></td>
<td>Count of used matrix/vector elements</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>
MX_DIMSET – Set Matrix/Vector dimensions

Block Symbol

Licence: STANDARD

Function Description

The function block MX_DIMSET sets number rows m of the vector or number of rows m, number of columns n and the leading dimension ld of the matrix referenced by uMV. If any of the inputs m, n, ld is not connected, its original value is retained.

The output cnt contains the actual number of occupied elements of the matrix/vector and is determined by the formula

\[ \text{cnt} = \text{ld} \times (n - 1) + m \leq \text{amax} \]

where the output amax is the allocated count of matrix/vector elements. If this inequality is fulfilled the output cnt is set to the matrix/vector structure and can be retrieved by the MX_DIM block, otherwise the value of cnt shows the minimum necessary number of elements of the matrix/vector.

The output reference yMV is always set to the corresponding input reference uMV.

The error flag E is set to on if:

- the reference uMV is not defined (i.e. input uMV is not connected),
- the number of rows m < 1 or m > ld,
- the number of columns n < 1,
- the required number of elements cnt > amax.

Inputs

<table>
<thead>
<tr>
<th></th>
<th>Input reference to a matrix or vector</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uMV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>Number of matrix rows</td>
<td>Long (132)</td>
</tr>
<tr>
<td>ld</td>
<td>Leading dimension ((\geq) number of rows)</td>
<td>Long (132)</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th></th>
<th>Output reference to a matrix or vector</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>yMV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cnt</td>
<td>Count of used matrix/vector elements</td>
<td>Long (132)</td>
</tr>
<tr>
<td>amax</td>
<td>Number of allocated matrix/vector elements</td>
<td>Long (132)</td>
</tr>
</tbody>
</table>
E Error indicator Bool
**MX_DSAGET – Set subarray of A into B**

Block Symbol

**Function Description**

Generally, the function block **MX_DSAGET** copies the subarray (submatrix) of matrix referenced by **uA** into the matrix referenced by **uB**.

The output references **yA** and **yB** are always set to the corresponding input references **uA** and **uB**. If **HLD = on** then nothing is copied otherwise the submatrix of matrix referenced by **uA** starting the row with zero based index **I** and the column with zero based index **J** containing **M** rows and **N** columns is copied (with respect to the value of the input **uplo** to the matrix referenced by **uB**. The mentioned variables have the following meanings:

- If the input \( i \leq 0 \) then \( I \) is set to 0 else if \( i \geq MA \) then \( I \) is set to \( MA - 1 \) else \( I \) is set to \( i \), where \( MA \) is the number of rows of the matrix referenced by **uA**.

- If the input \( j \leq 0 \) then \( J \) is set to 0 else if \( j \geq NA \) then \( J \) is set to \( NA - 1 \) else \( J \) is set to \( j \), where \( NA \) is the number of columns of the matrix referenced by **uA**.

- Number of copied rows \( M \) is set in two stages. First, \( M \) is set to minimum of \( MA - I \) and number of rows of the matrix referenced by **uB**. Second, if \( m > 0 \) then \( M \) is set to the minimum of \( m \) and \( M \).

- Number of copied columns \( N \) is set in two stages. First, \( N \) is set to minimum of \( NA - J \) and number of columns of the matrix referenced by **uB**. Second, if \( n > 0 \) then \( N \) is set to the minimum of \( n \) and \( N \).

The error flag **E** is set to **on** if:

- the reference **uA** or **uB** is not defined (i.e. input **uA** or **uB** is not connected),

- **uplo** is less than 0 or greater than 3,

- the number of elements of the matrix referenced by **uB** is less than \( M \times N \).
# Chapter 14. Matrix - Blocks for Matrix and Vector Operations

## Inputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>uB</td>
<td>Input reference to matrix B</td>
<td>Reference</td>
</tr>
<tr>
<td>uplo</td>
<td>Part of the matrix to be copied</td>
<td>0......All 2......Upper 1......All 3......Lower</td>
</tr>
<tr>
<td>i</td>
<td>Index of the subarray first row</td>
<td>0.0E+00</td>
</tr>
<tr>
<td>j</td>
<td>Index of the subarray first column</td>
<td>0.0E+00</td>
</tr>
<tr>
<td>m</td>
<td>Number of matrix rows</td>
<td>0.0E+00</td>
</tr>
<tr>
<td>n</td>
<td>Number of matrix columns</td>
<td>0.0E+00</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>yB</td>
<td>Output reference to matrix B</td>
<td>Reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
MX_DSAREF – Set reference to subarray of A into B

Block Symbol

Function Description

The function block MX_DSAREF creates a reference yB to the subarray (submatrix) of matrix referenced by uA. This operation is very fast because no matrix element is copied.

The output reference yA is always set to the corresponding input reference uA, the output reference yB is created inside each instance of this function block. If HLD = on then no other operation is performed otherwise the reference to the matrix yB is created with the following properties:

- Number of rows of the submatrix is set to M – i, where M is number of rows of the matrix referenced by uA.
- Number of columns of the submatrix is set to N – j, where N is number of columns of the matrix referenced by uA.
- The first element in position (0, 0) of the submatrix is the element of the matrix referenced by uA in position (i, j), all indices are zero based.
- The matrix referenced by yB has the same leading dimension as the matrix referenced by uA.

The error flag E is set to on if:

- the reference uA is not defined (i.e. input uA is not connected),
- 0 > i ≥ M,
- 0 > j ≥ N.

Inputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>i</td>
<td>Index of the subarray first row</td>
<td>0.00E+00 Long (132)</td>
</tr>
<tr>
<td>j</td>
<td>Index of the subarray first column</td>
<td>0.00E+00 Long (132)</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>
Parameter

ay     Output reference of the subarray  \( [0, 0] \)  Double (F64)

Outputs

yA     Output reference to matrix A  Reference
yB     Output reference to matrix B  Reference
E      Error indicator  Bool
MX_DSASET – Set A into subarray of B

Block Symbol

Licence: STANDARD

Function Description

Generally, the function block MX_DSASET copies the matrix referenced by uA into the subarray (submatrix) of the matrix referenced by uB.

The output references yA and yB are always set to the corresponding input references uA and uB. If HLD = on then nothing is copied otherwise the matrix referenced by uA is copied (with respect to the value of the input uplo) to the submatrix of the matrix referenced by uB to the row with zero based index I and the column with zero based index J containing M rows and N columns. The mentioned variables have the following meanings:

- If the input i ≤ 0 then I is set to 0 else if i ≥ MB then I is set to MB – 1 else I is set to i, where MB is the number of rows of the matrix referenced by uB.

- If the input j ≤ 0 then J is set to 0 else if j ≥ NB then J is set to NB – 1 else J is set to j, where NB is the number of columns of the matrix referenced by uB.

- Number of copied rows M is set in two stages. First, M is set to minimum of MB – I and number of rows of the matrix referenced by uA. Second, if m > 0 then M is set to the minimum of m and M.

- Number of copied columns N is set in two stages. First, N is set to minimum of NB – J and number of columns of the matrix referenced by uA. Second, if n > 0 then N is set to the minimum of n and N.

The error flag E is set to on if:

- the reference uA or uB is not defined (i.e. input uA or uB is not connected),

- uplo is less than 0 or greater than 3,

- the number of elements of the matrix referenced by uB is less than M ∗ N.
416 CHAPTER 14. MATRIX - BLOCKS FOR MATRIX AND VECTOR OPERATIONS

**Inputs**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>uB</td>
<td>Input reference to matrix B</td>
<td>Reference</td>
</tr>
<tr>
<td>uplo</td>
<td>Part of the matrix to be copied</td>
<td>○0.0E+00 Long (I32)</td>
</tr>
<tr>
<td></td>
<td>0 .... All</td>
<td>2 .... Upper</td>
</tr>
<tr>
<td></td>
<td>1 .... All</td>
<td>3 .... Lower</td>
</tr>
<tr>
<td>i</td>
<td>Index of the subarray first row</td>
<td>○0.0E+00 Long (I32)</td>
</tr>
<tr>
<td>j</td>
<td>Index of the subarray first column</td>
<td>○0.0E+00 Long (I32)</td>
</tr>
<tr>
<td>m</td>
<td>Number of matrix rows</td>
<td>○0.0E+00 Long (I32)</td>
</tr>
<tr>
<td>n</td>
<td>Number of matrix columns</td>
<td>○0.0E+00 Long (I32)</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>Reference</td>
</tr>
<tr>
<td>yB</td>
<td>Output reference to matrix B</td>
<td>Reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
MX_DTRNSP – General matrix transposition: B := alpha*A^T

Block Symbol

License: STANDARD

Function Description

The function block MX_DTRNSP stores the scalar multiple of the general (i.e. rectangular) matrix referenced by uA into the matrix referenced by uB.

The output references yA and yB are always set to the corresponding input references uA and uB. If HLD = on then nothing else is done otherwise the BLAS-like function X_DTRNSP is called internally:

X_DTRNSP(M, N, ALPHA, uA, LDA, uB, LDB);

where parameters of X_DTRNSP are set in the following way:

- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- If the input alpha is equal to 0 then ALPHA is set to 1 else ALPHA is set to alpha.
- LDA and LDB are leading dimensions of matrices referenced by uA and uB.

The error flag E is set to on if:

- the reference uA or uB is not defined (i.e. input uA or uB is not connected),
- the call of the function X_DTRNSP returns error using the function XERBLA, see the system log.

Inputs

<table>
<thead>
<tr>
<th>uA</th>
<th>Input reference to matrix A</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uB</td>
<td>Input reference to matrix B</td>
<td>Reference</td>
</tr>
<tr>
<td>alpha</td>
<td>Scalar coefficient alpha</td>
<td>≥0.0</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>yA</th>
<th>Output reference to matrix A</th>
<th>Reference</th>
</tr>
</thead>
</table>
### Chapter 14. Matrix - Blocks for Matrix and Vector Operations

<table>
<thead>
<tr>
<th>yB</th>
<th>Output reference to matrix B</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
**Block Symbol**

**Function Description**

The function block `MX_DTRNSQ` transpose the scalar multiple of the square matrix referenced by `uA` in-place.

The output reference `yA` is always set to the corresponding input references `uA`. If `HLD = on` then nothing else is done otherwise the BLAS-like function `X_DTRNSQ` is called internally:

\[ X\_DTRNSQ(N, \text{ALPHA}, uA, LDA); \]

where parameters of `X_DTRNSQ` are set in the following way:

- `N` is number of rows and columns of the matrix referenced by `uA`.
- If the input `alpha` is equal to 0 then `ALPHA` is set to 1 else `ALPHA` is set to `alpha`.
- `LDA` is the leading dimension of the matrix referenced by `uA`.

The error flag `E` is set to `on` if:

- the reference `uA` is not defined (i.e. input `uA` is not connected),
- the matrix referenced by `uA` is not square,
- the call of the function `X_DTRNSQ` returns error using the function `XERBLA`, see the system log.

**Inputs**

- `uA`: Input reference to matrix A
- `alpha`: Scalar coefficient alpha
- `HLD`: Hold

**Outputs**

- `yA`: Output reference to matrix A
- `E`: Error indicator

**Licence:** STANDARD
**MX_FILL** – Fill real matrix or vector

Block Symbol

License: **STANDARD**

**Function Description**

The function block **MX_FILL** fills elements of the matrix or vector referenced by **uMV** according to the input **mode**.

The output reference **yMV** is always set to the corresponding input references **uMV**. If **HLD = on** then nothing else is done.

The error flag **E** is set to **on** if:

- the reference **uMV** is not defined (i.e. input **uMV** is not connected),
- 0 > **mode** > 4.

**Inputs**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uMV</strong></td>
<td>Input reference to a matrix or vector</td>
<td></td>
</tr>
<tr>
<td><strong>value</strong></td>
<td>Fill value of matrix/vector</td>
<td>0.0</td>
</tr>
<tr>
<td><strong>mode</strong></td>
<td>Fill mode</td>
<td>0.00E+00</td>
</tr>
</tbody>
</table>

- 0, 1 ... Value – All elements are set to **value**
- 2 ....... Ones – All elements are set to 1
- 3 ....... Diagonal value – Diagonal is set to **value**, the other elements to 0
- 4 ....... Diagonal ones – Initializes identity matrix (eye)

**HLD** Hold  Bool

**Outputs**

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>yMV</strong></td>
<td>Output reference to a matrix or vector</td>
<td></td>
</tr>
<tr>
<td><strong>E</strong></td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
MX_MAT – Matrix data storage block

Block Symbol

Licence: STANDARD

Function Description

The function block MX_MAT allocates memory (during the block initialization) for \( m \times n \) elements of the type determined by the parameter etype of the matrix referenced by the output yMat. Also matrix leading dimension can be set by the parameter ld. If \( ld < m \) then the leading dimension is set to \( m \).

Note that the present version of the MATRIX function block set supports only matrices with the etype equal to 8: Double.

Parameters

\[
\begin{align*}
\text{m} & \quad \text{Number of matrix rows} & \quad 1 \downarrow 1000000000 \circ1.00E+01 & \quad \text{Long (I32)} \\
\text{n} & \quad \text{Number of matrix columns} & \quad 1 \uparrow 1000000000 \circ1.00E+01 & \quad \text{Long (I32)} \\
\text{ld} & \quad \text{Leading dimension (\( \geq \) number of rows)} & \quad \downarrow 0 \uparrow 1000000000 \circ0.00E+00 & \quad \text{Long (I32)} \\
\text{etype} & \quad \text{Type of elements} & \quad 8 \circ8.00E+00 & \quad \text{Long (I32)} \\
1 & \quad \text{Bool} & \quad 6 & \quad \text{DWord (U32)} \\
2 & \quad \text{Byte (U8)} & \quad 7 & \quad \text{Float (F32)} \\
3 & \quad \text{Short (I16)} & \quad 8 & \quad \text{Double (F64)} \\
4 & \quad \text{Long (I32)} & \quad \circ & \quad \text{Double (F64)} \\
5 & \quad \text{Word (U16)} & \quad 10 & \quad \text{Large (I64)} \\
\end{align*}
\]

Output

\[
\begin{align*}
\text{yMat} & \quad \text{Output reference to a matrix} \\
\end{align*}
\]
CHAPTER 14. MATRIX – BLOCKS FOR MATRIX AND VECTOR OPERATIONS

**MX_RAND – Randomly generated matrix or vector**

Block Symbol

Licence: **STANDARD**

**Function Description**

The function block **MX_RAND** generates random elements of the matrix or vector referenced by **uMV**.

The output reference **yMV** is always set to the corresponding input references **uMV**. If **HLD** = on then nothing is generated otherwise pseudo-random values of the matrix or vector elements referenced by **uMV** are generated using these rules:

- If the parameter **BIP** is on then the generated elements are inside the interval \([-\text{scale}; \text{scale}]\) else they are inside the interval \([0; \text{scale}]\).

- Elements are internally generated using the standard C language function **rand()** which generates pseudo-random numbers in the range from 0 to **RAND_MAX**. Note, that the value of **RAND_MAX** can be platform dependent (and it should be at least 32767).

- The rising edge on the input **SET** causes that the standard C language function **srand(nseed)** (initializes the pseudo-random generator with the value of **nseed**) is called before the generation of random elements. The same sequences of pseudo-random numbers are generated after calls of **srand(nseed)** for the same values of **nseed**.

The error flag **E** is set to on if the reference **uMV** is not defined (i.e. input **uMV** is not connected).

**Inputs**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uMV</strong></td>
<td>Input reference to a matrix or vector</td>
<td></td>
</tr>
<tr>
<td><strong>nseed</strong></td>
<td>Random number seed</td>
<td>(0.00E+00) Long (I32)</td>
</tr>
<tr>
<td><strong>SET</strong></td>
<td>Set initial value of random number generator to <strong>nseed</strong> on rising edge</td>
<td>Bool</td>
</tr>
<tr>
<td><strong>HLD</strong></td>
<td>Hold</td>
<td>Bool</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BIP</strong></td>
<td>Bipolar random values flag</td>
<td>Bool</td>
</tr>
</tbody>
</table>
scale : Random values multiplication factor
\[ \odot 1.0 \quad \text{Double (F64)} \]

**Outputs**

- \( y_{MV} \) : Output reference to a matrix or vector
- \( E \) : Error indicator
  
  Reference
  
  Bool
**MX_REFCOPY** – Copies input references of matrices A and B to their output references

<table>
<thead>
<tr>
<th>Block Symbol</th>
<th>Licence: <strong>STANDARD</strong></th>
</tr>
</thead>
</table>

**Function Description**

The function block **MX_REFCOPY** is an administrative block of the **MATRIX** blockset. It does nothing else than copying the input references **uA** and **uB** to the corresponding output references **yA** and **yB**.

But suitable insertion of this block to the function block scheme can substantially influence (change) the execution order of blocks which can be very advantageous especially in combination with such blocks as e.g. **MX_DSAREF**.

**Inputs**

<table>
<thead>
<tr>
<th><strong>uA</strong></th>
<th>Input reference to matrix A</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uB</strong></td>
<td>Input reference to matrix B</td>
<td>Reference</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th><strong>yA</strong></th>
<th>Output reference to matrix A</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>yB</strong></td>
<td>Output reference to matrix B</td>
<td>Reference</td>
</tr>
</tbody>
</table>
MX_SLFS – Save or load a Matrix/Vector into file or string

Block Symbol

Function Description

The block allows to convert a matrix or vector into text form and vice versa. The matrix is supplied as a reference to the uMV input. The yMV output refers to the same matrix as the uMV input, and is intended to chain matrix blocks in the correct order, as is common with all MATRIX blocks. The text can be either in the input uStr (or output yStr for the opposite direction of conversion) or in the file. If the text is in a file, its name is the string connected to the uStr input. The usual REXGEN system file name rules applies, i.e. it is relative to datadir and ./ is not allowed to leave the directory. If the uStr input is unattached (or empty string), the path name of the file is used with the full path (that is, including the task name and all subsystems) with the .dat extension.

The format of a matrix in a text file or in text input and output is determined by the format parameter. Supported English and Czech CSV (i.e., columns separated by comma or semicolon), JSON format (created by Google and often used in web applications) and the format used by MATLAB (for entering a matrix in MATLAB scripts).

Conversion from text to matrix/vector or vice versa can be performed at each step of the algorithm or is triggered by the LOAD and SAVE inputs. The exact method is determined by the mode parameter and is explained in detail in the description of this parameter. If an error occurs, it is signaled to the iE output and in the log. After a fatal error, the conversion from/to the matrix stops. Error reset for mode = 1..4 is done by setting LOAD = SAVE = off, resetting fatal error cannot be performed for mode = 5..8 (must switch to mode = 1..4 and then back).

The nmax parameter is used to allocate the output string. If nmax > 0, it is allocated specified number of chars during initialization. If this amount is insufficient, the block reports an error. If nmax = 0, the block increases the length of the output string as needed. If user don’t specify the nmax parameter it can lead to full RAM memory in extreme situations and unpredictable behaviour of entire system.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uMV</td>
<td>Input reference to a matrix or vector</td>
<td></td>
</tr>
<tr>
<td>uStr</td>
<td>Input string (to convert into matrix)</td>
<td>String</td>
</tr>
<tr>
<td>LOAD</td>
<td>Trigger to move data to matrix/vector</td>
<td>Bool</td>
</tr>
<tr>
<td>SAVE</td>
<td>Trigger to move data from matrix/vector</td>
<td>Bool</td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>mode</th>
<th>Triggering mode</th>
<th>( \odot 2 )</th>
<th>Long (I32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>level-triggered file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>edge-triggered file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>level-triggered string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>edge-triggered string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>continuous string to matrix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>continuous matrix to string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>continuous file to matrix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>continuous matrix to file</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>format</th>
<th>String/file format</th>
<th>( \odot 1 )</th>
<th>Long (I32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CSV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CSV(semicolon)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>JSON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>MATLAB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>prec</th>
<th>Number of digits for single value</th>
<th>( \downarrow 0 ) ( \uparrow 20 )</th>
<th>( \odot 6 )</th>
<th>Long (I32)</th>
</tr>
</thead>
</table>

| TRN    | Transposition flag                                   | Bool        |            |

| nmax   | Allocated size of string                             | \( \downarrow 0 \) | Long (I32) |

### Outputs

<table>
<thead>
<tr>
<th>yMV</th>
<th>Output reference to a matrix or vector</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>yStr</td>
<td>String representation of the matrix/vector</td>
<td>String</td>
</tr>
<tr>
<td>iE</td>
<td>Error code</td>
<td>Error</td>
</tr>
</tbody>
</table>
**MX_VEC – Vector data storage block**

**Block Symbol**

![MX_VEC](image)

**Licence:** STANDARD

**Function Description**

The function block **MX_VEC** allocates memory (during the block initialization) for \( n \) elements of the type determined by the parameter **etype** of the vector referenced by the output \( yVec \).

Note that the present version of the **MATRIX** function block set supports only vectors with the **etype** equal to 8: Double.

**Parameters**

\[
\begin{array}{|c|c|c|c|}
\hline
\text{n} & \text{Number of vector elements} & \downarrow\text{|1000000000 |0.00E+01} & \text{Long (I32)} \\
\text{etype} & \text{Type of elements} & \circ\text{|8.00E+00} & \text{Long (I32)} \\
\hline
1 & \text{Bool} & 6 & \text{DWord (U32)} \\
2 & \text{Byte (U8)} & 7 & \text{Float (F32)} \\
3 & \text{Short (I16)} & 8 & \text{Double (F64)} \\
4 & \text{Long (I32)} & \text{--} & \text{--} \\
5 & \text{Word (U16)} & 10 & \text{Large (I64)} \\
\hline
\end{array}
\]

**Output**

\[yVec\] Output reference to a vector

**Reference**
428 CHAPTER 14. MATRIX – BLOCKS FOR MATRIX AND VECTOR OPERATIONS

**MX_WRITE – Write a Matrix/Vector to the console/system log**

Block Symbol

![MX_WRITE Block Symbol](image)

Licence: STANDARD

**Function Description**

This function block can write a vector or matrix to the console or the system log. The severity of the console/system log output is set by the parameter **mode** in combination with settings of system log from REXYGEN Studio, menu Target/Configure System Log. Written data can be viewed in REXYGEN Studio, after opening the system log window by the command Target/Show System Log. The function block is very useful for debugging purposes of matrix/vector algorithms.

The output references **yMV** is always set to the input reference **uMV**. If **RUN = off** then nothing else is done otherwise matrix or vector is written to the system log if the configured target logging level for function blocks contains the configured **mode**. Format of each matrix/vector element is determined by parameters **mchars** and **mdec**. The error flag **E** is set to on if:

- the reference **uMV** is not defined (i.e. input **uMV** is not connected),
- \(3 > \text{mchars} > 25\),
- \(0 > \text{mdec} > \text{mchars} - 2\).

**Inputs**

- **uMV**: Input reference to a matrix or vector
- **RUN**: Enable execution

**Parameters**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mode</strong></td>
<td>Severity mode of writing</td>
<td>(\text{Long (I32)})</td>
</tr>
<tr>
<td><strong>mchars</strong></td>
<td>Number of characters per single element</td>
<td>(\uparrow 3 \downarrow 25 \circ 8.00\text{E+00}) (\text{Long (I32)})</td>
</tr>
<tr>
<td><strong>mdec</strong></td>
<td>Number of decimal digits per single element</td>
<td>(\downarrow 0 \uparrow 23 \circ 4.00\text{E+00}) (\text{Long (I32)})</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>y\text{MV}</td>
<td>Output reference to a matrix or vector</td>
<td>Reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
</tbody>
</table>
CHAPTER 14. MATRIX – BLOCKS FOR MATRIX AND VECTOR OPERATIONS

RTOV – Vector multiplexer

Block Symbol

Licence: STANDARD

Function Description

The RTOV block can be used to create vector signals in the REXYGEN system. It combines
the scalar input signals into one vector output signal.

It is also possible to chain the RTOV blocks to create signals with more than 8 items.

The nmax parameter defines the maximal number of items in the vector (in other
words, the size of memory allocated for the signal). The offset parameter defines the
position of the first input signal u1 in the resulting signal. Only the first n input signals
are combined into the resulting yVec vector signal.

Inputs

<table>
<thead>
<tr>
<th>uVec</th>
<th>Vector signal</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>Analog input of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>u2</td>
<td>Analog input of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>u3</td>
<td>Analog input of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>u4</td>
<td>Analog input of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>u5</td>
<td>Analog input of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>u6</td>
<td>Analog input of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>u7</td>
<td>Analog input of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>u8</td>
<td>Analog input of the block</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

Parameters

| nmax      | Allocated size of vector    | 10 ≥ 8     | Long (I32) |
| offset    | Index of the first input in vector | 10 | Long (I32) |
| n         | Number of valid inputs      | 10 ≥ 8 ≥ 8 | Long (I32) |

Output

<table>
<thead>
<tr>
<th>yVec</th>
<th>Vector signal</th>
<th>Reference</th>
</tr>
</thead>
</table>
SWVMR – Vector/matrix/reference signal switch

Block Symbol

Function Description

The SWVMR allows switching of vector or matrix signals. It also allow switching of motion axes in motion control algorithms (see the RM_Axis block).

Use the SSW block or its alternatives SwR and SELU for switching simple signals.

Inputs

<table>
<thead>
<tr>
<th>uRef0</th>
<th>Vector signal</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uRef1</td>
<td>Vector signal</td>
<td>Reference</td>
</tr>
<tr>
<td>uRef2</td>
<td>Vector signal</td>
<td>Reference</td>
</tr>
<tr>
<td>uRef3</td>
<td>Vector signal</td>
<td>Reference</td>
</tr>
<tr>
<td>uRef4</td>
<td>Vector signal</td>
<td>Reference</td>
</tr>
<tr>
<td>uRef5</td>
<td>Vector signal</td>
<td>Reference</td>
</tr>
<tr>
<td>uRef6</td>
<td>Vector signal</td>
<td>Reference</td>
</tr>
<tr>
<td>uRef7</td>
<td>Vector signal</td>
<td>Reference</td>
</tr>
<tr>
<td>iSW</td>
<td>Active signal selector</td>
<td>○0.0E+00 Long (132)</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>yRef</th>
<th>Vector signal</th>
<th>Reference</th>
</tr>
</thead>
</table>
VTOR – Vector demultiplexer

Block Symbol

Function Description
The VTOR block splits the input vector signal into individual signals. The user defines the starting item and the number of items to feed to the output signals using the offset and N parameter respectively.

Input

<table>
<thead>
<tr>
<th>uVec</th>
<th>Vector signal</th>
</tr>
</thead>
</table>

Parameters

<table>
<thead>
<tr>
<th>n</th>
<th>Number of valid outputs</th>
<th>↓0 ↑8 ⊕8</th>
<th>Long (I32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>Index of the first output</td>
<td>↓0</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>y1</th>
<th>Analog output of the block</th>
<th>Double (F64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>y2</td>
<td>Analog output of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>y3</td>
<td>Analog output of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>y4</td>
<td>Analog output of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>y5</td>
<td>Analog output of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>y6</td>
<td>Analog output of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>y7</td>
<td>Analog output of the block</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>y8</td>
<td>Analog output of the block</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
Chapter 15

SPEC – Special blocks

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPC – External program call</td>
<td>434</td>
</tr>
<tr>
<td>HTTP – HTTP GET or POST request (obsolete)</td>
<td>437</td>
</tr>
<tr>
<td>HTTP2 – Block for generating HTTP GET or POST requests</td>
<td>439</td>
</tr>
<tr>
<td>SMTP – Send e-mail message via SMTP</td>
<td>441</td>
</tr>
<tr>
<td>RDC – Remote data connection</td>
<td>443</td>
</tr>
<tr>
<td>REXLANG – User programmable block</td>
<td>448</td>
</tr>
</tbody>
</table>
EPC – External program call

Block Symbol

Function Description
The EPC block executes an external program upon a rising edge (off→on) occurring at the EXEC input. The name and options of the program are defined by the cmd parameter. The format is the same as if the program was executed from the command line of the operating system.

It is possible to pass data from the REXYGEN system to the external program via files. The formatting of the files is defined by the format parameter. All the currently supported formats are textual and simple, which allows straightforward processing of the data in arbitrary program. Use e.g.

```
values=load('*.ASCII', 'epc_inputVec1');
```
for loading the data in MATLAB or

```
values=read('epc_inputVec1','-1,32);
```
in SCILAB. The filename and number of columns must be adjusted for the given project. Data exchange in the opposite direction is naturally also supported, the REXYGEN system can read the files in the same format.

The block works in two modes. In basic mode, the rising edge on the EXEC input triggers reading the data on inputs and storing them in the ifns file. The values of the i-th input vector uVec<i> are stored in the i-th file from the ifns list. In sampling mode, the data from the input vectors are stored in each period of the control algorithm. In both cases the values from one time instant form one line in the file.

Analogically, the data from output files are copied to the outputs of the block (one line from the i-th file in the ofns list to the i-th output vector yVec<i>).

The inputs working in the sampling mode are defined by the sl list (comma-separated numbers). The outputs work always in the sampling mode – the last values are kept when the end of file is reached. The copying of data to input files can be blocked by the DSI input, the same holds for output data and the DSO input.

Use the RTUV block to combine individual signals into a vector one for the uVec input.
The **RT0V** blocks can be chained, therefore it is possible to create a vector of arbitrary dimension. Similarly, use the **VT0R** block to demultiplex a vector signal to individual signals.

### Inputs

<table>
<thead>
<tr>
<th>uVec1..uVec8</th>
<th>Input vector signal</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXEC</td>
<td>External program is called on rising edge</td>
<td>Bool</td>
</tr>
<tr>
<td>RESET</td>
<td>Block reset (deletes the input and output files and terminates the external program)</td>
<td>Bool</td>
</tr>
<tr>
<td>DSI</td>
<td>Disable inputs sampling</td>
<td>Bool</td>
</tr>
<tr>
<td>DSO</td>
<td>Disable outputs sampling</td>
<td>Bool</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>yVec1..yVec8</th>
<th>Output vector signal</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>DONE</td>
<td>External program finished</td>
<td>Bool</td>
</tr>
<tr>
<td>BUSY</td>
<td>External program running</td>
<td>Bool</td>
</tr>
<tr>
<td>ERR</td>
<td>Error flag</td>
<td>Bool</td>
</tr>
<tr>
<td>errID</td>
<td>Error code</td>
<td>Error</td>
</tr>
<tr>
<td>res</td>
<td>External program return code</td>
<td>Long (132)</td>
</tr>
<tr>
<td>icnt</td>
<td>Current input sample</td>
<td>Long (132)</td>
</tr>
<tr>
<td>ocnt</td>
<td>Current output sample</td>
<td>Long (132)</td>
</tr>
</tbody>
</table>

### Parameters

<table>
<thead>
<tr>
<th>cmd</th>
<th>Operating system command to execute</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>ifns</td>
<td>Input filenames (separated by semicolon)</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>epc_uVec1;epc_uVec2</td>
<td></td>
</tr>
<tr>
<td>ofns</td>
<td>Output filenames (separated by semicolon)</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>epc_yVec1;epc_yVec2</td>
<td></td>
</tr>
<tr>
<td>sl</td>
<td>List of inputs working in the <strong>sampling mode</strong>. The format of the list is e.g. 1, 3..5, 8. Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.</td>
<td>Long (132)</td>
</tr>
<tr>
<td></td>
<td>0 to 255</td>
<td></td>
</tr>
<tr>
<td>ifm</td>
<td>Maximum number of input samples</td>
<td>Long (132)</td>
</tr>
<tr>
<td>format</td>
<td>Format of input and output files</td>
<td>Long (132)</td>
</tr>
<tr>
<td></td>
<td>1 . . . Space-delimited values</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 . . . CSV (decimal point and commas)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 . . . CSV (decimal comma and semicolons)</td>
<td></td>
</tr>
<tr>
<td>mmax</td>
<td>Maximum output vectors length</td>
<td>Long (132)</td>
</tr>
</tbody>
</table>

### Notes
• The called external program has the same priority as the calling task. This priority is high, in some cases higher than operating-system-kernel tasks. On Linux based systems, it is possible to lower the priority by using the `chrt` command:
  
  ```bash
  chrt -o 0 extprg.sh,
  ```

  where `extprg.sh` is the original external program.

• The size of signals is limited by parameter `nmax`. Bigger parameter means bigger memory consumption, so choose this parameter as small as possible.

• The filenames must respect the naming conventions of the target platform operating system. It is recommended to use only alphanumerical characters and an underscore to avoid problems. Also respect the capitalization, e.g. Linux is case-sensitive.

• The block also creates copies of the `ifns` and `ofns` files for implementation reasons. The names of these files are extended by the underscore character.

• The `ifns` and `ofns` paths are relative to the folder where the archives of the REXYGEN system are stored. It is recommended to define a symbolic link to a RAM-drive inside this folder for improved performance. On the other hand, for long series of data it is better to store the data on a permanent storage medium because the data can be appended e.g. after a power-failure recovery.

• The `OSCALL` block can be used for execution of some operating system functions.
HTTP – HTTP GET or POST request (obsolete)

Block Symbol

Function Description

The HTTP block performs a single HTTP GET or POST request. Target address (URL) is defined by url parameter and urldata input. A final URL is formed in the way so that urldata input is simply added to url parameter.

HTTP request is started by the TRG parameter. Then the BUSY output is set until a request is finished, which is signaled by the DONE output. In case of an error, the ERROR output is set. The errId output carries last error identified by REXYGEN system error code. The hterror carries a HTTP status code. All data sent back by server to client is stored in the data output.

The block may be run in blocking or non-blocking mode which is specified by the blocking parameter. In blocking mode, execution of a task is suspended until a request is finished. In non-blocking mode, the block performs only single operation depending on available data and execution of a task is not blocked. It is advised to always run HTTP block in non-blocking mode. It is however necessary to mention that on various operating systems some operations can not be performed in the non-blocking mode, so be careful and do not use this block in quick tasks or in tasks with short execution period. The non-blocking operation is best supported on GNU/Linux operating system. The maximal duration of a request performed by the HTTP block is specified by the timeout parameter.

The block supports user authentication using basic HTTP authentication method. User name and password may be specified by user and password parameters. The block also supports secure HTTP (HTTPS). It is also possible to let the block verify server's certificate by setting the VERIFY parameter. SSL certificate of a server or server's trusted certificate authority must be stored in the certificate parameter in a PEM format. The block does not support any certificate storage.

Parameters postmime and acceptmime specify MIME encoding of data being sent to server or expected encoding of a HTTP response.

Parameters nmax, postmax, and datamax specify maximum sizes of buffers allocated by the block. The nmax parameter is maximal size of any string parameter. The postmax parameter specifies a maximal size of postdata. The datamax parameter specifies a maximal size of data.
Inputs

postdata  Data to put in HTTP POST request  String
urldata  Data to append to URL address  String
TRG  Trigger of the selected action  Bool

Parameters

url  URL address to send the HTTP request to  String
method  HTTP request type  ⬤1 Long (I32)
1   GET
2   POST
user  User name  String
password  Password  String
certificate  Authentication certificate  String
VERIFY  Enable server verification (valid certificate)  Bool
postmime  MIME encoding for POST request  ⬤application/json String
acceptmime  MIME encoding of HTTP response  ⬤application/json String
timeout  Timeout interval  ⬤5.0 Double (F64)
BLOCKING  Wait for the operation to finish  Bool
nmax  Allocated size of string  ↓0 ↑65520 Long (I32)
pmax  Allocated memory for POST request data  ↓128 ↑65520 ⬤256 Long (I32)
datamax  Allocated memory for HTTP response  ↓128 ↑10000000 ⬤1024 Long (I32)

Outputs

data  Response data  String
BUSY  Sending HTTP request  Bool
DONE  HTTP request processed  Bool
ERROR  Error indicator  Bool
errid  Error code  Error
terror  HTTP response  Long (I32)
HTTP2 – Block for generating HTTP GET or POST requests

Block Symbol

Function Description

The HTTP block performs a single HTTP GET or POST request. Target address (URL) is defined by url parameter and urldata input. A final URL is formed in the way so that urldata input is appended to the url parameter. The header input can be used for declaration of additional header fields.

A HTTP request is started by the TRG input. Then the BUSY output is set until the request is finished, which is signaled by a pulse at the DONE output. In case of an error, the pulse is generated at the ERROR output. The errId output carries information about the last error identified by REXYGEN system error code. The hterror carries a HTTP status code. All data received from server is published via the data output. All error outputs are reset when a new HTTP request is triggered by the TRG input.

The block may be run in blocking or non-blocking mode which is specified by the BLOCKING parameter. In blocking mode, execution of a task is suspended until the request is finished. In non-blocking mode, the block performs only single operation depending on available data and execution of a task is not blocked. It is advised to always run HTTP block in non-blocking mode. It is however necessary to mention that on various operating systems some operations cannot be performed in the non-blocking mode, so be careful and do not use this block in quick tasks (qtask) or in tasks with short execution period. The non-blocking operation is best supported on GNU/Linux operating system. The maximal duration of a request performed by the HTTP block is specified by the timeout parameter.

The block supports user authentication using basic HTTP authentication method. User name and password may be specified by user and password parameters. The block also supports secure HTTP (HTTPS). It is also possible to let the block verify server's certificate by setting the VERIFY parameter. SSL certificate of a server or server's trusted certificate authority must be stored in the certificate parameter in a PEM format. The block does not support any certificate storage.

Parameters postmime and acceptmime specify MIME encoding of data being sent to server and expected encoding of the HTTP response.

Parameters nmax, postmax, and datamax specify maximum sizes of buffers allocated by the block. The nmax parameter is maximal size of any string parameter. The postmax parameter specifies a maximal size of postdata. The datamax parameter specifies a
maximal size of data.

**Inputs**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>postdata</td>
<td>Data to put in HTTP POST request</td>
<td>String</td>
</tr>
<tr>
<td>urldata</td>
<td>Data to append to URL address</td>
<td>String</td>
</tr>
<tr>
<td>header</td>
<td>Additional header fields</td>
<td>String</td>
</tr>
<tr>
<td>TRG</td>
<td>Trigger of the selected action</td>
<td>Bool</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>URL address to send the HTTP request to</td>
<td>String</td>
</tr>
<tr>
<td>method</td>
<td>HTTP request type</td>
<td>Long (I32)</td>
</tr>
<tr>
<td></td>
<td>1 ...... GET</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ...... POST</td>
<td></td>
</tr>
<tr>
<td>user</td>
<td>User name</td>
<td>String</td>
</tr>
<tr>
<td>password</td>
<td>Password</td>
<td>String</td>
</tr>
<tr>
<td>certificate</td>
<td>Authentication certificate</td>
<td>String</td>
</tr>
<tr>
<td>VERIFY</td>
<td>Enable server verification (valid certificate)</td>
<td>Bool</td>
</tr>
<tr>
<td>postmime</td>
<td>MIME encoding for POST request</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>○application/json</td>
<td></td>
</tr>
<tr>
<td>acceptmime</td>
<td>MIME encoding for GET request</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>○application/json</td>
<td></td>
</tr>
<tr>
<td>timeout</td>
<td>Timeout interval</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>BLOCKING</td>
<td>Wait for the operation to finish</td>
<td>Bool</td>
</tr>
<tr>
<td>nmax</td>
<td>Allocated size of string</td>
<td>Long (I32)</td>
</tr>
<tr>
<td></td>
<td>↓0 ↑65520</td>
<td></td>
</tr>
<tr>
<td>postmax</td>
<td>Allocated memory for POST request data</td>
<td>Long (I32)</td>
</tr>
<tr>
<td></td>
<td>↓128 ↑65520</td>
<td></td>
</tr>
<tr>
<td>datamax</td>
<td>Allocated memory for HTTP response</td>
<td>Long (I32)</td>
</tr>
<tr>
<td></td>
<td>↓128 ↑10000000</td>
<td></td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Response data</td>
<td>String</td>
</tr>
<tr>
<td>BUSY</td>
<td>Sending HTTP request</td>
<td>Bool</td>
</tr>
<tr>
<td>DONE</td>
<td>HTTP request processed</td>
<td>Bool</td>
</tr>
<tr>
<td>ERROR</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
<tr>
<td>errId</td>
<td>Error code</td>
<td>Error</td>
</tr>
<tr>
<td>hterror</td>
<td>HTTP response</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>
SMTP – Send e-mail message via SMTP

Block Symbol

Function Description

The SMTP block sends a single e-mail message via standard SMTP protocol. The block acts as a simple e-mail client. It does not implement a mail server.

The contents of a message is defined by the inputs subj and body. Parameters from and to specify sender and receiver of a message. A message is sent when the TRG parameter is set. Then the BUSY output is set until the request is finished, which is signaled by the DONE output. In case of an error, the ERROR output is set. The errId output carries the last error identified by REXYGEN system error code. The domain parameter must always be set to identify the target device. The default value should work in most cases. There can be multiple recipients of the message. In such a case, the individual e-mail addresses must be comma-separated and no space character may be present.

The block may be run in non-blocking or blocking mode, which is specified by the BLOCKING parameter.

- In the blocking mode, the execution of a task is suspended until the sending of e-mail is completed. This mode is typically used in tasks with long execution period, \( T_S \geq 10s \). If the e-mail is not successfully sent until timeout expires, an error is indicated and the execution of the task is resumed.

- In the non-blocking mode, the SMTP block performs only a single operation in each execution of the block and the execution of a task is not suspended. This mode is typically used in tasks with short execution period, \( T_S \leq 0.1s \). In this mode, the timeout parameter should be set to at least \( 50 \cdot T_S \), where \( T_S \) is the execution period in seconds.

It is recommended to run the SMTP block in the non-blocking mode. It is however necessary to mention that on various operating systems some operations may not be performed in the non-blocking mode, so be careful and do not use this block in quick tasks (see QTask) or in tasks with extremely short execution period (few milliseconds). The non-blocking mode is best supported on GNU/Linux operating system.

The block supports user authentication using standard SMTP authentication method. User name and password may be specified by the user and password parameters. The block also supports secure connection. The encryption method is selected by the tls parameter. It is also possible to let the block verify server’s certificate by setting the
VERIFY parameter. SSL certificate of a server or server’s trusted certificate authority
must be stored in the certificate parameter in a PEM format. The block does not
support any certificate storage.

The length of the whole message (subject, body and headers) is limited to a maximum
of 1024 characters.

**Inputs**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>subj</td>
<td>Subject of the e-mail message</td>
<td>String</td>
</tr>
<tr>
<td>body</td>
<td>Body of the e-mail message</td>
<td>String</td>
</tr>
<tr>
<td>TRG</td>
<td>Trigger of the selected action</td>
<td>Bool</td>
</tr>
</tbody>
</table>

**Parameters**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>SMTP server address</td>
<td>String</td>
</tr>
<tr>
<td>to</td>
<td>E-mail of the recipient</td>
<td>String</td>
</tr>
<tr>
<td>from</td>
<td>E-mail of the sender</td>
<td>String</td>
</tr>
<tr>
<td>tls</td>
<td>Encryption method</td>
<td>☞1 Long (I32)</td>
</tr>
<tr>
<td></td>
<td>1 ...... None</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ...... StartTLS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 ...... TLS</td>
<td></td>
</tr>
<tr>
<td>user</td>
<td>User name</td>
<td>String</td>
</tr>
<tr>
<td>password</td>
<td>Password</td>
<td>String</td>
</tr>
<tr>
<td>domain</td>
<td>Domain name or identification of the target device</td>
<td>String</td>
</tr>
<tr>
<td>auth</td>
<td>Authentication method</td>
<td>☞1 Long (I32)</td>
</tr>
<tr>
<td></td>
<td>1 ...... Login</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ...... Plain</td>
<td></td>
</tr>
<tr>
<td>certificate</td>
<td>Authentication certificate</td>
<td>String</td>
</tr>
<tr>
<td>VERIFY</td>
<td>Enable server verification (valid certificate)</td>
<td>Bool</td>
</tr>
<tr>
<td>timeout</td>
<td>Timeout interval</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>BLOCKING</td>
<td>Wait for the operation to finish</td>
<td>Bool</td>
</tr>
</tbody>
</table>

**Outputs**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSY</td>
<td>Sending e-mail</td>
<td>Bool</td>
</tr>
<tr>
<td>DONE</td>
<td>E-mail has been sent</td>
<td>Bool</td>
</tr>
<tr>
<td>ERROR</td>
<td>Error indicator</td>
<td>Bool</td>
</tr>
<tr>
<td>errId</td>
<td>Error code</td>
<td>Error</td>
</tr>
</tbody>
</table>
RDC – Remote data connection

Block Symbol

Licence: ADVANCED

Function Description

The RDC block is a special input-output block. The values are transferred between two blocks on different computers, eventually two different Simulink on the same computer or Simulink and the REXYGEN system on the same computer. In order to communicate, the two RDC blocks must have the same id number. The communication is based on UDP/IP protocol. This protocol is used as commonly as the more known TCP/IP, i.e. it works over all LAN networks and the Internet. The algorithm performs the following operations in each step:

• If HLD = on, the block execution is terminated.

• If the period parameter is a positive number, the difference between the system timer and the time of the last packet sending is evaluated. The block execution is stopped if the difference does not exceed the period parameter. If the period parameter is zero or negative, the time difference is not checked.

• A data packet is created. The packet includes block number, the so-called invoke number (serial number of the packet) and the values u0 to u15. All values are stored in the commonly used so-called network byte order, therefore the application is computer and/or processor independent.

• The packet is sent to the specified IP address and port.

• The invoke number is increased by 1.

• It is checked whether any incoming packets have been received.

• If so, the packet validity is checked (size, id number, invoke number).
CHAPTER 15. SPEC – SPECIAL BLOCKS

- If the data is valid, all outputs y0 to y15 are set to the values contained in the packet received.

- The fresh output is updated. In case of error, the error code is displayed by the err output.

There are 16 values transmitted in each direction periodically between two blocks with the same id number. The u(i) input of the first block is transmitted the y(i) output of the other block. Unlike the TCP/IP protocol, the UDP/IP protocol does not have any mechanism for dealing with lost or duplicate packets, so it must be handled by the algorithm itself. The invoke number is used for this purpose. This state variable is increased by 1 each time a packet is sent. The block stores also the invoke number of the last received packet. It is possible to distinguish between various events by comparing these two invoke numbers. The packets with invoke numbers lower than the invoke number of the last received packet are denied unless the difference is greater than 10. This solves the situation when one of the RDC blocks is restarted and its invoke number is reset.

All RDC blocks in the same application must have the same local port number and the number of RDC blocks is limited to 64 for implementation reasons. If there are two applications using the RDC block running on the same machine, then each of them must use a different local port number.

Inputs

HLD Input for disabling the execution of the block. No packets are received nor transmitted when HLD = on.

u0..u15 Values which are sent/written to the output values y0 to y15 of the paired block

Outputs

iE Displays the code of the last error. The error codes are listed below:

0 . . . . No error
Persistent errors originating in the initialization phase (< 0). Cannot be fixed automatically.

-1 .... Maximum number of blocks exceeded (> 64)
-2 .... Local ports mismatch; the lport parameter must be the same for all RDC blocks within one application
-3 .... Error opening socket (the UDP/IP protocol is not available)
-4 .... Error assigning local port (port already occupied by another service or application)
-5 .... Error setting the so-called non-blocking socket mode (the RDC block requires this mode)
-10 .... Error initializing the socket library
-11 .... Error initializing the socket library
-12 .... Error initializing the socket library

Temporary errors originating in any cycle of the code (> 0). Can be fixed automatically.

  1 .... Initialization successful, yet no data packet has been received
  2 .... Packet consistency error (incorrect length – transmission error or conflicting service/application is running)
  4 .... Error receiving packet (socket library error)
  8 .... Error sending packet (socket library error)

fresh Elapsed time (in seconds) since the last received packet. Can be used for detection of an error in the paired block.
y0..y15 Values transmitted from the input ports u0 to u15 of the paired RDC block – data from the last packet received Double (F64)

Parameters

target Name or IP address running the paired RDC block. Broadcast address is allowed.

rport Remote port – address of the UDP/IP protocol service; it is recommended to keep the default value unless necessary (service/application conflict) Word (U16) ⃝1288

lport Local port – similar meaning as the rport parameter; remote port applies to the receiving machine, local port applies to the machine sending the packet ⃝1288

id Block ID – this number is contained within the data packet in order to reach the proper target block (all blocks on the target receive the packet but only the one with the corresponding id decomposes it and uses the data contained to update its outputs) Long (I32) ⌧1 ⌤32767 ⃝1
**period** The shortest time interval between transmitting/receiving *Double* (F64) packets (in seconds). The packets are transmitted/received during each execution of the block for \( \text{period} \leq 0 \) while the positive values of this parameter are extremely useful when sending data out of the Simulink continuous models based on a *Variable step* solver.

**Example**

The following example explains the function of the RDC block. The constants 3 and 5 are sent from *Computer1* to *Computer2*, where they appear at the \( y_0 \) and \( y_1 \) outputs of the RDC2 block. The constants are then summed and multiplied and sent back to *Computer1* via the \( u_{11} \) and \( u_{12} \) outputs of the RDC2 block. The displays connected to the \( y_{11} \) and \( y_{12} \) outputs of the RDC1 block show the results of mathematical operations \( 3 + 5 \) and \( (3 + 5) \times 5 \). The signal from the *SG* generator running on *Computer2* is transmitted to the \( y_0 \) output of the RDC1 block, where it can be easily displayed. Note that *Display* and *Scope* are Matlab/Simulink blocks – to view the data in the REXYGEN system, the REXYGEN Diagnostics diagnostic program and the TRMID block must be used.

![Diagram of computer network](image)

The simplicity of the example is intentional. The goal is to demonstrate the functionality of the block, not the complexity of the system. In reality, the RDC block is used in more complex tasks, e.g. for remote tuning of the PID controller as shown below. The PID control algorithm is running on *Computer1* while the tuning algorithm is executed by *Computer2*. See the PIDU, PIDMA and SSW blocks for more details.
OPC server of the RDC block

There is also an OPC server embedded in the RDC block. Detailed description will be available soon.
REXLANG – User programmable block

Function Description

The standard function blocks of the REXYGEN system cover the most typical needs in control applications. But there still exist situations where it is necessary (or more convenient) to implement an user-defined function. The REXLANG block covers this case.

It implements an user-defined algorithm written in a scripting language very similar to the C language (or Java).

Scripting language

As mentioned, the scripting language is similar to the C language. Nevertheless, there are some differences and limitations:

- Only the `double` and `long` data types are supported (it is possible to use `int, short, bool` as well, but these are internally converted to `long`. The `float` type can be used but it is converted internally to `double`. The `typedef` type is not defined.

- Pointers and structures are not implemented. However, it is possible to define arrays and use the indexes (the `[ ]` operator).

- The `,` operator is not implemented.

- The preprocessor supports the following commands: `#include, #define, #ifdef .. #else .. | #endif, #ifndef .. | #else .. | #endif (i.e. #pragma and if .. [else .. ] #endif are not supported).

- The standard ANSI C libraries are not implemented, however the majority of mathematic functions from `math.h` and some other functions are implemented (see the text below).
The **input**, **output** and **parameter** keywords are defined for referencing the **REXLANG** block inputs, outputs and parameters. System functions for controlling the execution and diagnostics are implemented (see the text below).

The **main()** function is executed periodically during runtime. Alongside the **main()** function the **init()** (executed once at startup), **exit()** (executed once when the control algorithm is stopped) and the **pchange()** (executed on parameters change in **REXYGEN**).

The functions and procedures without parameters must be explicitly declared **void**.

The identifiers cannot be overloaded, i.e. the keywords and built-in functions cannot share the name with an identifier. The local and global variables cannot share the same name.

Array initializers are not supported. Neither in local arrays nor the global ones.

User defined return values of **main()**, **init()** and **exit()** functions are written to **iE** output. Values < -99 will stop algorithm execution (reinitialization by **RESET** input needed for further algorithm run). Return values:

- **iE >= 0** ... No error occurred
- **0 > iE >= -99** ... Warning, no changes to function block algorithm execution
- **iE < -99** ... Error occurred, function block algorithm execution stopped

**Scripting language syntax**

The scripting language syntax is based on the C language, but pointers are not supported and the data types are limited to **long** and **double**. Moreover the **input**, **output** and **parameter** keywords are defined for referencing the **REXLANG** block inputs, outputs and parameters. The syntax is as follows:

- `<type> input(<input number>) <variable name>;`
- `<type> output(<output number>) <variable name>;`
- `<type> parameter(<parameter number>) <variable name>;`

The **input** and **parameter** variables are read-only while the **output** variables are write-only. For example:

```c
double input(1) input_signal; /* declaration of a variable of type double, which corresponds with the u1 input of the block */
long output(2) output_signal; /* declaration of a variable of type long, which corresponds with the y2 output of the block */
```
input_signal = 3; //not allowed, inputs are read-only
sum = output_signal + 1; //not allowed, outputs are write-only
if (input_signal>1) output_signal = 3 + input_signal; //correct

Available functions

The following functions are available in the scripting language:

- **Mathematic functions** (see ANSI C, math.h):
  
  - atan, sin, cos, exp, log, sqrt, tan, asin, acos, fabs, fmod, sinh, cosh, tanh, pow, atan2, ceil, floor and abs. Please note that the abs function works with integer numbers. All the other functions work with variables of type double.

- **Vector functions** (not part of ANSI C)

  ```
  double max(n,ival1,...,ivaln)
  Returns the maximum value. The first parameter defining the number of items is optional.
  
  double max(n,vec)
  Returns the value of maximal item in the vec vector.
  
  double min([n],ival1,...,ivaln)
  Returns the minimum value. The first parameter defining the number of items is optional.
  
  double min(n,vec)
  Returns the value of minimal item in the vec vector.
  
  double poly([n],x,an,...,a1,a0)
  Evaluates the polynomial \( y = a_n x^n + ... + a_1 x + a_0 \). The first parameter defining the number of items is optional.
  
  double poly(n,x,vec)
  Evaluates the polynomial \( y = vec[n] * x^n + ... + vec[1] * x + vec[0] \).
  
  double scal(n,vec1,vec2)
  Evaluates the scalar product \( y = vec1[0] * vec2[0] + ... + vec1[n-1] * vec2[n-1] \).
  
  double scal(n,vec1,vec2,skip1,skip2)
  Evaluates the scalar product \( y = vec1[0] * vec2[0] + vec1[skip1] * vec2[skip2] + ... + vec1[(n-1)*skip1] * vec2[(n-1)*skip2] \). This is well suited for multiplication of matrices, which are stored as vectors (line by line or column by column).
  
  double conv(n,vec1,vec2)
  Evaluates the convoluntary product \( y = vec1[0] * vec2[n-1] + vec1[1] * vec2[n-1] + ... + vec1[n-1] * vec2[0] \).
  
  double sum(n,vec)
  Sums the items in a vector, i.e. \( y = vec[0] + vec[1] + ... + vec[n-1] \).
  ```
double sum([n,]val1,...,valn)
    Sums the items, i.e. \( y = val1 + val2 + \ldots + valn \). The first parameter
defining the number of items is optional.

[]array([n,]len-1,...,a1,a0)
    Returns an array/vector with the given values. The first parameter defining
the number of items is optional. The type of the returned value is
chosen automatically to fit the type of parameters (all must be of the
same type).

[]subarray(idx,vec)
    Returns a subarray/subvector of the vec array, starting at the idx index.
The type of the returned value is chosen automatically according to the
vec array.

copyarray(count,vecSource,idxSource,vecTarget,idxTarget)
    Copies count items of the vecSource array, starting at idxSource index,
to the vecTarget array, starting at idxTarget index. Both arrays must be
of the same type.

void fillarray(vector, value, count)
    Copies value to count items of the vector array (always starting from
index 0).

- **String functions** (ANSI C contains analogous functions in the string.h file)

    string strsub(str,idx,len)
        Returns a substring of length len starting at index idx.

    long strlen(str)
        Returns string length (number of characters).

    long strfind(str,substr)
        Returns the position of first occurrence of substr in str.

    long strfindb(str,substr)
        Returns the position of last occurrence of substr in str.

    strreplace(str,pattern,substr)
        Find all occurrences of pattern in str and replace it with substr (in-place
replacement, so new string is stored into str).

    strupr(str)
        Converts a string to uppercase.

    strlwr(str)
        Converts a string to lowercase.

    long str2long(str [, default])
        Converts string to integer number. The first non-numerical character is
considered the end of the input string and the remaining characters are
ignored. If conversion failed, 2nd parameter is returned (or 0 if parameter
is not set).

    double str2double(str [, default])
        Converts string to a decimal number. The first non-numerical character is
considered the end of the input string and the remaining characters are
ignored. If conversion failed, 2nd parameter is returned (or 0 if parameter
is not set).

string long2str(num [, radix])
Converts an integer number num to text. The optional parameter radix
specifies the numerical system in which the conversion is to be performed
(typically 10 or 16). If radix is not specified, default value is radix = 10.
The output string does not contain any identification of the numerical
system used (e.g. the 0x prefix for the hexadecimal system).

string double2str(num)
Converts a decimal number num to text.

strcpy(dest,src)
Function copies the src string to the dest string. Implemented for com-
patibility with ANSI C. The construction dest=src yields the same result.

strcat(dest,src)
Function appends a copy of the src string to the dest string. Implemented
for compatibility with ANSI C. The construction dest=dest+src yields
the same result.

strcmp(str1,str2)
Function compares strings str1 and str2. Implemented for compatibility
with ANSI C. The construction str1==str2 yields the same result.

float2buf(buf,x[,endian])
Function for converting real number x into 4 elements of array buf. Each
element represents an octet (byte) of number in single precision represen-
tation according to IEEE 754 (known as float). The function is useful for
filling communication buffers. Optional 3rd parameter has the following
meaning: 0 (default) = processor native endian, 1 = little endian, 2 = big
endian.

double2buf(buf,x[,endian])
Similar function to float2buf, but stores 8 elements in double precision
format.

double buf2float(buf[,endian])
Inverse function to float2buf

double buf2double(buf[,endian])
Inverse function to double2buf

long RegExp(str,regexp,capture[])
Compares the str string with regular expression regexp. When the string
matches the pattern, the capture array contains individual sections of the
regular expression. capture[0] is always the complete regular expression.
The function return the number of captured strings or a negative value in
case of an error. The regular expression may contain the following:

(?i) ... Must be at the beginning of the regular expression. Makes
the matching case-insensitive.

^ ... Match beginning of a string

$ ... Match end of a string
() ... Grouping and substring capturing
\s ... Match whitespace
\S ... Match non-whitespace
\d ... Match decimal digit
\n ... Match new line character
\r ... Match line feed character
\f ... Match vertical tab character
\v ... Match horizontal tab character
\t ... Match horizontal tab character
\b ... Match backspace character
+ ... Match one or more times (greedy)
+? ... Match one or more times (non-greedy)
* ... Match zero or more times (greedy)
*? ... Match zero or more times (non-greedy)
? ... Match zero or once (non-greedy)
x|y ... Match x or y (alternation operator)
\meta ... Match one of the meta characters: ^$().[]\*+?\\]
\xHH ... Match byte with hex value 0xHH, e.g. \x4a.
[...][...] ... Match any character from the set. Ranges like [a-z] are supported.
[^...][^...] ... Match any character but the ones from the set.

long ParseJson(string, cnt, names[], values[])
The json string is supposed to contain text in JSON format. The names array contains the requested objects (subitems are accessed via [], index of the array via []). The values array then contains values of the requested objects. The cnt parameter defines the number of requested objects (length of both the names and values arrays). The function returns the number of values, negative numbers indicate errors.

Note: String variable is declared just like in ANSI C, i.e. char <variable name>[<maximum number of characters>]; For passing the strings to functions use char <variable name>[] or string <variable name>.

- System functions (not part of ANSI C)

Archive(arc, type, id, lvl_cnt, value)
Stores a value into the archive subsystem. arc is a bit mask of archives to write the events to (e.g. for writing to archives 3,5 set arc=20 -> (BIN)10100 = (DEC)20). The archives are numbered from 1 and the maximum number of archives is limited to 15 (archive no. 0 is an internal system log). type
1 ... Bool
2 ... Byte (U8)
3 ... Short (I16)
4 ... Long (I32)
5 ... Word (U16)
6 ... DWord (U32)
7 ... Float (F32)
8 ... Double (F64)
9 ... Time
10 ... Large (I64)
11 ... Error
12 ... String
17 ... Bool Group
18 ... Byte Group (U8)
19 ... Short Group (I16)
20 ... Long Group (I32)
21 ... Word Group (U16)
22 ... DWord Group (U32)
23 ... Float Group (F32)
24 ... Double Group (F64)
25 ... Time Group
26 ... Large Group (I64)
27 ... Error Group

id is a unique archive item ID. lv1_cnt is Alarm level in case of alarms or number of elements in case of Group type. value is a value to be written or an array reference in case of Group type.

\textbf{Trace(id, val)}

Displays the id value and the val value. The function is intended for debugging. The id is a user-defined constant (from 0 to 9999) for easy identification of the displayed message. The val can be of any data type including text string. The output can be found in the system log of REXYGEN.

In order to view these debugging messages in System log it is necessary to enable them. Go to the menu \textit{Target}→\textit{Diagnostic messages} and tick the \textit{Information} checkbox in the \textit{Function block messages} box. Logging has to be also enabled for the particular block by ticking the \textit{Enable logging} checkbox in the \textit{Runtime} tab of the block parameters dialog. By default, this is enabled after placing a new block from library. Only then are the messages displayed in the \textit{System log}. 
TraceError(id, val) TraceWarning(id, val) TraceVerbose(id, val)

These commands are similar to the Trace command, only the output is routed to the corresponding logging group (Error, Warning, Verbose). Messages with the Error level are always written to the log. To view the Warning and Verbose messages, enable the corresponding message group. Go to the menu Target→Diagnostic messages and tick the corresponding checkbox in the Function block messages box.

Suspend(sec)

The script is suspended if its execution within the given sampling period takes more seconds than specified by the sec parameter. At the next start of the block the script continues from the point where it was suspended. Use Suspend(0) to suspend the code immediately.

double GetPeriod()

Returns the sampling period of the block in seconds.

double CurrentTime()

Returns the current time (in internal format). Intended for use with the ElapsedTime() function.

double ElapsedTime(new_time, old_time)

Returns the elapsed time in seconds (decimal number), i.e. the difference between the two time values new_time and old_time. The CurrentTime() function is typically used in place of the new_time parameter.

double Random()

Returns a pseudo-random number from the (0,1) interval. The pseudo-random number generator is initialized prior to calling the init() function so the sequence is always the same.

long QGet(var)

Returns the quality of the var variable (see the QFC, QFD, VIN, VOUT blocks). The function is intended for use with the inputs, outputs and parameters. It always returns 0 for internal variables.

void QSet(var, value)

Sets the quality of the var variable (see the QFC, QFD, VIN, VOUT blocks). The function is intended for use with the inputs, outputs and parameters. It has no meaning for internal variables.

long QPropag[in]val1,...,valn)

Returns the quality resulting from merging of qualities of val1,...,valn. The basic rule for merging is that the resulting quality correspond with the worst quality of val1,...,valn. To obtain the same behavior as in other blocks of the REXYGEN system, use this function to set the quality of output, use all the signals influencing the output as parameters.

double LoadValue(fileid, idx)

Reads a value from a file. A binary file with double values or a text file with values on individual lines is supposed. The idx index (binary file)
or line number (text file) starts at 0. The file is identified by $\text{fileid}$. At present the following values are supported:

0 . . file on a disk identified by the $p0$ parameter
1 . . file on disk identified by name of the REXLANG block and extension .dat
2 . . file on a disk identified by the $srcname$ parameter, but the extension is changed to .dat
3 . . $\text{rexlang.dat}$ file in the current directory
4-7 . . same like 0-3, but format is text file. Each line contains one number. The index $\text{idx}$ is the line number and starts at zero. Value $\text{idx}=-1$ means next line (e.g. sequential writing).

void $\text{SaveValue(fileid, idx, value)}$
Stores the $\text{value}$ to a file. The meaning of parameters is the same as in the $\text{LoadValue}$ function.

void $\text{GetSystemTime(time)}$
Returns the system time. The time is usually returned as UTC but this can be altered by the operating system settings. The $\text{time}$ parameter must be an array of at least 8 items of type $\text{long}$. The function fills the array with the following values in the given order: year, month, day (in the month), day of week, hours, minutes, seconds, milliseconds. On some platforms the milliseconds value has a limited precision or is not available at all (the function returns 0 ms).

void $\text{Sleep(seconds)}$
Stop execution of the block’s algorithm (and whole task) for defined time. Use this block with extreme caution and only if there is no other possibility to achieve the desired behaviour of your algorithm. The sleep interval should not exceed 900 milliseconds. The shortest interval is about 0.01s, the precise value depends on the target platform.

long $\text{GetExtInt(ItemID)}$
Returns the value of input/output/parameter of arbitrary block in REXYGEN algorithm. Such an external data item is referenced by the $\text{ItemID}$ parameter. The structure of the string parameter $\text{ItemID}$ is the same as in e.g. the $\text{sc}$ parameter of the GETPI function block. If the value cannot be obtained (e.g. invalid or non-existing $\text{ItemID}$, data type conflict, etc.), the REXLANG block issues an error and must be reset.

long $\text{GetExtLong(ItemID)}$
See $\text{GetExtInt(ItemID)}$.

double $\text{GetExtReal(ItemID)}$
Similar to $\text{GetExtInt(ItemID)}$ but for decimal numbers.

double $\text{GetExtDouble(ItemID)}$
See $\text{GetExtReal(ItemID)}$.

string $\text{GetExtString(ItemID)}$
Similar to $\text{GetExtInt(ItemID)}$ but for strings.
void SetExt(ItemID, value)
Sets the input/output/parameter of arbitrary block in REXYGEN algorithm to value. Such an external data item is referenced by the ItemID parameter. The structure of the string parameter ItemID is the same as in e.g. the sc parameter of the SETPI function block. The type of the external data item (long/double/string) must correspond with the type of the value parameter. If the value cannot be set (e.g. invalid or non-existing ItemID, data type conflict, etc.), the REXLANG block issues an error and must be reset.

int BrowseExt(ItemID, first_subitem_index, max_count, subitems, kinds)
Function browses task address space. If ItemID is a block identifier (block_path), subitems string array will contain names of all inputs, outputs, parameters and internal states. Function returns number of subitems or negative error code. kinds values: executive = 0, module = 1, driver = 2, archive = 3, level = 4, task = 5, quicktask = 6, subsystem = 7, block = 8, input = 9, output = 10, internal state = 11, parameter or state array = 12, special = 13.

long CallExt(ItemID)
Run (one step) arbitrary block in REXYGEN algorithm. Such an external block is referenced by the ItemID parameter. The structure of the string parameter ItemID is the same as in e.g. the sc parameter of the GETPI function block. The function returns result code of the calling block (see REXYGEN error codes). It is strongly recommended to call halted blocks only (set checkbox Halt on the property page Runtime in the parameters dialog of the block) and the block (or subsystem) should be in same task as the REXLANG block.

long GetInArrRows(input)
Returns the number of rows of the array that is attached to the input with index input of the REXLANG block.

long GetInArrCols(input)
Returns the number of columns of the array that is attached to the input with index input of the REXLANG block.

double GetInArrDouble(input, row, col)
Returns the member of the array that is attached to the input with index input of the REXLANG block.

Void SetInArrValue(input, row, col, value)
Sets the member of the array that is attached to the input with index input of the REXLANG block.

Void SetInArrDim(input, row, col)
Sets the dimension of the array that is attached to the input with index input of the REXLANG block.

long memrd32(hMem, offset)
Reading physical memory. Get the handle by Open(/dev/mem", <physical address>, <area size>).
long memwr32(hMem, offset, value)
    Writing to physical memory. Get the handle by OpenMemory("/dev/mem",<physical
    address>,<area size>).

- Communication functions (not part of ANSI C)
  This set of functions is intended for communication over TCP/IP, UDP/IP, serial
  line (RS-232 or RS-485), SPI bus and I2C bus. Only a brief list of available functions
  is given below, see the example projects of the REXYGEN system for more details.

long OpenFile(string filename)
    Function for opening a file. Identification number (the so-called handle)
    of the file is returned. If a negative value is returned, the opening was not
    successful.

long OpenCom(string comname, long baudrate, long parity)
    Function for opening a serial line. Identification number (the so-called han-
    dle) of serial port is returned. If a negative value is returned, the opening
    was not successful. Parity setting: 0=None, 1=odd, 2=even.

long OpenUDP(string localname, long lclPort, string remotename, long remPort)
    Function for opening a UDP socket. Identification number (the so-called handle)
    of the socket is returned. If a negative value is returned, the opening
    was not successful.

long OpenTCPsvr(string localname, long lclPort)
    Function for opening a TCP socket (server, listening). Identification num-
    ber (the so-called handle) of the socket is returned. If a negative value is
    returned, the opening was not successful.

long OpenTCPcli(string remotename, long remPort)
    Function for opening a TCP socket (client). Identification number (the so-
    called handle) of the socket is returned. If a negative value is returned,
    the opening was not successful.

long OpenI2C(string devicename)
    Function for opening the I2C bus. Identification number (the so-called
    handle) of the bus is returned. If a negative value is returned, the opening
    was not successful.

long OpenSPI(string devicename)
    Function for opening the SPI bus. Identification number (the so-called
    handle) of the bus is returned. If a negative value is returned, the opening
    was not successful.

long OpenMemory(string devicename, long baseaddr, long size)
    Function for mapping physical memory. Identification number (the so-
    called handle) of the memory address is returned. If a negative value is
    returned, the opening was not successful.

void Close(long handle)
    Closes the socket, serial line, file or any device opened by the Open...
void SetOptions(long handle, long params[])  
Sets the parameters of a socket or serial line. The array size must be at least:

- 22 for serial line,
- 2 for file (1st item is mode: 1=seek begin, 2=seek current, 3=seek end,  
  4=set file end, 2nd item is offset for seek),
- 3 for SPI (1st item is SPI mode, 2nd item is bits per word, 3rd item is max speed in Hz),
- 5 for I2C (1st item is slave address, 2nd item is 10-bit address flag,  
  3rd item is Packet Error Checking flag, 4th item is number of retries,  
  5th item is timeout)

void GetOptions(long handle, long params[])  
Reads parameters of a socket or serial line to the params array. The array size must be big enough, at least 2 for files, 2 for a socket and 22 for serial line (see SetOptions).

long Accept(long hListen)  
Accepts the connection to listening socket hListen invoked by the client.  
A communication socket handle or an error is returned.

long Read(long handle, long buffer[], long count)  
Reads data from a serial line or socket. The count parameter defines the maximum number of bytes to read. The count of bytes read or an error code is returned. Each byte of incoming data is put to the buffer array of type long in the corresponding order.

It is also possible to use the form
long Read(long handle, string data[], long count) (i.e. a string is used instead of a data array; one byte in the input file corresponds to one character; not applicable to binary files).

The error codes are:
-1 it is necessary to wait for the operation to finish (the function is "non-blocking")
-309 reading failed; the operating system error code appears in the log (when function block logging is enabled)
-307 file/socket is not open

long Write(long handle, long buffer[], long count)  
Sends the data to a serial line or socket. The count parameter defines the number of bytes to send. The count of bytes or an error code sent is returned. Each byte of outgoing data is read from the buffer array of type long in the corresponding order.

It is also possible to use the form
long Write(long handle, string data) (i.e. a string is used instead of a data array; one byte in the output file corresponds to one character; not applicable to binary files).
The error codes are:
-1 it is necessary to wait for the operation to finish (the function is "non-blocking")
-310 write failed; the operating system error code appears in the log (when function block logging is enabled)
-307 file/socket is not open

long ReadLine(long handle, string data)
Read one line from (text) file, serial line or socket; read characters are in the variable data up to allocated size of the string; the function returns real size (number of bytes) of line or error code.

long DeleteFile(string filename)
Delete file. Return 0 if success; negative value is error code.

long RenameFile(string filename, string newfilename)
Rename file. Return 0 if success; negative value is error code.

bool ExistFile(string filename)
Return true if file or device exist (is it possible to open it for reading).

long I2C(long handle, long addr, long bufW[], long cntW, long bufR[], long cntR)
Communication over the I2C bus. Works only in Linux operating system on devices with the I2C bus (e.g. Raspberry Pi). Sends and receives data to/from the slave device with address addr. The parameter handle is returned by the OpenI2C function, whose parameter defines the device name (according to the operating system). The parameter bufW is a buffer (an array) for the data which is sent out, cntW is the number of bytes to send out, bufR is a buffer (an array) for the data which comes in and cntR is the number of bytes to receive. The function returns 0 or an error code.

long SPI(long handle, 0, long bufW[], long cntW, long bufR[], long cntR)
Execution of one transaction over the SPI bus. Works only in Linux operating system on devices with the SPI bus (e.g. Raspberry Pi). The parameter handle is returned by the OpenSPI function, whose parameter defines the device name (according to the operating system). The second parameter is always 0 (reserved for internal use). The parameter bufW is a buffer (an array) for the data which is sent out, cntW is the number of bytes to send out, bufR is a buffer (an array) for the data which comes in and cntR is the number of bytes to receive. Note that SPI communication is full-duplex, therefore the resulting length of the SPI transaction is given by maximum of the cntW and cntR parameters, not their sum. The function returns 0 or an error code.

long Seek(long handle, long mode[], long offset)
Set position for Read/Write command. Parameter mode means: 1=offset from begin of the file, 2= offset from current position, 3=offset from end of the file.

longRecv(long handle, long buffer[], long count)
Obsolete function. Use Read instead.
long Send(long handle, long buffer[], long count)
    
    // Obsolete function. Use Write instead.

long crc16(data, length, init, poly, flags, offset)
    Compute 16-bit cyclic redundancy code that is used as checksum/hash in
    many communication protocols. data byte array (represented by long array)
    or string to compute hash length number of bytes in input array/text
    (could be -1 for whole string) init so called initial vector poly so called
    control polynomial flags 1...revert bit order (in input bytes as so as in result
    crc), 2...result crc is xored with 0xFFF, 4...if data is long array, all 4 bytes
    in long are processed (LSB first), 8... same like 4, but MSB first offset
    index of the first processed byte in data array (usually 0) Notice: there is
    same function for 32-bit CRC long crc32(data, length, init, poly, flags, offset),
    and for 8-bit CRC long crc8(data, length, init, poly, flags, offset). The
    initial vector, control polynomial, flags for many protocols could be found on
    https://crccalc.com/ Examples: MODBUS: crc16("123456789",-1,0xFFF,0x8005,1,0));
    DECT-X: crc16("123456789",-1,0,0x0589,0,0));

Remarks

- The data type of inputs u0..u15, outputs y0..y15 and parameters p0..p15 is
determined during compilation of the source code according to the input, output
and parameter definitions.

- All error codes < -99 require restarting of the REXLANG function block by input
RESET. Of course it is necessary to remove the cause of the error first.

- WARNING! – The inputs and outputs of the block cannot be accessed within the
init() function (the values of inputs are 0, outputs are not set).

- It is possible to include path in the srcname parameter. Otherwise the file is ex-
pected directly in the project directory or in the directories specified by the -I
command line option of the REXYGEN Compiler compiler.

- All parameters of the vector functions are of type double (or array of type double).
The only exception is the n parameter of type long. Note that the functions with
one vector parameter exist in three variants:

    double function(val1,...,valn)
    Vector is defined as a sequence of values of type double.

    double function(n,val1,...,valn)
    Vector is defined as in the first case, only the first parameter defines the
number of values – the size of the vector. This variant is compatible with
the C compiler. The n parameter must be a number, not the so-called
const variable and it must correspond with the number of the following
elements defining the vector.
double function(n, vec)
    The n parameter is an arbitrary expression of type long and defines the number of elements the function takes into account.

- The optional parameter n of the vector functions must be specified if the compatibility with C/C++ compiler is required. In such a case all the nonstandard functions must be implemented as well and the functions with variable number of parameters need to know the parameter count.

- In all case it is important to keep in mind that the vectors start at index 0 and that the array limits are not checked (just like in the C language). E.g. if double vec[10], x; is defined, the elements have indexes 0 to 9. The expression x=vec[10]; is neither a syntax nor runtime error, the value is not defined. More importantly, it is possible to write vec[11]=x;, which poses a threat, because some other variable might be overwritten and the program works unexpectedly or even crashes.

- Only the parser error and line number are reported during compilation. This means a syntax error. If everything seems fine, the problem can be caused by identifier/keyword/function name conflict.

- All jumps are translated as relative, i.e. the corresponding code is restricted to 32767 instructions (in portable format for various platforms).

- All valid variables and temporary results are stored in the stack, namely:

  - Global variables and local static variables (permanently at the beginning of the stack)
  - Return addresses of functions
  - Parameters of functions
  - Local function variables
  - Return value of function
  - Temporary results of operations (i.e. the expression a=b+c; is evaluated in the following manner: b is stored in the stack, c is stored in the stack (it follows after b), the sum is evaluated, both values are removed from the stack and the result is stored in the stack

Each simple variable (long or double) thus counts as one item in the stack. For arrays, only the size is important, not the type.

- The arrays are passed to the functions as a reference. This means that the parameter counts as one item in the stack and that the function works directly with the referenced array, not its local copy.
• If the stack size is not sufficient (less than space required for global variables plus 10), the stack size is automatically set to twice the size of the space required for the global variables plus 100 (for computations, function parameters and local variables in the case that only a few global variables are present).

• If basic debug level is selected, several checks are performed during the execution of the script, namely initialization of the values which are read and array index limits. Also a couple of uninitialized values are inserted in front of and at the back of each declared array. The \texttt{NOP} instructions with line number of the source file are added to the *.ill file.

• If full debug is selected, additional check is engaged – the attempts to access invalid data range are monitored (e.g. stack overflow).

• The term instruction in the context of this block refers to an instruction of a processor-independent mnemocode. The mnemocode is stored in the *.ill file.

• The \texttt{Open()} function set serial line always 19200\,Bd, no parity, 8 bit per character, 1 stopbit, binary mode, no timeout. Optional 2nd (bitrate) and 3th (parity) parameters can be used in the \texttt{Open()} function.

• Accessing text file is significantly slower than binary file. A advantage of the text file is possibility view/edit data in file without special editor.

• This block does not call the \texttt{parchange()} function. It is necessary to call it in \texttt{init()} function (if it is required).

• The block’s inputs are available in the \texttt{init()} function, but all are equal to zero. It is possible (but not common) to set block’s outputs.

• The \texttt{Open()} function also allows opening of a regular file. Same codes like in the \texttt{LoadValue()} function are used.

\section*{Debugging the code}

Use the \texttt{Trace} command mentioned above.

\section*{Inputs}

\begin{itemize}
  \item \texttt{HLD} \quad Hold – the block code is not executed if the input is set to on \quad \texttt{Bool}
  \item \texttt{RESET} \quad Rising edge resets the block. The block gets initialized again (all global variables are cleared and the \texttt{Init()} function is called). \quad \texttt{Bool}
  \item \texttt{u0..u15} \quad Input signals which are accessible from the script \quad \texttt{Unknown}
\end{itemize}
Outputs

**iE**  
Runtime error code. For error codes iE < -99 the algorithm is stopped until it is reinitialized by the RESET input or by restarting the executive.

0 . . . . No error occurred, the whole main() function was executed (also the init() function).

-1 . . . . The execution was suspended using the suspend() command, i.e. the execution will resume as soon as the REXLANG block is executed again.

< -1 . . Error code of the REXYGEN system, see Appendix C.

> 0 . . . User-defined return values, algorithm execution without any change.

**y0..y15**  
Output signals which can be set from within the script.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcname</td>
<td>Source file name</td>
<td>String</td>
</tr>
<tr>
<td>srctype</td>
<td>Coding of source file</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>

1: C-like  
Text file respecting the C-like syntax described above.

2: STL  
Text file respecting the IEC61131-3 standard. The standard is implemented with the same limitations as the C-like script (i.e. no structures, only INT, REAL and STRING data types, function blocks are global variables VAR_INPUT, outputs are global variables VAR_OUTPUT, parameters are global variables VAR_PARAMETER, standard functions according to specification, system and communication functions are the same as in C-like).

3: RLB  
REXYGEN binary file which results from compilation of C-like or STL scripts. Use this option if you do not wish to share the source code of your block.

4: ILL  
Text file with mnemonics, which can be compared to assembler. This choice is currently not supported.

**stack**  
Stack size defined as number of variables. Default and recommended value is 0, which enables automatic estimation of the necessary stack size.

**debug**  
Debug level – checking is safer but slows down the execution of the algorithm. Option No check can crash REXYGEN application on target platform if code is incorrect.

1 . . . . No check

2 . . . . Basic check

3 . . . . Full check

**strs**  
Total size of buffer for strings. Enter the maximum number of characters to allocate memory for. The default value 0 means that the buffer size is determined automatically.

**p0..p15**  
Parameters which are accessible from the script.
Example C-like

The following example shows a simple code to sum two input signals and also sum two user-defined parameters.

double input(0) input_u0;
double input(2) input_u2;

double parameter(0) param_p0;
double parameter(1) param_p1;

double output(0) output_y0;
double output(1) output_y1;

double my_value;

long init(void)
{
    my_value = 3.14;
    return 0;
}

long main(void)
{
    output_y0 = input_u0 + input_u2;
    output_y1 = param_p0 + param_p1 + my_value;
    return 0;
}

long exit(void)
{
    return 0;
}

Example STL

And here is the same example in Structured Text.

VAR_INPUT
    input_u0:REAL;
    input_u1:REAL;
    input_u2:REAL;
END_VAR

VAR_OUTPUT
    output_y0:REAL;
output_y1:REAL;
END_VAR

VAR_PARAMETER
  param_p0:REAL;
  param_p1:REAL;
END_VAR

VAR
  my_value: REAL;
END_VAR

FUNCTION init : INT;
  my_value := 3.14;
  init := 0;
END_FUNCTION

FUNCTION main : INT;
  output_y0 := input_u0 + input_u2;
  output_y1 := param_p0 + param_p1 + my_value;
  main := 0;
END_FUNCTION

FUNCTION exit : INT;
  exit := 0;
END_FUNCTION
Chapter 16

LANG – Special blocks

Contents

   PYTHON – User programmable block in Python     . . . . . . . . . .468
PYTHON – User programmable block in Python

Block Symbol

 Licence: REXLANG

Function Description

The standard function blocks of the REXYGEN system cover the most typical needs in control applications. But there still exist situations where it is necessary (or more convenient) to implement an user-defined function. The REXLANG block covers this case for application where real-time behavior is strictly demanded. In the rest of the cases the PYTHON block can be used.

The PYTHON block implements an user-defined algorithm written in a Python scripting language and in comparison to the REXLANG block it provides a better user experience in the stage of development of the algorithm and can extend the feature set of REXYGEN system through various 3rd party libraries that are available in the Python environment.

Warning: the PYTHON block is intended for prototyping and experiments so please consider using the block in your application very carefully. It is an experimental block and always will be. There are many corner cases that may lead to unexpected behavior or even block the runtime. Packages may be poorly written or provide incorrect finalization and reinitialization which may even lead to a crash. Only a very limited support is provided for this block.

Scripting language

The scripting language is a standard Python v.3 language (see [9]). Every block references a script written in a *.py source file. The file can optionally contain functions with a reserved name that are then executed by REXYGEN. The main() function is executed periodically during runtime. Alongside the main() function the init() function is executed once at startup and after reset of the block, the exit() function is executed once when the control algorithm is stopped and before reset of the block and the parchange() function is executed on parameters change in REXYGEN.
Scripts on the target device

Standard python interpreter can load modules/scripts from various locations on the target device. The PYTHON block can reference any python script available for the standard interpreter and in addition the block can access scripts located in a directory /rex/scripts/python. User scripts can be directly uploaded to this directory or if the parameter embedded is set to on the script referenced by the block gets embedded in the REXYGEN configuration during compilation process and will be temporarily stored in the directory /rex/scripts/python/embedded during initialization of the block once the configuration is downloaded and executed on the target device.

Data exchange API

For the purpose of data exchange between a Python interpreter and REXYGEN system a module PyRexExt was developed as a native extension to the interpreter. The module contains an object REX that handles the data exchange operations. Use the following snippet at the start of the script to setup the data exchange API.

```python
from PyRexExt import REX
```

I/O objects

- **RX.0 - RX.15**: objects representing block *inputs* in Python environment
- **RX.p0 - RX.p15**: objects representing block *parameters* in Python environment
- **RX.y0 - RX.y15**: objects representing block *outputs* in Python environment

Access to values

All I/O objects contain a property v. Reading of the property v performs a conversion from REXYGEN data types to Python data types. The value then can be stored in variables and used in the block algorithm. A REXYGEN array type converts into a list of values in case of one-dimensional array or into a list of lists in case of multidimensional array.

Example of reading a value of the block input:

```python
x = RX.0.v
```

Writing to the property v, on the other hand, performs a conversion from Python data types to REXYGEN data types and exports the value to the corresponding block output/parameter.

Example of writing a value to the block output:
REX.y0.v = 5

Arrays
Input and output objects have a readonly property size. It is a tuple with number of rows and columns. Arrays can be manipulated through property v but direct conversions between REXYGEN arrays and Python lists are not very memory efficient. However, input and output objects support indexing operator [] that restricts the conversion overhead only on the specified item.
Example of reading a value of the block input for one-dimensional array:

\[ x = \text{REX}.u0[0] \]

Example of writing a value to the block output for multidimensional array:

\[ \text{REX}.u0[1, 3] = 5 \]

External items
The object REX contains a method Item that returns a handle to an external REXYGEN item based on a connection string specified in a parameter of the method.
Example of creating a handle to an external item and setting its value:

\[ \text{cns} = \text{REX}.\text{Item}("\text{myproject_task.CNS:scv}\"") \]
\[ \text{cns.v} = "abc" \]

Tracing
The object REX contains methods Trace, TraceError, TraceWarning, TraceVerbose and TraceInfo can be used to write messages into REXYGEN system log. Every message has a stacktrace attached.
Example of logging a message:

\[ \text{REX}.\text{Trace}("abc") \]

Additional features
REX.RexDataPath – RexDataPath is a string constant that contains a path to a data folder of the REX system on the given platform. That can come handy for writing a platform independent code that requires access to the file system using absolute paths.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLD</td>
<td>Hold – the block code is not executed if the input is set to on</td>
</tr>
<tr>
<td>RESET</td>
<td>Rising edge resets the block. The block gets initialized again (all global variables are cleared and the init () function is called).</td>
</tr>
<tr>
<td>u0..u15</td>
<td>Input signals which are accessible from the script.</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>iE</th>
<th>Runtime error code.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>0       . . . No error occurred, the whole main() function was executed (also the init() function).</td>
</tr>
<tr>
<td></td>
<td>xxx     . . . Error code of the REXYGEN system, see Appendix C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iRes</th>
<th>Execution result code.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Long (132)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>y0..y15</th>
<th>Output signals which can be set from within the script.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unknown</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>srcname</th>
<th>Source file name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>oprogram.py</td>
</tr>
<tr>
<td></td>
<td>String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>embedded</th>
<th>Embedding of the script</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>on</td>
</tr>
<tr>
<td></td>
<td>Bool</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>p0..p15</th>
<th>Parameters which are accessible from the script.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unknown</td>
</tr>
</tbody>
</table>

Data types definition

For data exchange between REXYGEN system and Python environment the data types of block inputs signals u0..u15, outputs signals y0..y15 and parameters p0..p15 must be explicitly specified. For that purpose a configuration file must be created for every python script with the same name plus a suffix .cfg (e.g. program.py.cfg). If the file is missing during the compilation process it is created with all signal types set to double. It is not expected this file to be edited directly. User should use a build-in editor specific to the PYTHON block instead. Available types for inputs outputs and parameters are boolean, uint8, int16, uint16, int32, uint32, int64, float, double, string and in addition the inputs and outputs support array, numpy and image data types.

For types numpy and image the numpy python package must be installed on the target device. Inputs of the type numpy expect the connected signal to be of the type array that gets converted in the runtime to a native numpy representation. Inputs of the type image expects the connected signal to be of the type image data type from the RexVision module that also gets converted in the runtime to a native numpy representation and can therefore be directly used with the OpenCV Python package.

Outputs of the type numpy expect to be set in the script from a numpy array object that gets converted to a regular array. Outputs of the type image expect to be set in the script from a numpy array object that gets converted to image data type defined in the RexVision module.
Example data types definition

The following example shows a shortened JSON file describing the data types of the program inputs and outputs.

```
{
  "types": {
    "in": [
      {
        "idx": 0,
        "type": "double"
      },
      ...
      {
        "idx": 15,
        "type": "double"
      }
    ],
    "param": [
      {
        "idx": 0,
        "type": "double"
      },
      ...
      {
        "idx": 15,
        "type": "double"
      }
    ],
    "out": [
      {
        "idx": 0,
        "type": "double"
      },
      ...
      {
        "idx": 15,
        "type": "double"
      }
    ]
  }
}
```
Example Python script

The following example shows a simple code to sum two input signals and also sum two user-defined parameters.

```
from PyReXExt import REX

def main():
    REX.y0.v = REX.u0.v + REX.u1.v
    REX.y1.v = REX.p0.v + REX.p1.v
    return

Installation - Debian

The Python environment should be correctly installed and configured just by installing the PythonBlk_T debian package. To install the package with optional numpy and OpenCV packages execute these commands from the terminal.

```
sudo apt install rex-pythonblkt
sudo apt install python3-numpy python3-opencv
```

Installation - Windows

To install the correct version of Python the recommended way is to download and install the 64-bit version from official repository (https://www.python.org/ftp/python/3.7.3/). During the installation make sure to enable installation of the pip program and adding of the python binaries to the system variable PATH.

To install numpy and OpenCV as optional dependencies execute following commands from the command line.

```
!pip install numpy
!pip install opencv-python
```

Limitations

Due to the limitations of the standard Python interpreter implementation it is not recommended to use multiple PYTHON block instances at the different levels of executive. Doing so can lead to an unpredictable behavior and instability of the RexCore program.
Chapter 17

MQTT – Communication via MQTT protocol

Contents

\begin{itemize}
\item \texttt{MQttPublish} – Publish MQTT message \hspace{1cm} 476
\item \texttt{MQttSubscribe} – Subscribe to MQTT topic \hspace{1cm} 478
\end{itemize}
MqttPublish — Publish MQTT message

Block Symbol

Licence: MQTT

Function Description

This function block depends on the MQTT driver. Please read the MQTTDrv manual [10] before use.

The MqttPublish block publishes messages to an MQTT broker through the connection established by the MQTTDrv driver.

The first parameter is the topic the block will publish the messages to. MQTT delivers Application Messages according to the Quality of Service (QoS) levels. Use the QoS parameter to set a different Quality of Service level. See the MQTT specification [11] for more details.

If the RETAIN parameter is set a RETAIN flag will be set on the outgoing PUBLISH Control Packet. See the MQTT specification [11] for more details.

The defBuffSize parameter can be used to optimize the memory usage of the block. It states the amount of the statically allocated memory for the inner buffer for the outgoing messages. If the value is unnecessarily large the memory is being wasted. On the other hand if the value of the parameter is too small it leads to frequent dynamic memory allocations which can be time consuming.

The message to be published is constructed from the value input signal. The value input signal is expected to be a string. If it is not a string it will be converted automatically. To request a message to be published in the current period set the RUN flag to on. The BUSY flag is on if the block has a pending request and waits for a response from a broker. When the response is received in the current cycle the DONE flag is set to on.

Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>String</td>
</tr>
<tr>
<td>RUN</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>String</td>
</tr>
</tbody>
</table>
QoS  Quality of Service  ⊗1  Long (132)
  1 . . . . QoS0 (At most once)
  2 . . . . QoS1 (At least once)
  3 . . . . QoS2 (Exactly once)

RETAIN  Retain last message  ⊗on  Bool

defBuffSize  Default buffer size  ⊖1 ⊖2048  Long (132)

Outputs

BUSY  Busy flag  Bool
DONE  Indicator of finished transaction  Bool
errId  Error code  Error
MqttSubscribe – Subscribe to MQTT topic

Function Description

This function block depends on the MQTT driver. Please read the MQTTDrv manual [10] before use.

The MqttSubscribe block subscribes to a topic on an MQTT broker and receives Publish messages on that topic through the connection established by the MQTTDrv driver. The first parameter is the topic the block will subscribe to. MQTT protocol delivers Application Messages according to the Quality of Service (QoS) levels. Use the QoS parameter to set a different Quality of Service level. See the MQTT specification [11] for more details.

By setting the type parameter of the block it can be specified the expected data type of the incoming message. The block converts the incoming message to the specified type and sets the value output signal in case of success or it sets the errId to the resulting error code.

The mode parameter has two available options: Last value and Buffered values. If Last value mode is used the block will always output only the last message received even if multiple messages were received in the last period. If the mode is set to Buffered values than the block buffers the incoming messages and outputs one by one in consecutive ticks of the task.

The defBuffSize parameter can be used to optimize the memory usage of the block. It states the amount of the statically allocated memory in the inner buffer for the incoming messages. If the value is unnecessarily large the memory is being wasted. On the other hand if the value of the parameter is too small it leads to frequent dynamic memory allocations which can be time consuming.

A Subscribe action is performed upon a rising edge (off→on) and an Unsubscribe action is performed upon a falling edge (on→off) at the RUN input.

The nDRDY output specifies how many messages were received and are available in the inner buffer. If the mode of the block is set to Last value the nDRDY output can only have value 0 or 1.

The RETAIN output flag is set if the received Publish packet had the RETAIN flag set. See the MQTT specification [11] for more details.
Note that subscribing to topics containing wildcards is not supported.

**Input**

| RUN | Enable execution | Bool |

**Parameters**

<table>
<thead>
<tr>
<th>topic</th>
<th>MQTT topic</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
<td>○1 Long (I32)</td>
</tr>
</tbody>
</table>

1. ...  QoS0 (At most once)
2. ...  QoS1 (At least once)
3. ...  QoS2 (Exactly once)

<table>
<thead>
<tr>
<th>type</th>
<th>Expected type of incoming data</th>
<th>○1 Long (I32)</th>
</tr>
</thead>
</table>
1. ...  string
2. ...  double
3. ...  long
4. ...  bool
5. ...  byte vector/blob

<table>
<thead>
<tr>
<th>mode</th>
<th>Incoming messages buffering mode</th>
<th>○1 Long (I32)</th>
</tr>
</thead>
</table>
1. ...  Last value
2. ...  Buffered values

| defBuffSize | Default buffer size | ▪1 ○2048 Long (I32) |

**Outputs**

<table>
<thead>
<tr>
<th>value</th>
<th>Output signal</th>
<th>Unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>nDRDY</td>
<td>Number of received messages</td>
<td>▪0 ↑10 Long (I32)</td>
</tr>
<tr>
<td>errId</td>
<td>Error code</td>
<td>Error</td>
</tr>
</tbody>
</table>
Chapter 18

MC_SINGLE – Motion control - single axis blocks

Contents

RM_Axis – Motion control axis ........................................ 484
MC_AccelerationProfile, MCP_AccelerationProfile – Acceleration
profile ................................................................. 489
MC_Halt, MCP_Halt – Stopping a movement (interruptible) .... 493
MC_HaltSuperimposed, MCP_HaltSuperimposed – Stopping a move-
ment (superimposed and interruptible) ............................. 494
MC_Home, MCP_Home – Homing .................................... 495
MC_MoveAbsolute, MCP_MoveAbsolute – Move to position (absolute
cordinate) ................................................................ 497
MC_MoveAdditive, MCP_MoveAdditive – Move to position (relative
to previous motion) .................................................... 501
MC_MoveRelative, MCP_MoveRelative – Move to position (relative
to execution point) ..................................................... 504
MC_MoveSuperimposed, MCP_MoveSuperimposed – Superimposed move 507
MC_MoveContinuousAbsolute, MCP_MoveContinuousAbsolute – Move
to position (absolute coordinate) ................................... 510
MC_MoveContinuousRelative, MCP_MoveContinuousRelative – Move
to position (relative to previous motion) ......................... 513
MC_MoveVelocity, MCP_MoveVelocity – Move with constant velocity 517
MC_PositionProfile, MCP_PositionProfile – Position profile .... 521
MC_Power – Axis activation (power on/off) ....................... 525
MC_ReadActualPosition – Read actual position ................... 526
MC_ReadAxisError – Read axis error ............................... 527
MC_ReadBoolParameter – Read axis parameter (bool) .......... 528
MC_ReadParameter – Read axis parameter ......................... 529
CHAPTER 18. MC_SINGLE – MOTION CONTROL - SINGLE AXIS BLOCKS

MC_ReadStatus – Read axis status .................. 531
MC_Reset – Reset axis errors ....................... 533
MC_SetOverride, MCP_SetOverride – Set override factors ...... 534
MC_Stop, MCP_Stop – Stopping a movement .......... 536
MC_TorqueControl, MCP_TorqueControl – Torque/force control ... 538
MC_VelocityProfile, MCP_VelocityProfile – Velocity profile .... 541
MC_WriteBoolParameter – Write axis parameter (bool) ....... 545
MC_WriteParameter – Write axis parameter ......... 546
RM_AxisOut – Axis output .......................... 547
RM_AxisSpline – Commanded values interpolation .... 548
RM_Track – Tracking and inching .................. 549

This library includes functional blocks for single axis motion control as it is defined in the PLCopen specification. It is recommended to study the PLCopen specification prior to using the blocks from this library. The knowledge of PLCopen is necessary for advanced use of the blocks included in this library.

PLCopen defines all blocks with the MC_ prefix. This notation is kept within this library. Nevertheless, there are also function blocks, which are not described by PLCopen or are described as vendor specific. These blocks can be recognized by the RM_ prefix. Note that PLCopen (and also IEC 61131-3 which is the base for PLCopen) does not use block parameters, all the parameters are specified by input signals. In the Rexygen, block parameters are used to simplify usage of the blocks. To keep compatibility with PLCopen and improve usability of the blocks, almost all of them are implemented twice: with prefix MC_ without parameters (parameters are inputs) and with prefix MCP_ with parameters. Some blocks require additional vendor specific parameters. In such a case even the MC_ prefixed blocks contain parameters.

PLCopen specifies that all inputs/parameters are sampled at rising-edge of the Execute input. In Rexygen block parameters are usually changed very rare. Therefore the parameters of the activated block have not be changed until block is finished (e.g. while output Busy is on).

The Rexygen system does not allow input-output signals and all signals must have different name. For these reasons the Axis input-output signal, which is used in all blocks, is divided into input uAxis and output yAxis. The block algorithm copies the input uAxis to the output yAxis. The yAxis output is not necessary for the function of motion control blocks, but "chaining!" the axis references makes it possible to order the blocks and define priorities. Other reference signals are either defined as input-only or use this mechanism as well.

PLCopen defines the outputs Busy, Active, CommandAborted as optional in almost all blocks. In Rexygen, some of them are never set, but the outputs are defined to simplify future extensions and/or changes in the implementation.

Units used for position and distance of axis are implementation specific. It can be meters, millimeters, encoder ticks, angular degrees (for rotational axis) or any others,
but all blocks connected to one axis must use the same position units. Time is always defined in seconds. Velocity unit is thus "position units per second" and acceleration unit is "position units per square second".

The REXYGEN system uses more threads for execution of the function blocks. In standard function blocks the synchronization is provided by the system and the user does not need to care about it. But using the Reference references could violate the synchronization mechanisms. However, there is no problem if all referenced blocks are located in the same task and therefore e.g. the RM_Axis block must be in the same task as all other blocks connected to this axis.

Some inputs/parameters are of enumeration type (for example BufferMode or Direction). It is possible to choose any of the defined values for this type in the MCP_ version of the blocks, although not all of them are valid for all blocks (for example the block MC_MoveVelocity does not support Direction = shortest_way). Valid values for each block are listed in this manual.
RM_Axis – Motion control axis

Block Symbol

Licence: MOTION CONTROL

Function Description

The RM_AXIS block is a cornerstone of the motion control solution within the REXYGEN system. This base block keeps all status values and implements basic algorithm for one motion control axis (one motor), which includes limits checking, emergency stop, etc. The block is used for both real and virtual axes. The real axis must have a position feedback controller, which is out of this block’s scope. The key status values are commanded position, velocity, acceleration and torque, as well as state of the axis, axis error code and a reference to the block, which controls the axis.

This block (like all blocks in the motion control library) does not implement a feedback controller which would keep the actual position as near to the commanded position as possible. Such a controller must be provided by using other blocks (e.g. PIDU) or external (hardware) controller. The feedback signals are used for lag checking, homing and could be used in special motion control blocks.

The parameters of this block correspond with the requirements of the PLCopen standard for an axis. If improper parameters are set, the errorID output is set to -700 (invalid parameter) and all motion blocks fail with the -720 error code (general failure).

Note that the default values for position, velocity and acceleration limits are intentionally set to 0, which makes them invalid. Limits must always be set by the user according to the real axis and the axis actuator.

Example

Following example illustrates basic principle of use of motion control blocks. It presents the minimal configuration which is needed for operation of a physical or virtual axis. The axis is represented by RM_Axis block. The limitations imposed on the motion trajectory in form of maximum velocity, acceleration, jerk and position have to be set in parameters of the RM_Axis block. The inputs can be connected to supply the values of actual position, speed and torque (feedback for slip monitoring) or logical limit switch signals for homing procedure. The axisRef output signal needs to be connected to any motion control block...
related to the corresponding axis. The axis has to be activated by enabling the \texttt{MC\_Power} block. The state of the axis changes from Disabled to Standstill (see the following state transition diagram) and any discrete, continuous or synchronized motion can be started by executing a proper functional block (e.g. \texttt{MC\_MoveAbsolute}). The trajectory of motion in form of desired position, velocity and acceleration is generated in output signals of the \texttt{RM\_Axis} block. The reference values are provided to an actuator control loop which is implemented locally in \texttt{REXYGEN} system in the same or different task or they are transmitted via a serial communication interface to end device which controls the motor motion (servo amplifier, frequency inverter etc.). In case of any error, the axis performs an emergency stop and indicates the error ID. The error has to be confirmed by executing the \texttt{MC\_Reset} block prior to any subsequent motion command. The following state diagram demonstrates the state transitions of an axis.
Axis state transition diagram

Note 1: From any state. An error in the axis occurred.
Note 2: From any state. MC_Power.Enable = FALSE and there is no error in the axis.
Note 3: MC_Reset AND MC_Power.Status = FALSE
Note 6: MC_Stop.Done = TRUE AND MC_Stop.Execute = FALSE
Motion blending

According to PLCoOpen specification, number of motion control blocks allow to specify BufferMode parameter, which determines a behaviour of the axis in case that a motion command is interrupted by another one before the first motion is finished. This transition from one motion to another (called "Blending") can be handled in various ways. The following table presents a brief explanation of functionality of each blending mode and the resulting shapes of generated trajectories are illustrated in the figure. For detailed description see full PLCoOpen specification.

<table>
<thead>
<tr>
<th>Blending</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aborting</td>
<td>The new motion is executed immediately</td>
</tr>
<tr>
<td>Buffered</td>
<td>the new motion is executed immediately after finishing the previous one, there is no blending</td>
</tr>
<tr>
<td>Blending low</td>
<td>the new motion is executed immediately after finishing the previous one, but the axis will not stop between the movements, the first motion ends with the lower limit for maximum velocity of both blocks at the first end-position</td>
</tr>
<tr>
<td>Blending high</td>
<td>the new motion is executed immediately after finishing the previous one, but the axis will not stop between the movements, the first motion ends with the higher limit for maximum velocity of both blocks at the first end-position</td>
</tr>
<tr>
<td>Blending previous</td>
<td>the new motion is executed immediately after finishing the previous one, but the axis will not stop between the movements, the first motion ends with the limit for maximum velocity of first block at the first end-position</td>
</tr>
<tr>
<td>Blending next</td>
<td>the new motion is executed immediately after finishing the previous one, but the axis will not stop between the movements, the first motion ends with the limit for maximum velocity of second block at the first end-position</td>
</tr>
</tbody>
</table>
Illustration of blending modes

Aborting

Buffered

Blending low

Blending high

Blending next

Blending previous

- Commanded velocity
- Active block 1 =false/Active block 2 = true
- Value of the maximum velocity v_1 =30 (block 1)
- Value of the maximum velocity v_2 =15 (block 2)
MC_AccelerationProfile, MCP_AccelerationProfile – Acceleration profile

Block Symbols

Function Description

The MC_AccelerationProfile and MCP_AccelerationProfile blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP version of the block.

The MC_PositionProfile block commands a time-position locked motion profile. Block implements two possibilities for definition of time-acceleration function:

1. sequence of values: the user defines a sequence of time-acceleration pairs. In each time interval, the values of velocity are interpolated. Times sequence is in array times, position sequence is in array values. Time sequence must be increasing and must start with zero or zero must be between the first and last point. Execution always starts from zero time, so if the sequence start with negative time, part of the profile is not executed (could be used for debugging or time shift). For MC_VelocityProfile and MC_AccelerationProfile interpolation is linear, but for MC_PositionProfile, 3rd order polynomial is used in order to avoid steps in velocity.

2. spline: time sequence is the same as in previous case. Each interval is interpolated by 5th order polynomial \( p(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \) where beginning of the time-interval is for \( x = 0 \), end of time-interval is for \( x = 1 \) and factors \( a_i \) are put in array values in ascending order (e.g. array values contains 6 values for each interval). This method allows smaller number of intervals and there is special editor for synthesis of the interpolating spline function.

For both case, the time sequence could be equally spaced and then array times includes only the first (usually zero) and last point.

Note 1: input TimePosition is missing, because all path data are in parameters of the block.

Note 2: parameter values must be set as vector in all cases, e.g. text string must not include semicolon.
Note 3: incorrect parameter cSeg (higher then real size of arrays times and/or
values) leads to unpredictable result and in some cases crashes whole runtime execution
(The problem is platform dependent and currently it is known only for SIMULINK -
crash of whole MATLAB).

Note 4: in the spline mode, polynomial is always 5th order and always in position
(also for sibling block MC_PositionProfile and MC_VelocityProfile) and it couldn’t
be changed. As the special editor exists, this is not important limitation.

Note 5: The block does not include ramp-in mode. If start position and/or velocity of
profile is different from actual (commanded) position of axis, block falls with error -707
(step). It is recommended to use BufferMode=BlendingNext to eliminate the problem
with start velocity.

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis)</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>Bool</td>
</tr>
<tr>
<td>TimeScale</td>
<td>Overall scale factor in time</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>AccelerationScale</td>
<td>Overall scale factor in value</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Offset</td>
<td>Overall profile offset in value</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
<td></td>
</tr>
<tr>
<td>1 ....</td>
<td>Aborting (start immediately)</td>
<td></td>
</tr>
<tr>
<td>2 ....</td>
<td>Buffered (start after finish of previous motion)</td>
<td></td>
</tr>
<tr>
<td>3 ....</td>
<td>Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
<td></td>
</tr>
<tr>
<td>4 ....</td>
<td>Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
<td></td>
</tr>
<tr>
<td>5 ....</td>
<td>Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)</td>
<td></td>
</tr>
<tr>
<td>6 ....</td>
<td>Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)</td>
<td></td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM_Axis.axisRef-yAxis or yAxis-yAxis)</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>Bool</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
<td>Bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>Bool</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
<td>Bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>Bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>Error</td>
</tr>
<tr>
<td>i ....</td>
<td>REXXYGEN general error</td>
<td></td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>alg</code></td>
<td>Algorithm for interpolation</td>
<td><img src="image" alt="alg" /></td>
<td><img src="image" alt="alg" /></td>
</tr>
<tr>
<td><code>cSeg</code></td>
<td>Number of profile segments</td>
<td><img src="image" alt="cSeg" /></td>
<td><img src="image" alt="cSeg" /></td>
</tr>
<tr>
<td><code>times</code></td>
<td>Times when segments are switched</td>
<td><img src="image" alt="times" /></td>
<td><img src="image" alt="times" /></td>
</tr>
<tr>
<td><code>values</code></td>
<td>Values or interpolating polynomial coefficients</td>
<td><img src="image" alt="values" /></td>
<td><img src="image" alt="values" /></td>
</tr>
</tbody>
</table>
Example
MC_Halt, MCP_Halt – Stopping a movement (interruptible)

Block Symbols

Function Description

The MC_Halt and MCP_Halt blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The MC_Halt block commands a controlled motion stop and transfers the axis to the state DiscreteMotion. After the axis has reached zero velocity, the Done output is set to true immediately and the axis state is changed to Standstill.

Note 1: Block MC_Halt is intended for temporary stop of an axis under normal working conditions. Any next motion command which cancels the MC_Halt can be executed in nonbuffered mode (opposite to MC_Stop, which cannot be interrupted). The new command can start even before the stopping sequence was finished.

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
</tbody>
</table>

i ..... REXYGEN general error
MC_HaltSuperimposed, MCP_HaltSuperimposed – Stopping a movement (superimposed and interruptible)

Block Symbols

Function Description

The MC_HaltSuperimposed and MCP_HaltSuperimposed blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

Block MC_HaltSuperimposed commands a halt to all superimposed motions of the axis. The underlying motion is not interrupted.

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis Reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis Reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td>ErrorID</td>
<td>REXYGEN general error</td>
</tr>
</tbody>
</table>
MC_Home, MCP_Home – Homing

Block Symbols

Function Description

The MC_Home and MCP_Home blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The MC_Home block commands the axis to perform the "search home" sequence. The details of this sequence are described in PLCopen and can be set by parameters of the block. The "Position" input is used to set the absolute position when reference signal is detected. This Function Bock completes at "StandStill".

Note 1: Parameter/input BufferMode is not supported. Mode is always Aborting. It is not limitation, because homing is typically done once in initialization sequence before some regular movement is proceeded.

Note 2: Homing procedure requires some of RM_Axis block input connected. Depending on homing mode, ActualPos, ActualTorque, LimP, LimZ, LimN can be required. It is expected that only one method is used. Therefore, there are no separate inputs for zero switch and encoder reference pulse (both must be connected to Lim2).

Note 3: HomingMode=4(Direct ) only sets the actual position. Therefore, the MC_SetPosition block is not implemented. HomingMode=5(Absolute) only switches the axis from state Homing to state StandStill.

Note 4: Motion trajectory for homing procedure is implemented in simpler way than for regular motion commands - acceleration and deceleration is same (only one parameter) and jerk is not used. For extremely precise homing (position set), it is recommended to run homing procedure twice. First, homing procedure is run with "high" velocity to move near zero switch, then small movement (out of zero switch) follows and finally second homing procedure with "small" velocity is performed.

Note 5: HomingMode=6(Block) detect home-position when the actual torque reach value in parameter TorqueLimit or position lag reach value in parameter MaxPositionLag
in attached RM_Axis block (only if the parameter has positive value).

**Inputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Axis reference (only RM_Axis.axisRef−uAxis or yAxis−uAxis</td>
<td>Reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>Bool</td>
</tr>
<tr>
<td>Velocity</td>
<td>Maximal allowed velocity [unit/s]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s²]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>TorqueLimit</td>
<td>Maximal allowed torque/force</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>TimeLimit</td>
<td>Maximal allowed time for the whole algorithm [s]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>DistanceLimit</td>
<td>Maximal allowed distance for the whole algorithm [unit]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Position</td>
<td>Requested target position (absolute) [unit]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Direction</td>
<td>Direction of movement (cyclic axis or special case only)</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>HomingMode</td>
<td>Homing mode algorithm</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxis</td>
<td>Axis reference (only RM_Axis.axisRef−uAxis or yAxis−uAxis</td>
<td>Reference</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>Bool</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
<td>Bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>Bool</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
<td>Bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>Bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>Error</td>
</tr>
</tbody>
</table>

i ..... REXYGEN general error
MC_MoveAbsolute, MCP_MoveAbsolute – Move to position (absolute coordinate)

Block Symbols

Licence: MOTION CONTROL

Function Description

The MC_MoveAbsolute and MCP_MoveAbsolute blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP version of the block.

The MC_MoveAbsolute block moves an axis to specified position as fast as possible. If no further action is pending, final velocity is zero (axis moves to position and stops) otherwise it depends on blending mode. For blending purposes, start and stop velocity of this block is maximum velocity with direction respecting current and final position. If start velocity of next pending block is in opposite direction, then blending velocity is always zero.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

The MC_MoveRelative block act almost same as MC_MoveAbsolute. The only difference is the final position is computed adding input Distance to current (when rising edge on input Execute occurred) position.

The MC_MoveAdditive block act almost same as MC_MoveRelative. The only difference is the final position is computed adding input Distance to final position of the previous block.

The MC_MoveSuperimposed block acts almost the same as the MC_MoveRelative
block. The only difference is the current move is not aborted and superimposed move is executed immediately and added to current move. Original move act like superimposed move is not run.

The following table describes all inputs, parameters and outputs which are used in some of the blocks in the described block suite.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>AAxis reference (only RM.Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>Position</td>
<td>Requested target position (absolute) [unit]</td>
</tr>
<tr>
<td>Velocity</td>
<td>Maximal allowed velocity [unit/s]</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s²]</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
</tr>
<tr>
<td>1</td>
<td>Aborting (start immediately)</td>
</tr>
<tr>
<td>2</td>
<td>Buffered (start after finish of previous motion)</td>
</tr>
<tr>
<td>3</td>
<td>Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
</tr>
<tr>
<td>4</td>
<td>Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
</tr>
<tr>
<td>5</td>
<td>Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)</td>
</tr>
<tr>
<td>6</td>
<td>Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)</td>
</tr>
<tr>
<td>Direction</td>
<td>Direction of movement (cyclic axis or special case only)</td>
</tr>
<tr>
<td>1</td>
<td>Positive</td>
</tr>
<tr>
<td>2</td>
<td>Shortest</td>
</tr>
<tr>
<td>3</td>
<td>Negative</td>
</tr>
<tr>
<td>4</td>
<td>Current</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxis</td>
<td>Axis reference (only RM.Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>i ......</td>
<td>REXYGEN general error</td>
</tr>
</tbody>
</table>
Example

![Example Diagram]

- **Execute** - block 1
- **Active** - block 1
- **Done** - block 1
- **Execute** - block 2
- **Active** - block 2
- **Done** - block 2

**Commanded velocity**

**Commanded position**

![Commanded Velocity Chart]

![Commanded Position Chart]
MC_MoveAdditive, MCP_MoveAdditive – Move to position (relative to previous motion)

Block Symbols

Function Description

The MC_MoveAdditive and MCP_MoveAdditive blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP version of the block.

The MC_MoveAdditive block moves an axis to specified position as fast as possible. The final position is determined by adding the value of Distance parameter to final position of previous motion block which was controlling the axis. If no further action is pending, final velocity is zero (axis moves to position and stops) otherwise it depends on blending mode. For blending purposes, start and stop velocity of this block is maximum velocity with direction respecting current and final position. If start velocity of next pending block is in opposite direction, then blending velocity is always zero.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis Reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>Distance</td>
<td>Requested target distance (relative to start point) [unit]</td>
</tr>
</tbody>
</table>

Licence: MOTION CONTROL
502 CHAPTER 18. MC_SINGLE - MOTION CONTROL - SINGLE AXIS BLOCKS

Velocity  Maximal allowed velocity [unit/s]  Double (F64)
Acceleration Maximal allowed acceleration [unit/s²]  Double (F64)
Deceleration Maximal allowed deceleration [unit/s²]  Double (F64)
Jerk Maximal allowed jerk [unit/s³]  Double (F64)
BufferMode Buffering mode  Long (I32)

1 . . . . Aborting (start immediately)
2 . . . . Buffered (start after finish of previous motion)
3 . . . . Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
4 . . . . Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
5 . . . . Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)
6 . . . . Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)

Outputs

yAxis Axis reference (only RM_Axis.axisRef=yAxis or yAxis=yAxis connections are allowed)  Reference
Done Algorithm finished  Bool
CommandAborted Algorithm was aborted  Bool
Busy Algorithm not finished yet  Bool
Active The block is controlling the axis  Bool
Error Error occurred  Bool
ErrorID Result of the last operation  Error
i . . . . REXYGEN general error
Example
MC_MoveRelative, MCP_MoveRelative – Move to position (relative to execution point)

Block Symbols

Function Description

The MC_MoveRelative and MCP_MoveRelative blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_version of the block.

The MC_MoveRelative block moves an axis to specified position as fast as possible. The final position is determined by adding the value of Distance parameter to the actual position at the moment of triggering the Execute input. If no further action is pending, final velocity is zero (axis moves to position and stops) otherwise it depends on blending mode. For blending purposes, start and stop velocity of this block is maximum velocity with direction respecting current and final position. If start velocity of next pending block is in opposite direction, then blending velocity is always zero.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

Inputs

- uAxis : Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis Reference connections are allowed)
- Execute : The block is activated on rising edge
- Distance : Requested target distance (relative to execution point) [unit]
Velocity  Maximal allowed velocity [unit/s]          Double (F64)
Acceleration Maximal allowed acceleration [unit/s²] Double (F64)
Deceleration Maximal allowed deceleration [unit/s²] Double (F64)
Jerk Maximal allowed jerk [unit/s³]                  Double (F64)
BufferMode Buffering mode

1 ...... Aborting (start immediately)
2 ...... Buffered (start after finish of previous motion)
3 ...... Blending low (start after finishing the previous
        motion, previous motion finishes with the lowest
        velocity of both commands)
4 ...... Blending high (start after finishing the previous
        motion, previous motion finishes with the lowest
        velocity of both commands)
5 ...... Blending previous (start after finishing the previous
        motion, previous motion finishes with its final
        velocity)
6 ...... Blending next (start after finishing the previous
        motion, previous motion finishes with the starting
        velocity of the next block)

Outputs

yAxis    Axis reference (only RMAxis.axisRef-uAxis or yAxis-uAxis connections are allowed) Reference
Done     Algorithm finished                        Bool
CommandAborted Algorithm was aborted              Bool
Busy     Algorithm not finished yet                Bool
Active   The block is controlling the axis         Bool
Error    Error occurred                            Bool
ErrorID  Result of the last operation              Error

i ...... REXYGEN general error
Example
MC_MoveSuperimposed, MCP_MoveSuperimposed — Superimposed move

Block Symbols

Function Description

The MC_MoveSuperimposed and MCP_MoveSuperimposed blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP version of the block.

The MC_MoveSuperimposed block moves an axis to specified position as fast as possible (with respect to set limitations). Final position is specified by input parameter Distance. In case that the axis is already in motion at the moment of execution of the MC_MoveSuperimposed block, the generated values of position, velocity and acceleration are added to the values provided by the previous motion block. If there is no previous motion, the block behaves in the same way as the MC_MoveRelative command.

Note: There is no BufferMode parameter which is irrelevant in the superimposed mode. If there is already an superimposed motion running at the moment of execution, the new block is started immediately (analogous to aborting mode).

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis Reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>Distance</td>
<td>Requested target distance (relative to execution point) [unit]</td>
</tr>
<tr>
<td>VelocityDiff</td>
<td>Maximal allowed velocity [unit/s]</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s²]</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
</tr>
</tbody>
</table>
 Outputs

*yAxis* Axis reference (only `RM_Axis.axisRef-uAxis` or `yAxis-uAxis` connections are allowed)

*Done* Algorithm finished          *Bool*

*CommandAborted* Algorithm was aborted           *Bool*

*Busy* Algorithm not finished yet           *Bool*

*Active* The block is controlling the axis           *Bool*

*Error* Error occurred           *Error*

*ErrorID* Result of the last operation           *Error*

  i . . . . REXYGEN general error
Example
MC_MoveContinuousAbsolute, MCP_MoveContinuousAbsolute — Move to position (absolute coordinate)

Function Description

The MC_MoveContinuousAbsolute and MCP_MoveContinuousAbsolute blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP version of the block.

The MC_MoveContinuousAbsolute block moves an axis to specified position as fast as possible. If no further action is pending, final velocity is specified by parameter EndVelocity. For blending purposes, start and stop velocity of this block is maximum velocity with direction respecting current and final position. If start velocity of next pending block is in opposite direction, then blending velocity is always zero.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

Note 1: If the EndVelocity is set to zero value, the block behaves in the same way as MC_MoveAbsolute.

Note 2: If next motion command is executed before the final position is reached, the block behaves in the same way as MC_MoveAbsolute.
Inputs

uAxis Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)
Execute The block is activated on rising edge Bool
Position Requested target position (absolute) [unit] Double (F64)
Velocity Maximal allowed velocity [unit/s] Double (F64)
Acceleration Maximal allowed acceleration [unit/s²] Double (F64)
Deceleration Maximal allowed deceleration [unit/s²] Double (F64)
Jerk Maximal allowed jerk [unit/s³] Double (F64)
BufferMode Buffering mode Long (I32)

1 . . . . Aborting (start immediately)
2 . . . . Buffering (start after finish of previous motion)
3 . . . . Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
4 . . . . Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
5 . . . . Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)
6 . . . . Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)

Direction Direction of movement (cyclic axis or special case only) Long (I32)

1 . . . . Positive
2 . . . . Shortest
3 . . . . Negative
4 . . . . Current

EndVelocity End velocity Double (F64)

Outputs

yAxis Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)
InEndVelocity Algorithm finished Bool
CommandAborted Algorithm was aborted Bool
Busy Algorithm not finished yet Bool
Active The block is controlling the axis Bool
Error Error occurred Bool
ErrorID Result of the last operation Error

i . . . . REXYGEN general error
**MC_MoveContinuousRelative, MCP_MoveContinuousRelative – Move to position (relative to previous motion)**

**Block Symbols**

License: **MOTION CONTROL**

**Function Description**

The `MC_MoveContinuousRelative` and `MCP_MoveContinuousRelative` blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the `MCP_` version of the block.

The `MC_MoveContinuousRelative` block moves an axis to specified position as fast as possible. The final position is determined by adding the value of `Distance` parameter to the actual position at the moment of triggering the `Execute` input. If no further action is pending, final velocity is specified by parameter `EndVelocity`. For blending purposes, start and stop velocity of this block is maximum velocity with direction respecting current and final position. If start velocity of next pending block is in opposite direction, then blending velocity is always zero.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input `Jerk` is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

Note 1: If the `EndVelocity` is set to zero value, the block behaves in the same way as `MC_MoveRelative`.

Note 2: If next motion command is executed before the final position is reached, the block behaves in the same way as `MC_MoveRelative`.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input `Jerk`
is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)</td>
<td>Reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>Bool</td>
</tr>
<tr>
<td>Distance</td>
<td>Requested target distance (relative to execution point) [unit]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Velocity</td>
<td>Maximal allowed velocity [unit/s]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s²]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
BufferMode Buffering mode

1 ..... Aborting (start immediately)
2 ..... Buffered (start after finish of previous motion)
3 ..... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
4 ..... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
5 ..... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)
6 ..... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)

EndVelocity End velocity

Outputs

yAxis Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis Reference connections are allowed)
InEndVelocity PLCopen Done (algorithm finished) Bool
CommandAborted PLCopen CommandAborted (algorithm was aborted) Bool
Busy PLCopen Busy (algorithm not finished yet) Bool
Active PLCopen Active (the block is controlling the axis) Bool
Error PLCopen Error (error occurred) Bool
ErrorID Result of the last operation Error
i ..... REXYGEN general error
Example
MC_MoveVelocity, MCP_MoveVelocity – Move with constant velocity

Block Symbols

Function Description

The MC_MoveVelocity and MCP_MoveVelocity blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP version of the block.

The MC_MoveVelocity block changes axis velocity to specified value as fast as possible and keeps the specified velocity until the command is aborted by another block or event.

Note: parameter Direction enumerate also shortest_way although for this block it is not valid value.

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis Reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>Velocity</td>
<td>Maximal allowed velocity [unit/s]</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration unit/s²]</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
</tr>
<tr>
<td>Direction</td>
<td>Direction of movement (cyclic axis or special case only)</td>
</tr>
</tbody>
</table>

1 ......  Positive
2 ......  Shortest
3 ......  Negative
4 ......  Current

Licence: MOTION CONTROL
BufferMode Buffering mode Long (I32)

1 ...... Aborting (start immediately)
2 ...... Buffered (start after finish of previous motion)
3 ...... Blending low (start after finishing the previous
motion, previous motion finishes with the lowest
velocity of both commands)
4 ...... Blending high (start after finishing the previous
motion, previous motion finishes with the lowest
velocity of both commands)
5 ...... Blending previous (start after finishing the previous
motion, previous motion finishes with its final
velocity)
6 ...... Blending next (start after finishing the previous
motion, previous motion finishes with the starting
velocity of the next block)

Outputs

yAxis Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis Reference
connections are allowed)
InVelocity Requested velocity reached Bool
CommandAborted Algorithm was aborted Bool
Busy Algorithm not finished yet Bool
Active The block is controlling the axis Bool
Error Error occurred Bool
ErrorID Result of the last operation Error

i ...... REXYGEN general error
Execute block 1

Active block 1

Done block 1

Execute block 2

Active block 2

Done block 2

Commanded velocity

Commanded position
MC_PositionProfile, MCP_PositionProfile – Position profile

Block Symbols

Function Description

The MC_PositionProfile and MCP_PositionProfile blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The MC_PositionProfile block commands a time-position locked motion profile. Block implements two possibilities for definition of time-position function:

1. sequence of values: the user defines a sequence of time-position pairs. In each time interval, the values of position are interpolated. Times sequence is in array times, position sequence is in array values. Time sequence must be increasing and must start with zero or zero must be between the first and last point. Execution always starts from zero time, so if the sequence start with negative time, part of the profile is not executed (could be used for debugging or time shift). For MC_VelocityProfile and MC_AccelerationProfile interpolation is linear, but for MC_PositionProfile, 3rd order polynomial is used in order to avoid steps in velocity.

2. spline: time sequence is the same as in previous case. Each interval is interpolate by the 5th order polynomial \( p(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \) where beginning of the time-interval is for \( x = 0 \), end of time-interval is for \( x = 1 \) and factors \( a_i \) are put in array values in ascending order (e.g. array values contains 6 values for each interval). This method allows smaller number of intervals and there is special editor for synthesis of the interpolating spline function.

For both case, the time sequence could be equally spaced and then array times includes only the first (usually zero) and last point.

Note 1: input TimePosition is missing, because all path data are in parameters of the block.

Note 2: parameter values must be set as vector in all cases, e.g. text string must not include semicolon.
Note 3: incorrect parameter cSeg (higher than real size of arrays times and/or values) leads to unpredictable result and in some cases crashes whole runtime execution (the problem is platform dependent and currently it is known only for SIMULINK - crash of whole MATLAB).

Note 4: in the spline mode, polynomial is always 5th order and always in position (also for sibling block MC_VelocityProfile and MC_AccelerationProfile) and it couldn’t be changed. As the special editor exists, this is not important limitation.

Note 5: The block does not include ramp-in mode. If start position and/or velocity of profile is different from actual (commanded) position of axis, block fails with error -707 (step). It is recommended to use BufferMode=BlendingNext to eliminate the problem with start velocity.

**Inputs**

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis Reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>TimeScale</td>
<td>Overall scale factor in time</td>
</tr>
<tr>
<td>PositionScale</td>
<td>Overall scale factor in value</td>
</tr>
<tr>
<td>Offset</td>
<td>Overall profile offset in value</td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
</tr>
<tr>
<td>1</td>
<td>Aborting (start immediately)</td>
</tr>
<tr>
<td>2</td>
<td>Buffered (start after finish of previous motion)</td>
</tr>
<tr>
<td>3</td>
<td>Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
</tr>
<tr>
<td>4</td>
<td>Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
</tr>
<tr>
<td>5</td>
<td>Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)</td>
</tr>
<tr>
<td>6</td>
<td>Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM_Axis.axisRef-yAxis or yAxis-yAxis Reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td>i</td>
<td>REXYGEN general error</td>
</tr>
</tbody>
</table>
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>alg</strong></td>
<td>Algorithm for interpolation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 ....  Sequence of time/value pairs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ....  Sequence of equidistant values</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 ....  Spline</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 ....  Equidistant spline</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cSeg</strong></td>
<td>Number of profile segments</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>times</strong></td>
<td>Times when segments are switched</td>
<td>[0 30]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>values</strong></td>
<td>Values or interpolating polynomial coefficients (a0, a1, a2, ...)</td>
<td>[0 100 100 50]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example
MC_Power – Axis activation (power on/off)

Function Description

The MC_Power block must be used with all axes. It is the only way to switch an axis from disable state to standstill (e.g. operation) state. The Enable input must be set (non zero value) for whole time the axis is active. The Status output can be used for switch on and switch off of the motor driver (logical signal for enabling the power stage of the drive).

The block does not implement optional parameters/inputs Enable_Positive, Enable_Negative. The same functionality can be implemented by throwing the limit switches (inputs limP and limN of block RM_Axis).

If the associated axis is turned off (by setting the Enable input to zero) while a motion is processed (commanded velocity is not zero), error stoping sequence is activated and the status is switched to off/disabled when the motion stops (commanded velocity reaches zero value).

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis) Reference connections are allowed</td>
</tr>
<tr>
<td>Enable</td>
<td>Block function is enabled</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxis</td>
<td>Axis reference (only RM_Axis.axisRef-yAxis or yAxis-yAxis) Reference connections are allowed</td>
</tr>
<tr>
<td>Status</td>
<td>Effective state of the power stage</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td></td>
<td>i .. . REXYGEN general error</td>
</tr>
</tbody>
</table>
MC_ReadActualPosition – Read actual position

A block in the library for motion control (MOTION CONTROL) is called MC_ReadActualPosition that reads the actual position of an axis.

Function Description

The block MC_ReadActualPosition displays the actual value of the position of a connected axis on the output Position. The output is valid only while the block is enabled by the logical input signal Enable.

The block displays logical position values which are entered into all of the motion blocks as position inputs. In case that no absolute position encoder is used or the internal position is set in another way (e.g. via MC_Home block), the CommandedPosition output of the corresponding RM_Axis may display different values.

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef→uAxis or yAxis→uAxis) Reference connections are allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Block function is enabled</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM_Axis.axisRef→uAxis or yAxis→uAxis) Reference connections are allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>Output value is valid</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td></td>
<td>i ...... REXYGEN general error</td>
</tr>
<tr>
<td>Position</td>
<td>Actual absolute position</td>
</tr>
</tbody>
</table>

The data types are Boolean (Bool), Double (F64).
MC_ReadAxisError - Read axis error

Block Symbol

Licence: MOTION CONTROL

Function Description

The block MC_ReadAxisError displays actual error code of a connected axis on the output AxisErrorID. In case of no error, the output is set to zero. The error value is valid only while the block is enabled by the logical input signal Enable. This block is implemented for sake of compatibility with PLCopen specification as it displays duplicit information about an error which is also accessible on the ErrorID output of the RM_Axis block.

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Block function is enabled</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>Output value is valid</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td>AxisErrorID</td>
<td>Result of the last operation read from axis</td>
</tr>
</tbody>
</table>

i .... REXYGEN general error
MC_ReadBoolParameter – Read axis parameter (bool)

Block Symbol

License: MOTION CONTROL

Function Description

The block MC_ReadBoolParameter displays actual value of various signals related to the connected axis on its Value output. The user chooses from a set of accessible logical variables by setting the ParameterNumber input. The output value is valid only while the block is activated by the logical Enable input.

The block displays the parameters and outputs of RM_Axis block and is implemented for sake of compatibility with the PLCopen specification.

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Block function is enabled</td>
</tr>
<tr>
<td>ParameterNumber</td>
<td>Parameter ID</td>
</tr>
<tr>
<td></td>
<td>4      ... Enable sw positive limit</td>
</tr>
<tr>
<td></td>
<td>5      ... Enable sw negative limit</td>
</tr>
<tr>
<td></td>
<td>6      ... Enable position Iq monitoring</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>Output value is valid</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td></td>
<td>i ... REXYGEN general error</td>
</tr>
<tr>
<td>Value</td>
<td>Parameter value</td>
</tr>
</tbody>
</table>
MC_ReadParameter – Read axis parameter

Block Symbol

Function Description

The block MC_ReadParameter displays actual value of various system variables of the connected axis on its Value output. The user chooses from a set of accessible variables by setting the ParameterNumber input. The output value is valid only while the block is activated by the logical Enable input.

The block displays the parameters and outputs of RM_Axis block and is implemented for sake of compatibility with the PLCOpen specification.

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Block function is enabled</td>
</tr>
<tr>
<td>ParameterNumber</td>
<td>Parameter ID</td>
</tr>
<tr>
<td>1</td>
<td>Commanded position</td>
</tr>
<tr>
<td>2</td>
<td>Positive sw limit switch</td>
</tr>
<tr>
<td>3</td>
<td>Negative sw limit switch</td>
</tr>
<tr>
<td>7</td>
<td>Maximal position tag</td>
</tr>
<tr>
<td>8</td>
<td>Maximal velocity (system)</td>
</tr>
<tr>
<td>9</td>
<td>Maximal velocity (appl)</td>
</tr>
<tr>
<td>10</td>
<td>Actual velocity</td>
</tr>
<tr>
<td>11</td>
<td>Commanded velocity</td>
</tr>
<tr>
<td>12</td>
<td>Maximal acceleration (system)</td>
</tr>
<tr>
<td>13</td>
<td>Maximal acceleration (appl.)</td>
</tr>
<tr>
<td>14</td>
<td>Maximal deceleration (system)</td>
</tr>
<tr>
<td>15</td>
<td>Maximal deceleration (appl.)</td>
</tr>
<tr>
<td>16</td>
<td>Maximal jerk</td>
</tr>
<tr>
<td>1000</td>
<td>Actual position</td>
</tr>
<tr>
<td>1001</td>
<td>Maximal torque/force</td>
</tr>
<tr>
<td>1003</td>
<td>Actual torque/force</td>
</tr>
<tr>
<td>1004</td>
<td>Commanded torque/force</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM.Axis.axisRef–uAxis or yAxis–uAxis) Reference connections are allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>Output value is valid</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td></td>
<td>i ...... REXYGEN general error</td>
</tr>
<tr>
<td>Value</td>
<td>Parameter value</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MC_ReadStatus – Read axis status

Function Description

The block MC_ReadStatus indicates the state of the connected axis on its logical output signals. The values of the states are valid only while the Enable input is set to nonzero value. This state is indicated by Valid output.

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis Reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Block function is enabled</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis Reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>Output value is valid</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td>ErrorStop</td>
<td>Axis is in the ErrorStop state</td>
</tr>
<tr>
<td>Disabled</td>
<td>Axis is in the Disabled state</td>
</tr>
<tr>
<td>Stopping</td>
<td>Axis is in the Stopping state</td>
</tr>
<tr>
<td>StandStill</td>
<td>Axis is in the StandStill state</td>
</tr>
<tr>
<td>DiscreteMotion</td>
<td>Axis is in the DiscreteMotion state</td>
</tr>
<tr>
<td>ContinuousMotion</td>
<td>Axis is in the ContinuousMotion state</td>
</tr>
<tr>
<td>SynchronizedMotion</td>
<td>Axis is in the SynchronizedMotion state</td>
</tr>
<tr>
<td>Homing</td>
<td>Axis is in the Homing state</td>
</tr>
</tbody>
</table>

| ErrorStop     | Axis is in the ErrorStop state                                                      |
| Disabled      | Axis is in the Disabled state                                                       |
| Stopping      | Axis is in the Stopping state                                                       |
| StandStill    | Axis is in the StandStill state                                                     |
| DiscreteMotion| Axis is in the DiscreteMotion state                                                 |
| ContinuousMotion| Axis is in the ContinuousMotion state                                               |
| SynchronizedMotion| Axis is in the SynchronizedMotion state                                           |
| Homing        | Axis is in the Homing state                                                         |
Constant Velocity  Axis is moving with constant velocity  Bool
Accelerating  Axis is accelerating  Bool
Decelerating  Axis is decelerating  Bool
MC_Reset – Reset axis errors

Block Symbol

Licence: MOTION CONTROL

Function Description

The MC_Reset block makes the transition from the state ErrorStop to StandStill by resetting all internal axis-related errors.

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef→uAxis or yAxis→uAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM_Axis.axisRef→uAxis or yAxis→uAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
</tbody>
</table>
                   ål Error

i . . . . REXYGEN general error
MC_SetOverride, MCP_SetOverride – Set override factors

Block Symbols

Function Description

The MC_SetOverride and MCP_SetOverride blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_version of the block.

The MC_SetOverride block sets the values of override for the whole axis, and all functions that are working on that axis. The override parameters act as a factor that is multiplied to the commanded velocity, acceleration, deceleration and jerk of the move function block.

This block is level-sensitive (not edge-sensitive like other motion control blocks). So factors are update in each step while input Enable is not zero. It leads to recalculation of movement’s path if a block like MC_MoveAbsolute commands the axis. This recalculation needs lot of CPU time and also numerical problem could appear. For this reasons, a deadband (parameter diff) is established. The movement’s path recalculation is proceeded only if one of the factors is changed more then the deadband.

Note: all factor must be positive. Factor greater then 1.0 are possible, but often lead to overshooting of axis limits and failure of movement (with errorID=-700 - invalid parameter; if factor is set before start of block) or error stop of axis (with errorID=-701 - out of range; if factor is changed within movement and actual value overshoot limit).

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef→uAxis or yAxis→uAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Block function is enabled</td>
</tr>
<tr>
<td>VelFactor</td>
<td>Velocity multiplication factor</td>
</tr>
<tr>
<td>AccFactor</td>
<td>Acceleration/deceleration multiplication factor</td>
</tr>
<tr>
<td>JerkFactor</td>
<td>Jerk multiplication factor</td>
</tr>
</tbody>
</table>

Reference

Bool
Double (F64)
Double (F64)
Double (F64)
Outputs

**yAxis**  
Axis reference (only RM_Axis.axisRef or yAxisRef or yAxis)  
Reference connections are allowed

**Enabled**  
Block function is enabled  
Bool

**Busy**  
Algorithm not finished yet  
Bool

**Error**  
Error occurred  
Bool

**ErrorID**  
Result of the last operation  
Error

i ..... REXYGEN general error

Parameter

**diff**  
Deadband (difference for recalculation)  
\[ 0.0 \uparrow 1.0 \downarrow 0.1 \]  
Double (F64)
MC_Stop, MCP_Stop – Stopping a movement

Block Symbols

Function Description

The MC_Stop and MCP_Stop blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The MC_Stop block commands a controlled motion stop and transfers the axis to the state Stopping. It aborts any ongoing Function Block execution. While the axis is in state Stopping, no other FB can perform any motion on the same axis. After the axis has reached velocity zero, the Done output is set to true immediately. The axis remains in the state Stopping as long as Execute is still true or velocity zero is not yet reached. As soon as Done=true and Execute=false the axis goes to state StandStill.

Note 1: parameter/input BufferMode is not supported. Mode is always Aborting

Note 2: Failing stop-command could be dangerous. This block does not generate invalid-parameter-error but tries to stop the axis anyway (e.g. uses parameters from RM_Axis or generates error-stop-sequence).

Inputs

- uAxis Axis reference (only RM_Axis.axisRef_uAxis or yAxis_uAxis) Reference connections are allowed
- Execute The block is activated on rising edge Bool
- Deceleration Maximal allowed deceleration [unit/s²] Double (F64)
- Jerk Maximal allowed jerk [unit/s³] Double (F64)

Outputs

- yAxis Axis reference (only RM_Axis.axisRef_uAxis or yAxis_uAxis) Reference connections are allowed
- Done Algorithm finished Bool
- CommandAborted Algorithm was aborted Bool
- Busy Algorithm not finished yet Bool
- Active The block is controlling the axis Bool
- Error Error occurred Bool
<table>
<thead>
<tr>
<th>ErrorID</th>
<th>Result of the last operation</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>i ......</td>
<td>REXYGEN general error</td>
<td></td>
</tr>
</tbody>
</table>
MC_TorqueControl, MCP_TorqueControl – Torque/force control

Block Symbols

Function Description

The MC_TorqueControl and MCP_TorqueControl blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP version of the block.

The MCP_TorqueControl block generates constant slope torque/force ramp until maximum requested value has been reached. Similar profile is generated for velocity. The motion trajectory is limited by maximum velocity, acceleration / deceleration, and jerk, or by the value of the torque, depending on the mechanical circumstances.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Axis reference (only RM_Axis.axisRef−uAxis or yAxis−uAxis) Reference connections are allowed</td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>Bool</td>
</tr>
<tr>
<td>Torque</td>
<td>Maximal allowed torque/force</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>TorqueRamp</td>
<td>Maximal allowed torque/force ramp</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Velocity</td>
<td>Maximal allowed velocity [unit/s]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s²]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Direction</td>
<td>Direction of movement (cyclic axis or special case only)</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>

1 ..... Positive
2 ..... Shortest
3 ..... Negative
4 ..... Current
BufferMode Buffering mode

1 . . . . Aborting (start immediately)
2 . . . . Buffered (start after finish of previous motion)
3 . . . . Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
4 . . . . Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
5 . . . . Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)
6 . . . . Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)

Outputs

yAxis Axis reference (only RM.Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)
InTorque Requested torque/force is reached
CommandAborted Algorithm was aborted
Busy Algorithm not finished yet
Active The block is controlling the axis
Error Error occurred
ErrorID Result of the last operation

i . . . . REXYGEN general error

Parameter

kma Torque/force to acceleration ratio

Double (F64)
540 CHAPTER 18. MC_SINGLE – MOTION CONTROL - SINGLE AXIS BLOCKS

Example
MC_VelocityProfile, MCP_VelocityProfile – Velocity profile

Function Description

The MC_PositionProfile block commands a time-position locked motion profile. Block implements two possibilities for definition of time-velocity function:

1. sequence of values: the user defines a sequence of time-velocity pairs. In each time interval, the values of velocity are interpolated. Times sequence is in array times, position sequence is in array values. Time sequence must be increasing and must start with zero or zero must be between the first and last point. Execution always starts from zero time, so if the sequence start with negative time, part of the profile is not executed (could be used for debugging or time shift). For MC_VelocityProfile and MC_AccelerationProfile interpolation is linear, but for MC_PositionProfile, 3rd order polynomial is used in order to avoid steps in velocity.

2. spline: time sequence is the same as in previous case. Each interval is interpolated by 5th order polynomial $p(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$ where beginning of the time-interval is for $x = 0$, end of time-interval is for $x = 1$ and factors $a_i$ are put in array values in ascending order (e.g. array values contains 6 values for each interval). This method allows smaller number of intervals and there is special editor for synthesis of the interpolating spline function.

For both case, the time sequence could be equally spaced and then array times includes only the first (usually zero) and last point.

Note 1: input TimePosition is missing, because all path data are in parameters of the block.

Note 2: parameter values must be set as vector in all cases, e.g. text string must not include semicolon.

Note 3: incorrect parameter cSeg (higher then real size of arrays times and/or values) leads to unpredictable result and in some cases crashes whole runtime execution (The problem is platform dependent and currently it is known only for SIMULINK - crash of whole MATLAB).
542 CHAPTER 18. MC_SINGLE - MOTION CONTROL - SINGLE AXIS BLOCKS

Note 4: in the spline mode, polynomial is always 5th order and always in position (also for sibling block MC_PositionProfile and MC_AccelerationProfile) and it couldn’t be changed. As the special editor exists, this is not important limitation.

Note 5: The block does not include ramp-in mode. If start position and/or velocity of profile is different from actual (commanded) position of axis, block fails with error -707 (step). It is recommended to use BufferMode=BlendingNext to eliminate the problem with start velocity.

**Inputs**

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>TimeScale</td>
<td>Overall scale factor in time</td>
</tr>
<tr>
<td>VelocityScale</td>
<td>Overall scale factor in value</td>
</tr>
<tr>
<td>Offset</td>
<td>Overall profile offset in value</td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
</tr>
<tr>
<td>1 . . . .</td>
<td>Aborting (start immediately)</td>
</tr>
<tr>
<td>2 . . . .</td>
<td>Buffered (start after finish of previous motion)</td>
</tr>
<tr>
<td>3 . . . .</td>
<td>Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
</tr>
<tr>
<td>4 . . . .</td>
<td>Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
</tr>
<tr>
<td>5 . . . .</td>
<td>Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)</td>
</tr>
<tr>
<td>6 . . . .</td>
<td>Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM_Axis.axisRef-yAxis or yAxis-yAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm not finished</td>
</tr>
<tr>
<td>Busy</td>
<td>The block is controlling the axis</td>
</tr>
<tr>
<td>Active</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td>i . . . .</td>
<td>REXYGEN general error</td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>alg</code></td>
<td>Algorithm for interpolation</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1 . . . . Sequence of time/value pairs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 . . . . Sequence of equidistant values</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 . . . . Spline</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 . . . . Equidistant spline</td>
<td></td>
</tr>
<tr>
<td><code>cSeg</code></td>
<td>Number of profile segments</td>
<td>3</td>
</tr>
<tr>
<td><code>times</code></td>
<td>Times when segments are switched</td>
<td>[15, 25, 30]</td>
</tr>
<tr>
<td><code>values</code></td>
<td>Values or interpolating polynomial coefficients (a0, a1, a2, ...)</td>
<td>[0, 100, 100, 50]</td>
</tr>
</tbody>
</table>
Example
MC_WriteBoolParameter – **Write axis parameter** (bool)

**Block Symbol**

```
  uAxis  yAxis
  Execute Busy
  ParameterNumber
  Value  Error
  ErrorID
```

**Licence:** MOTION CONTROL

**Function Description**

The block `MC_WriteBoolParameter` writes desired value of various system parameters entered on its **Value** input to the connected axis. The user chooses from a set of accessible logical variables by setting the **ParameterNumber** input.

The block is implemented for sake of compatibility with the PLCopen specification as the parameters can be written by the `SETPB` block.

**Inputs**

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>ParameterNumber</td>
<td>Parameter ID</td>
</tr>
<tr>
<td>Value</td>
<td>Parameter value</td>
</tr>
</tbody>
</table>

| yAxis | Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed) |

**Outputs**

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
</tbody>
</table>

i ...... REXYGEN general error
**MC_WriteParameter** – Write axis parameter

**Block Symbol**

<table>
<thead>
<tr>
<th>uAxis</th>
<th>yAxis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Done</td>
</tr>
<tr>
<td>ParameterNumber</td>
<td>Busy</td>
</tr>
<tr>
<td>Value</td>
<td>Error</td>
</tr>
<tr>
<td>ErrorID</td>
<td></td>
</tr>
</tbody>
</table>

**MC_WriteParameter**

**Function Description**

The block **MC_WriteParameter** writes desired value of various system parameters entered on its **Value** input to the connected axis. The user chooses from a set of accessible variables by setting the **ParameterNumber** input.

The block is implemented for sake of compatibility with the **PLCopen** specification as the parameters can be written by the **SETPR** block.

**Inputs**

- **uAxis**  
  Axis reference (only **RM_Axis.axisRef-uAxis** or **yAxis-uAxis** connections are allowed)
- **Execute**  
  The block is activated on rising edge  
  **Bool**
- **ParameterNumber**  
  Parameter ID  
  2 . . . Positive sw limit switch
  3 . . . Negative sw limit switch
  7 . . . Maximal position lag
  8 . . . Maximal velocity (system)
  9 . . . Maximal velocity (appl.)
  13 . . . Maximal acceleration (appl.)
  15 . . . Maximal deceleration (appl.)
  16 . . . Maximal jerk
  1001 . . . Maximal torque/force
- **Value**  
  Parameter value  
  **Double (F64)**

**Outputs**

- **yAxis**  
  Axis reference (only **RM_Axis.axisRef-uAxis** or **yAxis-uAxis** connections are allowed)
- **Done**  
  Algorithm finished  
  **Bool**
- **Busy**  
  Algorithm not finished yet  
  **Bool**
- **Error**  
  Error occurred  
  **Bool**
- **ErrorID**  
  Result of the last operation
  - i . . . REXYGEN general error
RM_AxisOut – Axis output

Function Description

The RM_AxisOut block allows an access to important states of block RM_Axis. Same outputs are also available directly on RM_Axis (some of them), but this direct output is one step delayed. Blocks are ordered for execution by flow of a signal, so RM_Axis is first then all motion blocks (that actualize RM_Axis state), then RM_AxisOut (should be last) and finally waiting for next period.

Note 1: Control system REXYGEN orders blocks primary by flow of signal, secondarily by name of block (ascendent in alphabetical order), so name like "zzz" is good choice. For checking the order, you can use REXYGEN Diagnostics tool where the blocks are sorted by execution order.

Note 2: almost all blocks do not work with torque so commanded torque is 0. Commanded acceleration and torque should be used as feed-forward value for position/velocity controller so this value does not make any problem.

Inputs

uAxis axis reference that must be connected to axisRef of the RM_Axis Reference block (direct or indirect throw output Axis of some other block)
RM_AxisSpline – Commanded values interpolation

Function Description

There are lot of motion control blocks which implement complicated algorithms so they require bigger sampling period (typical update rate is from 10 to 200ms). On the other side, the motor driver usually requires small sampling period for smooth/waveless movement. The RM_AxisSpline block solves this problem of multirate execution of motion planning and motion control levels. The block can run in different task than other motion control blocks with highest sampling period possible. It interpolates commanded position, velocity and torque and generates smooth curve which is more suited for motor driver controllers.

There are two possibilities of connection to RM_Axis block: connect all necessary values (outputs of the block RM_AxisOut) as input of interpolating block or use only axis reference and read the state directly. This block uses axis reference. For correct synchronization between two tasks, the block RM_Axis must be executed first followed by all axis related motion control blocks and finally by block RM_AxisOut at the end.

Note 1: For interpolation of position signal, 3rd order polynomial \(p(t)\) is used, where \(p_0(0) = pos0, p_0(t_S) = pos1, \frac{dp_0(t)}{dt}_{t=0} = vel0, \frac{dp_0(t)}{dt}_{t=t_S} = vel1\). To interpolate velocity, also an 3rd order polynomial \(p_v(t)\) is used, where \(p_v(0) = vel0, p_v(t_S) = vel1, \frac{dp_v(t)}{dt}_{t=0} = acc0, \frac{dp_v(t)}{dt}_{t=t_S} = acc1\). Torque is interpolated by linear function.

Note 2: Because the time of execution of motion blocks is varying in time, the block uses one or two step prediction for interpolation depending on actual conditions and timing of the motion blocks in slower tasks. The use of predicted values is signalized by states RUN0, RUN1, RUN2.

Note 3: Control system REXYGEN orders the blocks primarily by flow of signal, secondarily by name of block (ascendant in alphabetical order), so name like "zzz" is good choice for the block RM_AxisOut. For checking the order, you can use REXYGEN Diagnostics tool where the blocks are sorted by execution order.
RM_Track – Tracking and inching

Block Symbol

License: MOTION CONTROL

Function Description

The **RM_Track** block includes few useful functions.

If the input **TRACK** is active (not zero), the block tries to track requested position (input **pos** with respect to the limits for velocity, acceleration/deceleration and jerk. The block expects that requested position is changed in each step and therefore recalculates the path in each step. This is difference to **MC_MoveAbsolute** block, which does not allow to change target position while the movement is not finished. This mode is useful if position is generated out of the motion control subsystem, even thought the **MC_PositionProfile** block is better if whole path is known.

If the input **JOG** is active (not zero), the block works like the **MC_MoveVelocity** block (e.g. moves axis with velocity given by parameter **pv** in positive direction with respect to maximum acceleration and jerk). When input **JOG** is released (switched to zero), the block activates stopping sequence and releases the axis when the sequence is finished. This mode is useful for jogging (e.g. setting of position of axis by an operator using up/down buttons).

Input **JOGN** works like **JOG**, but direction is negative.

Note 1: This block hasn’t parameter **BufferMode**. Mode is always aborting.

Note 2: If more functions are selected, only the first one is activated. Order is **TRACK**, **JOGP**, **JOGN**. Simultaneous activation of more than one function is not recommended.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Axis reference (only <strong>RM_Axis.axisRef-uAxis</strong> or <strong>yAxis-uAxis</strong> connections are allowed)</td>
</tr>
<tr>
<td>posvel</td>
<td>Requested target position or velocity [unit]</td>
</tr>
<tr>
<td>TRACKP</td>
<td>Position tracking mode</td>
</tr>
<tr>
<td>TRACKV</td>
<td>Velocity tracking mode</td>
</tr>
<tr>
<td>JOGP</td>
<td>Moving positive direction mode</td>
</tr>
<tr>
<td>JOGN</td>
<td>Moving negative direction mode</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pv</td>
<td>Maximal allowed velocity [unit/s]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>pa</td>
<td>Maximal allowed acceleration [unit/s²]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>pd</td>
<td>Maximal allowed deceleration [unit/s²]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>pj</td>
<td>Maximal allowed jerk [unit/s³]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>iLen</td>
<td>Length of buffer for estimation</td>
<td>⊗10 Long (I32)</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>InTrack</td>
<td>Requested position is reached</td>
<td>Bool</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
<td>Bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>Bool</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
<td>Bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>Bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>Error</td>
</tr>
</tbody>
</table>

REXYGEN general error
Chapter 19

MC_MULTI – Motion control - multi axis blocks

Contents

<table>
<thead>
<tr>
<th>Block Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC_CamIn, MCP_CamIn</td>
<td>Engage the cam</td>
<td>552</td>
</tr>
<tr>
<td>MC_CamOut</td>
<td>Disengage the cam</td>
<td>556</td>
</tr>
<tr>
<td>MCP_CamTableSelect</td>
<td>Cam definition</td>
<td>558</td>
</tr>
<tr>
<td>MC_CombineAxes, MCP_CombineAxes</td>
<td>Combine the motion of 2 axes into a third axis</td>
<td>560</td>
</tr>
<tr>
<td>MC_GearIn, MCP_GearIn</td>
<td>Engage the master/slave velocity ratio</td>
<td>563</td>
</tr>
<tr>
<td>MC_GearInPos, MCP_GearInPos</td>
<td>Engage the master/slave velocity ratio in defined position</td>
<td>566</td>
</tr>
<tr>
<td>MC_PhasingAbsolute, MCP_PhasingAbsolute</td>
<td>Phase shift in synchronized motion (absolute coordinates)</td>
<td>573</td>
</tr>
<tr>
<td>MC_PhasingRelative, MCP_PhasingRelative</td>
<td>Phase shift in synchronized motion (relative coordinates)</td>
<td>576</td>
</tr>
</tbody>
</table>

This block set is the second part of motion control blocks library according to the PLCopen standard for multi axis control. General vendor specific rules are the same as described in chapter 18 (the MC_SINGLE library, blocks for single axis motion control).
MC_CamIn, MCP_CamIn – Engage the cam

Block Symbols

Licence: MOTION CONTROL

Function Description

The MC_CamIn and MCP_CamIn blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The MC_CamIn block switches on a mode in which the slave axis is commanded to position which corresponds to the position of master axis transformed with with a function defined by the MCP_CamTableSelect block (connected to CamTableID input). Denoting the transformation as \( \text{Cam}(x) \), master axis position \( PosM \) and slave axis position \( PosS \), we obtain (for absolute relationship, without phasing): \( PosS = \text{Cam}((PosM - \text{MasterOffset})/\text{MasterScaling}) \times \text{SlaveScaling} + \text{SlaveOffset} \). This form of synchronized motion of the slave axis is called electronic cam.

The cam mode is switched off by executing other motion block on slave axis with mode aborting or by executing a MC_CamOut block. The cam mode is also finished when the master axis leaves a non-periodic cam profile. This situation is indicated by the EndOfProfile output.

In case of a difference between real position and/or velocity of slave axis and cam profile slave axis position and velocity, some transient trajectory must be generated to cancel this offset. This mode is called ramp-in. The ramp-in function is added to the cam profile to eliminate the difference in start position. The RampIn parametr is an average velocity of the ramp-in function. Ramp-in path is not generated for RampIn=0 and error \(-707\) (position or velocity step) is invoked if some difference is detected. Recommended value for the RampIn parametr is 0.1 to 0.5 of maximal slave axis velocity. The parameter has to be lowered if maximal velocity or acceleration error is detected.
## Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uMaster</td>
<td>Master axis reference</td>
<td>Reference</td>
</tr>
<tr>
<td>uSlave</td>
<td>Slave axis reference</td>
<td>Reference</td>
</tr>
<tr>
<td>CamTableID</td>
<td>Cam table reference (connect to MCP_CamTableSelect.CamTableID)</td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>Bool</td>
</tr>
<tr>
<td>MasterOffset</td>
<td>Offset in cam table on master side [unit]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>SlaveOffset</td>
<td>Offset in cam table on slave side [unit]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>MasterScaling</td>
<td>Overall scaling factor in cam table on master side</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>SlaveScaling</td>
<td>Overall scaling factor in cam table on slave side</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>StartMode</td>
<td>Select relative or absolute cam table</td>
<td>Long (I32)</td>
</tr>
<tr>
<td></td>
<td>1 .... Master relative</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 .... Slave relative</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 .... Both relative</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 .... Both absolute</td>
<td></td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
<td>Long (I32)</td>
</tr>
<tr>
<td></td>
<td>1 .... Aborting (start immediately)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 .... Buffered (start after finish of previous motion)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 .... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 .... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 .... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6 .... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)</td>
<td></td>
</tr>
<tr>
<td>RampIn</td>
<td>RampIn factor (0 = RampIn mode not used); average additive velocity (absolute value) during ramp-in process</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yMaster</td>
<td>Master axis reference</td>
<td>Reference</td>
</tr>
<tr>
<td>ySlave</td>
<td>Slave axis reference</td>
<td>Reference</td>
</tr>
<tr>
<td>InSync</td>
<td>Slave axis reached the cam profile</td>
<td>Bool</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
<td>Bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>Bool</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
<td>Bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>Bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td>i .... REXYGEN general error</td>
<td></td>
</tr>
<tr>
<td>EndOfProfile</td>
<td>Indicate end of cam profile (not periodic cam only)</td>
<td>Bool</td>
</tr>
</tbody>
</table>
SyncDistance  Position deviation of the slave axis from synchronized position  Double (F64)
Example
MC_CamOut — Disengage the cam

Block Symbol Licence: MOTION CONTROL

Function Description

The MC_CamOut block switches off the cam mode on slave axis. If cam mode is not active, the block does nothing (no error is activated).

Inputs

<table>
<thead>
<tr>
<th>uSlave</th>
<th>Slave axis reference</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>ySlave</th>
<th>Slave axis reference</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>Bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>Bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>Bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td>i .... REXYGEN general error</td>
<td></td>
</tr>
</tbody>
</table>
Example

Diagram showing execution of different blocks in a control system.

- **Execute**: Blocks 0, 1, 2, and 3 are executed.
- **Active**: Block 1 and Block 2 are active.
- **Acceleration**: Acceleration chart for blocks 1 and 2.
- **Velocity**: Velocity chart for blocks 1 and 2.
- **Position**: Position chart for blocks 1 and 2.

Graphs illustrating the execution status, active blocks, and performance metrics for each block.
MCP_CamTableSelect – Cam definition

Block Symbol Licence: MOTION CONTROL

Function Description

The MCP_CamTableSelect block defines a cam profile. The definition is similar to MC_PositionProfile block, but the time axis is replaced by master position axis. There are also two possible ways for cam profile definition:

1. sequence of values: given sequence of master-slave position pairs. In each master position interval, value of slave position is interpolated by 3rd-order polynomial (simple linear interpolation would lead to steps in velocity at interval border). Master position sequence is in array/parameter mvalues, slave position sequence is in array/parameter svalues. Master position sequence must be increasing.

2. spline: master position sequence is the same as in previous case. Each interval is interpolated by 5th-order polynomial \( p(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \) where beginning of time-interval is defined for \( x = 0 \), end of time-interval holds for \( x = 1 \) and factors \( a_i \) are put in array/parameter svalues in ascending order (e.g. array/parameter svalues contain 6 values for each interval). This method allows to reduce the number of intervals and there is special graphical editor available for interpolating curve synthesis.

For both cases the master position sequence can be equidistantly spaced in time and then the time array includes only first and last point.

Note 1: input CamTable which is defined in PLCOpen specification is missing, because all path data are set in the parameters of the block.

Note 2: parameter svalues must be set as a vector in all cases, e.g. text string must not include a semicolon.

Note 3: incorrect parameter value cSeg (higher then real size of arrays times and/or values) can lead to unpredictable results and in some cases to crash of the whole runtime execution (The problem is platform dependent and currently it is observed only for SIMULINK version).

Inputs

| uMaster | Master axis reference | Reference |
| uSlave  | Slave axis reference  | Reference |
| Execute | The block is activated on rising edge | Bool |
Outputs

- **yMaster**: Master axis reference
- **ySlave**: Slave axis reference
- **Done**: Algorithm finished
- **Busy**: Algorithm not finished yet
- **Error**: Error occurred
- **ErrorID**: Result of the last operation
  - i . . . . REXYGEN general error
- **CamTableID**: Cam table reference (connect to MC_CamIn.CamTableID)

Parameters

- **alg**: Algorithm for interpolation
  - 1 . . . . Sequence of time/value pairs
  - 2 . . . . Sequence of equidistant values
  - 3 . . . . Spline
  - 4 . . . . Equidistant spline

- **cSeg**: Number of profile segments

- **Periodic**: Indicate periodic cam profile

- **camname**: Filename of special editor data file (filename is generated by system if parameter is empty)

- **mvalues**: Master positions where segments are switched

- **svalues**: Slave positions or interpolating polynomial coefficients (a0, a1, a2, ...)

- **Reference**
- **Bool**
- **Error**
- **Long (I32)**
- **String**
- **Double (F64)**

- **[0 30]**
- **[0 100 100 0]**
MC_CombineAxes, MCP_CombineAxes – Combine the motion of 2 axes into a third axis

Block Symbols

**Function Description**

The **MC_CombineAxes** block combines a motion of two master axes into a slave axis command. The slave axis indicates synchronized motion state. Following relationship holds:

\[
\text{SlavePosition} = \text{Master1Position} \cdot \frac{\text{GearRatioNumeratorM1}}{\text{GearRatioDenominatorM1}} + \frac{\text{Master2Position}}{\text{GearRatioDenominatorM2}}
\]

Negative number can be set in **GearRatio**... parameter to obtain the resulting slave movement in form of difference of master axes positions.

**Inputs**

- **uMaster1**: First master axis reference (Reference)
- **uMaster2**: Second master axis reference (Reference)
- **uSlave**: Slave axis reference (Reference)
- **Execute**: The block is activated on rising edge (Bool)
- **GearRatioNumeratorM1**: Numerator for the gear factor for master axis 1 (Long (I32))
- **GearRatioDenominatorM1**: Denominator for the gear factor for master axis 1 (Long (I32))
- **GearRatioNumeratorM2**: Numerator for the gear factor for master axis 2 (Long (I32))
- **GearRatioDenominatorM2**: Denominator for the gear factor for master axis 2 (Long (I32))
**BufferMode** Buffering mode

1 . . . . Aborting (start immediately)
2 . . . . Buffered (start after finish of previous motion)
3 . . . . Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
4 . . . . Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
5 . . . . Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)
6 . . . . Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)

**RampIn** RampIn factor (0 = RampIn mode not used)

**Outputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yMaster1</td>
<td>First master axis reference</td>
<td>Reference</td>
</tr>
<tr>
<td>yMaster2</td>
<td>Second master axis reference</td>
<td>Reference</td>
</tr>
<tr>
<td>ySlave</td>
<td>Slave axis reference</td>
<td>Reference</td>
</tr>
<tr>
<td>InSync</td>
<td>Slave axis reached the cam profile</td>
<td>Bool</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
<td>Bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>Bool</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
<td>Bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>Bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>Error</td>
</tr>
<tr>
<td>SyncDistance</td>
<td>Position deviation of the slave axis from synchronized position</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
Example
MC_GearIn, MCP_GearIn – Engange the master/slave velocity ratio

Block Symbols

Function Description

The MC_GearIn and MCP_GearIn blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_version of the block.

The MC_GearIn block commands the slave axis motion in such a way that a preset ratio between master and slave velocities is maintained. Considering the velocity of master axis \(V_{elM}\) and velocity of slave axis \(V_{elS}\), following relation holds (without phasing): \(V_{elS} = V_{elM} \times \frac{\text{RatioNumerator}}{\text{RatioDenominator}}\). Position and acceleration is commanded to be consistent with velocity; position/distance ratio is also locked. This mode of synchronized motion is called electronic gear.

The gear mode is switched off by executing other motion block on slave axis with mode aborting or by executing a MC_GearIn block.

Similarly to the MC_CamIn block, ramp-in mode is activated if initial velocity of slave axis is different from master axis and gearing ratio. Parameters Acceleration, Deceleration, Jerk are used during ramp-in mode.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{uMaster}</td>
<td>Master axis reference</td>
<td>Reference</td>
</tr>
<tr>
<td>\textit{uSlave}</td>
<td>Slave axis reference</td>
<td>Reference</td>
</tr>
<tr>
<td>\textit{Execute}</td>
<td>The block is activated on rising edge</td>
<td>Bool</td>
</tr>
<tr>
<td>\textit{RatioNumerator}</td>
<td>Gear ratio Numerator</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>\textit{RatioDenominator}</td>
<td>Gear ratio Denominator</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>\textit{Acceleration}</td>
<td>Maximal allowed acceleration ([\text{unit/s}^2])</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>\textit{Deceleration}</td>
<td>Maximal allowed deceleration ([\text{unit/s}^2])</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
CHAPTER 19. MC MULTI - MOTION CONTROL - MULTI AXIS BLOCKS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s^3]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>1 . . .</td>
<td>Aborting (start immediately)</td>
<td></td>
</tr>
<tr>
<td>2 . . .</td>
<td>Buffered (start after finish of previous motion)</td>
<td></td>
</tr>
<tr>
<td>3 . . .</td>
<td>Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
<td></td>
</tr>
<tr>
<td>4 . . .</td>
<td>Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
<td></td>
</tr>
<tr>
<td>5 . . .</td>
<td>Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)</td>
<td></td>
</tr>
<tr>
<td>6 . . .</td>
<td>Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)</td>
<td></td>
</tr>
</tbody>
</table>

**Outputs**

- **yMaster**: Master axis reference
- **ySlave**: Slave axis reference
- **InGear**: Slave axis reached gearing ratio
- **CommandAborted**: Algorithm was aborted
- **Busy**: Algorithm not finished yet
- **Active**: The block is controlling the axis
- **Error**: Error occurred
- **ErrorID**: Result of the last operation
  - i . . . REXYGEN general error

Reference

Reference

Bool

Bool

Bool

Bool

Error
Example
**MC_GearInPos, MCP_GearInPos – Engage the master/slave velocity ratio in defined position**

**Function Description**

*The MC_GearInPos and MCP_GearInPos blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.*

The functional block **MC_GearInPos** engages a synchronized motion of master and slave axes in such a way that the ratio of velocities of both axes is maintained at a constant value. Compared to **MC_GearIn**, also the master to slave position ratio is determined in a given reference point, i.e. following relation holds:

\[
\frac{\text{SlavePosition} - \text{SlaveSyncPosition}}{\text{MasterPosition} - \text{MasterSyncPosition}} = \frac{\text{RatioNumerator}}{\text{RatioDenominator}}
\]

In case that the slave position does not fulfill this condition of synchronicity at the moment of block activation (i.e. in an instant of positive edge of Execute input and after execution of previous commands in buffered mode), synchronization procedure is started and indicated by output **StartSync**. During this procedure, proper slave trajectory which results in smooth synchronization of both axes is generated with respect to actual master motion and slave limits for Velocity, Acceleration, Deceleration and Jerk (these limits are not applied from the moment of successful synchronization). Parameter setting **MasterStartDate**=0 leads to immediate start of synchronization procedure...
at the moment of block activation (by the Execute input). Otherwise, the synchronization starts as soon as the master position enters the interval \( \text{MasterSyncPosition} \pm \text{MasterStartDistance} \).

Notes:
1. The synchronization procedure uses two algorithms: I. The algorithm implemented in \text{MC_MoveAbsolute} is recomputed in every time instant in such a way, that the end velocity is set to actual velocity of master axis. II. The position, velocity and acceleration is generated in the same manner as in the synchronized motion and a proper 5th order interpolation polynomial is added to achieve smooth transition to the synchronized state. The length of interpolation trajectory is computed in such a way that maximum velocity, acceleration and jerk do not violate the specified limits (for the interpolation polynomial). The first algorithm cannot be used for nonzero acceleration of the master axis whereas the second does not guarantee the compliance of maximum limits for the overall slave trajectory. Both algorithms are combined in a proper way to achieve the synchronized motion of both axes.

2. The block parameters (execution of synchronization and velocity/acceleration limits) have to be chosen so that the slave position is close to \text{SlaveSyncPosition} approximately at the moment when the master position enters the range for synchronization given by \text{MasterSyncPosition} and \text{MasterStartDistance}. Violation of this rule can lead to unpredictable behaviour of the slave axis during the synchronization or to an overrun of the specified limits for slave axis. However, the motion of both axes is usually well defined and predictable in standard applications and correct synchronization can be performed easily by proper configuration of motion commands and functional block parameters.

\subsection*{Inputs}
\begin{itemize}
    \item \text{uMaster} \hspace{1cm} \text{Master axis reference} \hspace{1cm} \text{Reference}
    \item \text{uSlave} \hspace{1cm} \text{Slave axis reference} \hspace{1cm} \text{Reference}
    \item \text{Execute} \hspace{1cm} \text{The block is activated on rising edge} \hspace{1cm} \text{Bool}
    \item \text{RatioNumerator} \hspace{1cm} \text{Gear ratio Numerator} \hspace{1cm} \text{Long (I32)}
    \item \text{RatioDenominator} \hspace{1cm} \text{Gear ratio Denominator} \hspace{1cm} \text{Long (I32)}
    \item \text{MasterSyncPosition} \hspace{1cm} \text{Master position for synchronization} \hspace{1cm} \text{Double (F64)}
    \item \text{SlaveSyncPosition} \hspace{1cm} \text{Slave position for synchronization} \hspace{1cm} \text{Double (F64)}
    \item \text{MasterStartDistance} \hspace{1cm} \text{Master distance for starting gear in procedure} \hspace{1cm} \text{Double (F64)}
    \item \text{Velocity} \hspace{1cm} \text{Maximal allowed velocity [unit/s]} \hspace{1cm} \text{Double (F64)}
    \item \text{Acceleration} \hspace{1cm} \text{Maximal allowed acceleration [unit/s²]} \hspace{1cm} \text{Double (F64)}
    \item \text{Deceleration} \hspace{1cm} \text{Maximal allowed deceleration [unit/s²]} \hspace{1cm} \text{Double (F64)}
    \item \text{Jerk} \hspace{1cm} \text{Maximal allowed jerk [unit/s³]} \hspace{1cm} \text{Double (F64)}
\end{itemize}
BufferMode Buffering mode
   1 ...... Aborting (start immediately)
   2 ...... Buffered (start after finish of previous motion)
   3 ...... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
   4 ...... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
   5 ...... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)
   6 ...... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)

SyncMode Synchronization mode (cyclic axes only)
   1 ...... CatchUp
   2 ...... Shortest
   3 ...... SlowDown

Outputs
   yMaster  Master axis reference Reference
   ySlave   Slave axis reference Reference
   StartSync Commanded gearing starts Bool
   InSync   Slave axis reached the cam profile Bool
   CommandAborted Algorithm was aborted Bool
   Busy     Algorithm not finished yet Bool
   Active   The block is controlling the axis Bool
   Error    Error occurred Bool
   ErrorID  Result of the last operation Error
      i ...... REXYGEN general error
   SyncDistance Position deviation of the slave axis from synchronized position Double (F64)
<table>
<thead>
<tr>
<th>Velocity</th>
<th>Sync Mode</th>
<th>Ratio Numerator</th>
<th>Ratio Denominator</th>
<th>Position</th>
<th>Mode</th>
<th>Execute</th>
<th>Deceleration</th>
<th>Acceleration</th>
<th>Sync Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>velocity1</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>30</td>
<td>1</td>
<td>execute1</td>
<td>20</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>velocity2</td>
<td>60</td>
<td>2</td>
<td>1</td>
<td>40</td>
<td>1</td>
<td>execute2</td>
<td>50</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>velocity3</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td></td>
<td>1</td>
<td>execute3</td>
<td>20</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

**Diagram:**
- **uAxis**:
  - Execute
  - Position
  - Velocity
  - Acceleration
  - Deceleration
  - Jerk
  - Buffer Mode
  - Direction
- **yAxis**:
  - Done
  - Command Aborted
  - Busy
  - Active
  - Error
  - Error ID

- **MC_GearInPos**:
  - Block 2
  - uMaster
  - uSlave
  - Execute
  - Ratio Numerator
  - Ratio Denominator
  - Master Sync Position
  - Slave Sync Position
  - Master Start Distance
  - Velocity
  - Acceleration
  - Deceleration
  - Jerk
  - Buffer Mode
  - Sync Mode
  - yMaster
  - ySlave
  - Start Sync
  - In Sync
  - Command Aborted
  - Busy
  - Active
  - Error
  - Error ID
  - Sync Distance
Execute

Active

InSync

Acceleration

Velocity

Position

bool

Block 1 execute (MoveAbsolute1)
Block 2 execute (GearIn)
Block 3 execute (MoveAbsolute2)

Active

Block 1 active (MoveAbsolute1)
Block 2 active (GearIn)
Block 3 active (MoveAbsolute2)

InSync

Block 2 InSync (GearIn)

Acceleration

Acceleration axis 1
Acceleration axis 2

Velocity

Velocity axis 1
Velocity axis 2

Position

Position axis 1
Position axis 2
MC_GearOut – Disengage the master/slave velocity ratio

Block Symbol

Licence: MOTION CONTROL

Function Description

The MC_GearOut block switches off the gearing mode on the slave axis. If gearing mode is not active (no MC_GearIn block commands slave axis at this moment), block does nothing (no error is activated).

Inputs

<table>
<thead>
<tr>
<th>uSlave</th>
<th>Slave axis reference</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>ySlave</th>
<th>Slave axis reference</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>Bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>Bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>Bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>Error</td>
</tr>
<tr>
<td>i . . . .</td>
<td>REXYGEN general error</td>
<td></td>
</tr>
</tbody>
</table>
Example

```
velocity2 30
ratio_numerator1 2
ratio_denominator1 1
position2 100
mode2 2
mode1 2
jerk3 0
jerk1 0
execute ...
nation
```

```
development
```

```
Execute
```

```
Active
```

```
Acceleration
```

```
Velocity
```

```
Position
```

```
0 1 2 3 4 5 6
0 0.5 1
```

```
oblock 1 execute (GearIn)
oblock 2 execute (MoveAbsolute)
oblock 3 execute (GearOut)
```

```
oblock 1 active (GearIn)
oblock 2 active (MoveAbsolute)
```

```
Acceleration axis 1 = master
Acceleration axis 2 = slave
```

```
Velocity axis 1 = master
Velocity axis 2 = slave
```

```
Position axis 1 = master
Position axis 2 = slave
```

```
0 1 2 3 4 5 6
0 50 100 150 200
```

```
```
MC_PhasingAbsolute, MCP_PhasingAbsolute – Phase shift in synchronized motion (absolute coordinates)

Block Symbols

Licence: MOTION CONTROL

Function Description

The MC_PhasingAbsolute and MCP_PhasingAbsolute blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_version of the block.

The MC_PhasingAbsolute block introduces an additional phase shift in master-slave relation defined by an electronic cam (MC_CamIn) or electronic gear (MC_GearIn). The functionality of this command is very similar to MC_MoveSuperimposed (additive motion from 0 to PhaseShift position with respect to maximum velocity acceleration and jerk). The only difference is that the additive position/velocity/acceleration is added to master axis reference position in the functional dependence defined by a cam or gear ratio for the computation of slave motion instead of its direct summation with master axis movement. The absolute value of final phase shift is specified by PhaseShift parameter.

Note: The motion command is analogous to rotation of a mechanical cam by angle

PhaseShift

Inputs

<table>
<thead>
<tr>
<th>uMaster</th>
<th>Master axis reference</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uSlave</td>
<td>Slave axis reference</td>
<td>Reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>Bool</td>
</tr>
<tr>
<td>PhaseShift</td>
<td>Requested phase shift (distance on master axis) for cam</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Velocity</td>
<td>Maximal allowed velocity [unit/s]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s²]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
CHAPTER 19. MC_MULTI – MOTION CONTROL - MULTI AXIS BLOCKS

Jerk Maximal allowed jerk [unit/s³] Double (F64)
BufferMode Buffering mode Long (I32)
  1 . . . Aborting
  2 . . . Buffered

Outputs

yMaster Master axis reference Reference
ySlave Slave axis reference Reference
Done Algorithm finished Bool
CommandAborted Algorithm was aborted Bool
Busy Algorithm not finished yet Bool
Active The block is controlling the axis Bool
Error Error occurred Bool
ErrorID Result of the last operation Error
  i . . . REXYGEN general error
Example
MC_PhasingRelative, MCP_PhasingRelative – Phase shift in synchronized motion (relative coordinates)

Block Symbols

Function Description

The MC_PhasingRelative and MCP_PhasingRelative blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_version of the block.

The MC_PhasingRelative introduces an additional phase shift in master-slave relation defined by an electronic cam (MC_CamIn) or electronic gear (MC_GearIn). The functionality of this command is very similar to MC_MoveSuperimposed (additive motion from 0 to PhaseShift position with respect to maximum velocity acceleration and jerk). The only difference is that the additive position/velocity/acceleration is added to master axis reference position in the functional dependence defined by a cam or gear ratio for the computation of slave motion instead of its direct summation with master axis movement. The relative value of final phase shift with respect to previous value is specified by PhaseShift parameter. Note: The motion command is analogous to rotation of a mechanical cam by angle PhaseShift.

Inputs

<table>
<thead>
<tr>
<th>uMaster</th>
<th>Master axis reference</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uSlave</td>
<td>Slave axis reference</td>
<td>Reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>Bool</td>
</tr>
<tr>
<td>PhaseShift</td>
<td>Requested phase shift (distance on master axis) for cam</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Velocity</td>
<td>Maximal allowed velocity [unit/s]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s²]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
Jerk  
Maximal allowed jerk [unit/s³]  

BufferMode Buffering mode  
  1 . . . . Aborting  
  2 . . . . Buffered

Outputs

yMaster  
Master axis reference  

ySlave  
Slave axis reference  

Done  
Algorithm finished  

CommandAborted  
Algorithm was aborted  

Busy  
Algorithm not finished yet  

Active  
The block is controlling the axis  

Error  
Error occurred  

ErrorID  
Result of the last operation  
  i . . . . REXYGEN general error
Chapter 20

MC_COORD – Motion control - coordinated movement blocks

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM_AxesGroup</td>
<td>Axes group for coordinated motion control</td>
<td>582</td>
</tr>
<tr>
<td>RM_Feed</td>
<td>* MC Feeder</td>
<td>585</td>
</tr>
<tr>
<td>RM_Gcode</td>
<td>* CNC motion control</td>
<td>586</td>
</tr>
<tr>
<td>MC_AddAxisToGroup</td>
<td>Adds one axis to a group</td>
<td>588</td>
</tr>
<tr>
<td>MC_UngroupAllAxes</td>
<td>Removes all axes from the group</td>
<td>589</td>
</tr>
<tr>
<td>MC_GroupEnable</td>
<td>Changes the state of a group to GroupEnable</td>
<td>590</td>
</tr>
<tr>
<td>MC_GroupDisable</td>
<td>Changes the state of a group to GroupDisabled</td>
<td>591</td>
</tr>
<tr>
<td>MC_SetCartesianTransform</td>
<td>Sets Cartesian transformation</td>
<td>592</td>
</tr>
<tr>
<td>MC_ReadCartesianTransform</td>
<td>Reads the parameter of the cartesian transformation</td>
<td>594</td>
</tr>
<tr>
<td>MC_GroupSetPosition, MCP_GroupSetPosition</td>
<td>Sets the position of all axes in a group</td>
<td>595</td>
</tr>
<tr>
<td>MC_GroupReadActualPosition</td>
<td>Read actual position in the selected coordinate system</td>
<td>597</td>
</tr>
<tr>
<td>MC_GroupReadActualVelocity</td>
<td>Read actual velocity in the selected coordinate system</td>
<td>598</td>
</tr>
<tr>
<td>MC_GroupReadActualAcceleration</td>
<td>Read actual acceleration in the selected coordinate system</td>
<td>599</td>
</tr>
<tr>
<td>MC_GroupStop</td>
<td>Stopping a group movement</td>
<td>600</td>
</tr>
<tr>
<td>MC_GroupHalt</td>
<td>Stopping a group movement (interruptible)</td>
<td>603</td>
</tr>
<tr>
<td>MC_GroupInterrupt, MCP_GroupInterrupt</td>
<td>Read a group interrupt</td>
<td>608</td>
</tr>
<tr>
<td>MC_GroupContinue</td>
<td>Continuation of interrupted movement</td>
<td>609</td>
</tr>
<tr>
<td>MC_GroupReadStatus</td>
<td>Read a group status</td>
<td>610</td>
</tr>
<tr>
<td>MC_GroupReadError</td>
<td>Read a group error</td>
<td>611</td>
</tr>
<tr>
<td>MC_GroupReset</td>
<td>Reset axes errors</td>
<td>612</td>
</tr>
</tbody>
</table>
MC_MoveLinearAbsolute — Linear move to position (absolute coordinates) ........................................ 613
MC_MoveLinearRelative — Linear move to position (relative to execution point) ........................................ 617
MC_MoveCircularAbsolute — Circular move to position (absolute coordinates) ........................................ 621
MC_MoveCircularRelative — Circular move to position (relative to execution point) ........................................ 625
MC_MoveDirectAbsolute — Direct move to position (absolute coordinates) ........................................ 629
MC_MoveDirectRelative — Direct move to position (relative to execution point) ........................................ 632
MC_MovePath — General spatial trajectory generation ........................................ 635
MC_GroupSetOverride — Set group override factors ........................................ 637
<table>
<thead>
<tr>
<th></th>
<th>Aborting</th>
<th>Buffered without Blending</th>
<th>Blending</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trajectory of TCP</td>
<td><img src="image1" alt="Aborting Trajectory" /></td>
<td><img src="image2" alt="Buffered Trajectory" /></td>
<td><img src="image3" alt="Blending Trajectory" /></td>
</tr>
<tr>
<td>Speed of TCP</td>
<td><img src="image4" alt="Aborting Speed" /></td>
<td><img src="image5" alt="Buffered Speed" /></td>
<td><img src="image6" alt="Blending Speed" /></td>
</tr>
</tbody>
</table>
**RM_AxesGroup** – Axes group for coordinated motion control

**Block Symbol**

```
RM_AxesGroup
```

**Function Description**

- Note 1: Applicable for all non-administrative (moving) function blocks.
- Note 2: In the states GroupErrorStop or GroupStopping, all Function Blocks can be called, although they will not be executed, except MC_GroupReset for GroupErrorStop and any occurring Error. They will generate the transition to GroupStandby or GroupErrorStop respectively.
- Note 3: MC_GroupStop.DONE AND NOT MC_GroupStop.EXECUTE
- Note 4: Transition is applicable if last axis is removed from the group
- Note 5: Transition is applicable while group is not empty.
- Note 6: MC_GroupDisable and MC_UngroupAllAxes can be issued in all states and will change the state to GroupDisabled.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NcsCount</td>
<td>Number of axis in MCS</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>NcsCount</td>
<td>Number of axis in ACS</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>PosCount</td>
<td>Number of position axis</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>Velocity</td>
<td>Maximal allowed velocity [unit/s]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s²]</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>refGroup</td>
<td>Axes group reference</td>
<td>Reference</td>
</tr>
<tr>
<td>defPos</td>
<td>Position, velocity and acceleration vector</td>
<td>Reference</td>
</tr>
<tr>
<td>iState</td>
<td>Group status</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>0 ........</td>
<td>Disabled</td>
<td></td>
</tr>
<tr>
<td>1 ........</td>
<td>Standby</td>
<td></td>
</tr>
<tr>
<td>2 ........</td>
<td>Homing</td>
<td></td>
</tr>
<tr>
<td>6 ........</td>
<td>Moving</td>
<td></td>
</tr>
<tr>
<td>7 ........</td>
<td>Stopping</td>
<td></td>
</tr>
<tr>
<td>8 ........</td>
<td>Error stop</td>
<td></td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>Error</td>
</tr>
<tr>
<td>i ........</td>
<td>REXYGEN general error</td>
<td></td>
</tr>
</tbody>
</table>
The State Diagram of AxesGroup
RM_Feed — * MC Feeder

Block Symbol Licence: COORDINATED MOTION

Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

- uAxesGroup Axes group reference Reference
- Execute The block is activated on rising edge Bool

Parameters

- Filename 0 String
- VelFactor 0 $\downarrow 0.01 \uparrow 100.0 \odot 1.0$ Double (F64)
- Relative 0 Bool
- CoordSystem 0 $\downarrow 1 \uparrow 3 \odot 2$ Long (I32)
- BufferMode 0 $\downarrow 1 \uparrow 6 \odot 1$ Long (I32)
- TransitionMode 0 $\downarrow 0 \uparrow 15 \odot 1$ Long (I32)
- TransitionParameter 0 Double (F64)

Outputs

- yAxesGroup Axes group reference Reference
- Done Algorithm finished Bool
- CommandAborted Algorithm was aborted Bool
- Busy Algorithm not finished yet Bool
- Active The block is controlling the axis Bool
- Error Error occurred Bool
- ErrorID Result of the last operation Error
- Aux 0 Double (F64)
**RM_Gcode – * CNC motion control**

**Block Symbol**

![Block Symbol](image)

**Function Description**

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

**Inputs**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Axes group reference</td>
<td>Reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>Bool</td>
</tr>
<tr>
<td>BlockSkip</td>
<td>MILAN</td>
<td>Bool</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseDir</td>
<td>Directory of the G-code files</td>
<td>String</td>
</tr>
<tr>
<td>MainFile</td>
<td>Source file number</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>CoordSystem</td>
<td></td>
<td>Long (I32)</td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>TransitionMode</td>
<td>Transition mode in blending mode</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>TransitionParameter</td>
<td>Parametr for transition (depends on transition mode)</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
workOffsets  Sets with initial coordinate
           ◆[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0] Double (F64)
toolOffsets  Sets of tool offset ◆[0 0 0] Double (F64)
cutterOffsets Tool radii ◆[0 0 0] Double (F64)

Outputs

yAxesGroup Axes group reference  Reference
Done       Algorithm finished  Bool
CommandAborted Algorithm was aborted  Bool
Busy       Algorithm not finished yet  Bool
Active     The block is controlling the axis  Bool
Error      Error occurred  Bool
ErrorID    Result of the last operation  Error
            i ...... REXYGEN general error
Cooling    Cooling  Bool
MoveType   Command execution  Long (I32)
ExecutingLine Current line of G-code  Long (I32)
SpindleSpeed Spindle speed  Long (I32)
**MC_AddAxisToGroup** – Adds one axis to a group

**Block Symbol**

![Diagram](image)

**Function Description**

The function block **MC_AddAxisToGroup** adds one **uAxis** to the group in a structure **uAxesGroup**. Axes Group is implemented by the function block **RM_AxesGroup**. The input **uAxis** must be defined by the function block **RM_Axis** from the MC_SINGLE library.

Note 1: Every **IdentInGroup** is unique and can be used only for one time otherwise the error is set.

**Inputs**

- **uAxesGroup** Axes group reference
- **uAxis** Axis reference (only **RM_Axis.axisRef-uAxis** or **yAxis-uAxis** connections are allowed)
- **Execute** The block is activated on rising edge
- **IdentInGroup** The order of axes in the group (0 = first unassigned)

**Outputs**

- **yAxesGroup** Axes group reference
- **yAxis** Axis reference (only **RM_Axis.axisRef-yAxis** or **yAxis-yAxis** connections are allowed)
- **Done** Algorithm finished
- **Busy** Algorithm not finished yet
- **Error** Error occurred
- **ErrorID** Result of the last operation
  - i ..... REXYGEN general error
- **ErrorID** Error
MC_UngroupAllAxes – Removes all axes from the group

Block Symbol

Function Description

The function block MC_UngroupAllAxes removes all axes from the group uAxesGroup. After finalization the state is changed to "GroupDisabled".

Note 1: If the function block is execute in the group state "GroupDisabled", "GroupStandBy" or "GroupErrorStop" the error is set and the block is not execute.

Inputs

<table>
<thead>
<tr>
<th>uAxesGroup</th>
<th>Axes group reference</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>yAxesGroup</th>
<th>Axes group reference</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>Bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>Bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>Bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>Error</td>
</tr>
<tr>
<td>i ... REXYGEN general error</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MC_GroupEnable – Changes the state of a group to GroupEnable

Block Symbol

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Axes group reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxesGroup</td>
<td>Axes group reference</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td></td>
<td>i . . . . REXYGEN general error</td>
</tr>
</tbody>
</table>

Licence: COORDINATED MOTION
**MC_GroupDisable** – Changes the state of a group to GroupDisabled

Block Symbol

Licence: **COORDINATED MOTION**

**Function Description**

The function block **MC_GroupDisable** changes the state for the group **uAxesGroup** to "GroupDisabled". If the axes are not standing still while issuing this command the state of the group is changed to "Stopping". It is mean stopping with the maximal allowed deceleration. When stopping is done the state of the group is changed to "GroupDisabled".

**Inputs**

- **uAxesGroup**: Axes group reference
- **Execute**: The block is activated on rising edge

**Outputs**

- **yAxesGroup**: Axes group reference
- **Done**: Algorithm finished
- **Busy**: Algorithm not finished yet
- **Error**: Error occurred
- **ErrorID**: Result of the last operation
  - **i**: REXYGEN general error
MC_SetCartesianTransform – Sets Cartesian transformation

Block Symbol

Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Axes group reference</td>
<td>Reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>Bool</td>
</tr>
<tr>
<td>TransX</td>
<td>X-component of translation vector</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>TransY</td>
<td>Y-component of translation vector</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>TransZ</td>
<td>Z-component of translation vector</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>RotAngle1</td>
<td>Rotation angle component</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>RotAngle2</td>
<td>Rotation angle component</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>RotAngle3</td>
<td>Rotation angle component</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>Relative</td>
<td>Mode of position inputs</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxesGroup</td>
<td>Axes group reference</td>
<td>Reference</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>Bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>Bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>Bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>Error</td>
</tr>
</tbody>
</table>

Note: REXYGEN general error
Cartesian Machine Coordinate System

\[ z' = z \]
\[ x' \]
\[ y' \]
\[ y'' = y' \]
\[ x'' \]
\[ z'' \]
\[ x''' = x'' \]
\[ y''' \]
\[ z''' \]

Trans

RotZ

RotY

RotX

TransX

TransY

RotZ
MC_ReadCartesianTransform – Reads the parameter of the cartesian transformation

Block Symbol

Function Description

The function block MC_ReadCartesianTransform reads the parameter of the cartesian transformation that is active between the MCS and PCS. The parameters are valid only if the output Valid is true which is achieved by setting the input Enable on true. If more than one transformation is active, the resulting cartesian transformation is given.

Inputs

| AxesGroup | Axes group reference | Reference |
| Enable | Block function is enabled | Bool |

Outputs

| AxesGroup | Axes group reference | Reference |
| Valid | Output value is valid | Bool |
| Busy | Algorithm not finished yet | Bool |
| TransX | X-component of translation vector | Double (F64) |
| TransY | Y-component of translation vector | Double (F64) |
| TransZ | Z-component of translation vector | Double (F64) |
| RotAngle1 | Rotation angle component | Double (F64) |
| RotAngle2 | Rotation angle component | Double (F64) |
| RotAngle3 | Rotation angle component | Double (F64) |
| Error | Error occurred | Bool |
| ErrorID | Result of the last operation | i ... REXYGEN general error |
**MC_GroupSetPosition, MCP_GroupSetPosition** — Sets the position of all axes in a group

**Block Symbols**

**Licence:** COORDINATED MOTION

**Function Description**

The **MC_GroupSetPosition** and **MCP_GroupSetPosition** blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the **MCP** version of the block.

The function block **MC_GroupSetPosition** sets the position of all axes in the group **uAxesGroup** without moving the axes. The new coordinates are described by the input **Position**. With the coordinate system input **CoordSystem** the according coordinate system is selected. The function block **MC_GroupSetPosition** shifts position of the addressed coordinate system and affect the higher level coordinate systems (so if ACS selected, MCS and PCS are affected).

**Inputs**

- **uAxesGroup**: Axes group reference, Reference
- **Program**: Execute the block is activated on rising edge, Bool
- **Position**: Array of coordinates (positions and orientations), Reference
- **Relative**: Mode of position inputs
  - **off** ... absolute
  - **on** ... relative
- **CoordSystem**: Reference to the coordinate system used
  - 1 .... ACS
  - 2 .... MCS
  - 3 .... PCS

**Outputs**

- **yAxesGroup**: Axes group reference, Reference
- **Done**: Algorithm finished, Bool
- **Busy**: Algorithm not finished yet, Bool
- **CommandAborted**: Algorithm was aborted, Bool
- **Error**: Error occurred, Bool
<table>
<thead>
<tr>
<th>Error ID</th>
<th>Result of the last operation</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>i ........</td>
<td>REXYGEN general error</td>
<td></td>
</tr>
</tbody>
</table>
MC_GroupReadActualPosition – Read actual position in the selected coordinate system

**Function Description**

The function block `MC_GroupReadActualPosition` returns the actual position in the selected coordinate system of an axes group. The position is valid only if the output `Valid` is true which is achieved by setting the input `Enable` on true.

**Inputs**

- **uAxesGroup**: Axes group reference
- **Enable**: Block function is enabled
- **CoordSystem**: Reference to the coordinate system used
  - 1: ACS
  - 2: MCS
  - 3: PCS

**Outputs**

- **yAxesGroup**: Axes group reference
- **Valid**: Output value is valid
- **Busy**: Algorithm not finished yet
- **Error**: Error occurred
  - i: REXYGEN general error
- **ErrorID**: Result of the last operation
- **Position**: Reference
MC_GroupReadActualVelocity - Read actual velocity in the selected coordinate system

Block Symbol

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Axes group reference</td>
</tr>
<tr>
<td>Enable</td>
<td>Block function is enabled</td>
</tr>
<tr>
<td>CoordSystem</td>
<td>Reference to the coordinate system used</td>
</tr>
<tr>
<td></td>
<td>1       ACS</td>
</tr>
<tr>
<td></td>
<td>2       MCS</td>
</tr>
<tr>
<td></td>
<td>3       PCS</td>
</tr>
<tr>
<td>yAxesGroup</td>
<td>Axes group reference</td>
</tr>
<tr>
<td>Valid</td>
<td>Output value is valid</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td>Velocity</td>
<td>xxx      Reference</td>
</tr>
</tbody>
</table>

Inputs

- uAxesGroup: Axes group reference
- Enable: Block function is enabled
- CoordSystem: Reference to the coordinate system used
  - 1: ACS
  - 2: MCS
  - 3: PCS

License: COORDINATED MOTION

Function Description

The function block MC_GroupReadActualVelocity returns the actual velocity in the selected coordinate system of an axes group. The position is valid only if the output Valid is true which is achieved by setting the input Enable on true.

Inputs

- uAxesGroup: Axes group reference
- Enable: Block function is enabled
- CoordSystem: Reference to the coordinate system used
  - 1: ACS
  - 2: MCS
  - 3: PCS

Outputs

- yAxesGroup: Axes group reference
- Valid: Output value is valid
- Busy: Algorithm not finished yet
- Error: Error occurred
- ErrorID: Result of the last operation
- Velocity: xxx Reference
MC_GroupReadActualAcceleration – Read actual acceleration in the selected coordinate system

Block Symbol

Licence: COORDINATED MOTION

Function Description

The function block `MC_GroupReadActualAcceleration` returns the actual velocity in the selected coordinate system of an axes group. The position is valid only if the output `Valid` is true which is achieved by setting the input `Enable` on true.

Inputs

- **uAxesGroup**: Axes group reference
- **Enable**: Block function is enabled
- **CoordSystem**: Reference to the coordinate system used
  1. ... ACS
  2. ... MCS
  3. ... PCS

Outputs

- **yAxesGroup**: Axes group reference
- **Valid**: Output value is valid
- **Busy**: Algorithm not finished yet
- **Error**: Error occurred
- **ErrorID**: Result of the last operation
  1. ... REXYGEN general error
- **Acceleration**: Reference
MC_GroupStop – Stopping a group movement

Block Symbol Licence: COORDINATED MOTION

Function Description
The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

- **uAxesGroup**: Axes group reference
- **Execute**: The block is activated on rising edge
- **Deceleration**: Maximal allowed deceleration [unit/s²]
- **Jerk**: Maximal allowed jerk [unit/s³]

Outputs

- **yAxesGroup**: Axes group reference
- **Done**: Algorithm finished
- **CommandAborted**: Algorithm was aborted
- **Busy**: Algorithm not finished yet
- **Active**: The block is controlling the axis
- **Error**: Error occurred
- **ErrorID**: Result of the last operation
  - i ..... REXYGEN general error
CHAPTER 20. MC_COORD - MOTION CONTROL - COORDINATED MOVEMENT BLOCKS

Execute - MC_MoveLinearAbsolute

Done - MC_MoveLinearAbsolute

Error - MC_MoveLinearAbsolute

Execute - MC_GroupStop

Done - MC_GroupStop

Velocity AxesGroup

Position AxesGroup

Time [s]

Position [rad]

Velocity [rad/s]

commanded position x

commanded position y

0 1 2 3 4 5 6 7 8 9 10
0
0.5
1
bool
Execute

MC_MoveLinearAbsolute

0 1 2 3 4 5 6 7 8 9 10
0
0.5
1
bool
Done

MC_MoveLinearAbsolute

0 1 2 3 4 5 6 7 8 9 10
0
0.5
1
bool
Error

MC_MoveLinearAbsolute

0 1 2 3 4 5 6 7 8 9 10
0
0.5
1
bool
Execute

MC_GroupStop

0 1 2 3 4 5 6 7 8 9 10
0
0.5
1
bool
Done

MC_GroupStop

0 1 2 3 4 5 6 7 8 9 10
0.2
0.4
0.6
0.8
1
Velocity [rad/s]

0 1 2 3 4 5 6 7 8 9 10
0
0.6
0.8
1.2
Position [rad]

x-axis

y-axis

Position AxesGroup

Velocity AxesGroup
MC_GroupHalt – Stopping a group movement (interruptible)

Block Symbol

Licence: COORDINATED MOTION

Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

\textbf{uAxesGroup} Axes group reference \hspace{1cm} \textbf{Reference}

\textbf{Execute} The block is activated on rising edge \hspace{1cm} \textbf{Bool}

\textbf{Deceleration} Maximal allowed deceleration [unit/s\(^2\)] \hspace{1cm} \textbf{Double (F64)}

\textbf{Jerk} Maximal allowed jerk [unit/s\(^3\)] \hspace{1cm} \textbf{Double (F64)}

Outputs

\textbf{yAxesGroup} Axes group reference \hspace{1cm} \textbf{Reference}

\textbf{Done} Algorithm finished \hspace{1cm} \textbf{Bool}

\textbf{CommandAborted} Algorithm was aborted \hspace{1cm} \textbf{Bool}

\textbf{Busy} Algorithm not finished yet \hspace{1cm} \textbf{Bool}

\textbf{Active} The block is controlling the axis \hspace{1cm} \textbf{Bool}

\textbf{Error} Error occurred \hspace{1cm} \textbf{Bool}

\textbf{ErrorID} Result of the last operation \hspace{1cm} \textbf{Error}

\textbf{i \ldots REXYGEN general error}
MC_GroupInterrupt, MCP_GroupInterrupt – Read a group interrupt

Block Symbols

Function Description

The MC_GroupInterrupt and MCP_GroupInterrupt blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP version of the block.

The function block MC_GroupInterrupt interrupts the on-going motion and stops the group from moving, however does not abort the interrupted motion (meaning that at the interrupted FB the output CommandAborted will not be Set, Busy is still high and Active is reset). It stores all relevant track or path information internally at the moment it becomes active. The uAxesGroup stays in the original state even if the velocity zero is reached and the Done output is set.

Note 1: This function block is complementary to the function block MC_GroupContinue which execution the uAxesGroup state is reset to the original state (before MC_GroupInterrupt execution)

Inputs

<table>
<thead>
<tr>
<th>uAxesGroup</th>
<th>Axes group reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>yAxesGroup</th>
<th>Axes group reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td></td>
<td>i ..... REXYGEN general error</td>
</tr>
</tbody>
</table>

Licence: COORDINATED MOTION
MC_GroupContinue – Continuation of interrupted movement

Block Symbol

Licence: COORDINATED MOTION

Function Description

The function block MC_GroupContinue transfers the program back to the situation at issuing MC_GroupInterrupt. It uses internally the data set as stored at issuing MC_GroupInterrupt, and at the end (output Done set) transfer the control on the group back to the original FB doing the movements on the axes group, meaning also that at the originally interrupted FB the output Busy is still high and the output Active is set again.

Inputs

- **uAxesGroup**  Axes group reference
- **Execute**  The block is activated on rising edge

Outputs

- **yAxesGroup**  Axes group reference
- **Done**  Algorithm finished
- **Busy**  Algorithm not finished yet
- **CommandAborted**  Algorithm was aborted
- **Error**  Error occurred
- **ErrorID**  Result of the last operation
  
  i ..... REXYGEN general error
MC_GroupReadStatus – Read a group status

Block Symbol

Licence: COORDINATED MOTION

Function Description

The function block MC_GroupReadStatus returns the status of the uAxesGroup. The status is valid only if the output Valid is true which is achieved by setting the input Enable on true.

Inputs

<table>
<thead>
<tr>
<th>uAxesGroup</th>
<th>Axes group reference</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Block function is enabled</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>yAxesGroup</th>
<th>Axes group reference</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>Output value is valid</td>
<td>Bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>Bool</td>
</tr>
<tr>
<td>GroupMoving</td>
<td>State GroupMoving</td>
<td>Bool</td>
</tr>
<tr>
<td>GroupHoming</td>
<td>State GroupHoming</td>
<td>Bool</td>
</tr>
<tr>
<td>GroupErrorStop</td>
<td>State ErrorStop</td>
<td>Bool</td>
</tr>
<tr>
<td>GroupStandby</td>
<td>State Standby</td>
<td>Bool</td>
</tr>
<tr>
<td>GroupStopping</td>
<td>State Stopping</td>
<td>Bool</td>
</tr>
<tr>
<td>GroupDisabled</td>
<td>State Disabled</td>
<td>Bool</td>
</tr>
<tr>
<td>ConstantVelocity</td>
<td>Constant velocity motion</td>
<td>Bool</td>
</tr>
<tr>
<td>Accelerating</td>
<td>Accelerating</td>
<td>Bool</td>
</tr>
<tr>
<td>Decelerating</td>
<td>Decelerating</td>
<td>Bool</td>
</tr>
<tr>
<td>InPosition</td>
<td>Symptom achieve the desired position</td>
<td>Bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>Bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>Error</td>
</tr>
</tbody>
</table>

i ...... REXYGEN general error
MC_GroupReadError – Read a group error

Block Symbol

Licence: COORDINATED MOTION

Function Description

The function block MC_GroupReadError describes general error on the uAxesGroup which is not relating to the function blocks. If the output GroupErrorID is equal to 0 there is no error on the axes group. The actual error code GroupErrorID is valid only if the output Valid is true which is achieved by setting the input Enable on true.

Note 1: This function block is implemented because of compatibility with the PLCopen norm. The same error value is on the output ErrorID of the function block RM_AxesGroup.

Inputs

| uAxesGroup | Axes group reference     | Reference |
| Enable     | Block function is enabled | Bool      |

Outputs

| yAxesGroup | Axes group reference     | Reference |
| Valid      | Output value is valid    | Bool      |
| Busy       | Algorithm not finished yet | Bool      |
| Error      | Error occurred           | Error     |
| ErrorID    | Result of the last operation | Error     |
| GroupErrorID | Result of the last operation | Error     |
MC_GroupReset – Reset axes errors

Block Symbol

Function Description

The function block MC_GroupReset makes the transition from the state "GroupErrorStop" to "GroupStandBy" by resetting all internal group-related errors. This function block also resets all axes in this group like the function block MC_Reset from the MC_SINGLE library.

Inputs

<table>
<thead>
<tr>
<th>uAxesGroup</th>
<th>Axes group reference</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>yAxesGroup</th>
<th>Axes group reference</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>Bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>Bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>Bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>Error</td>
</tr>
</tbody>
</table>

i . . . . REXYGEN general error
MC_MoveLinearAbsolute – Linear move to position (absolute coordinates)

Block Symbol  Licence: COORDINATED MOTION

Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

| uAxesGroup | Axes group reference | Reference |
| Execute | The block is activated on rising edge | Bool |
| Position | Array of coordinates (positions and orientations) | Reference |
| Velocity | Maximal allowed velocity [unit/s] | Double (F64) |
| Acceleration | Maximal allowed acceleration [unit/s²] | Double (F64) |
| Jerk | Maximal allowed jerk [unit/s³] | Double (F64) |
| CoordSystem | Reference to the coordinate system used | Long (I32) |
| 1 | ACS |
| 2 | MCS |
| 3 | PCS |
| BufferMode | Buffering mode | Long (I32) |
| 1 | Aborting |
| 2 | Buffered |
| 3 | Blending low |
| 4 | Blending high |
| 5 | Blending previous |
| 6 | Blending next |
**Chapter 20. MC_COORD - MOTION CONTROL - COORDINATED MOVEMENT BLOCKS**

**TransitionMode**  Transition mode in blending mode  Long (I32)

1  TMNone
2  TMStartVelocity
3  TMConstantVelocity
4  TMCornerDistance
5  TMMaxCornerDeviation
11  Smooth

**TransitionParameter**  Parameter for transition (depends on transition mode)  Double (F64)

**Outputs**

- **yAxesGroup**  Axes group reference  Reference
- **Done**  Algorithm finished  Bool
- **CommandAborted**  Algorithm was aborted  Bool
- **Busy**  Algorithm not finished yet  Bool
- **Active**  The block is controlling the axis  Bool
- **Error**  Error occurred  Bool
- **ErrorID**  Result of the last operation  Error

i  REXYGEN general error
Sequence of two complete motions (Done>Execute)
Execute - MC_MoveLinearAbsolute — Function Block 1

Active - MC_MoveLinearAbsolute — Function Block 1

Done - MC_MoveLinearAbsolute — Function Block 1

Execute - MC_MoveLinearAbsolute — Function Block 2

Active - MC_MoveLinearAbsolute — Function Block 2

Done - MC_MoveLinearAbsolute — Function Block 2

Velocity AxesGroup

Position AxesGroup

commanded position x,y
MC_MoveLinearRelative – Linear move to position (relative to execution point)

Block Symbol

Licence: COORDINATED MOTION

Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

| uAxesGroup | Axes group reference | Reference |
| Execute | The block is activated on rising edge | Bool |
| Distance | Array of coordinates (relative distances and orientations) | Reference |
| Velocity | Maximal allowed velocity [unit/s] | Double (F64) |
| Acceleration | Maximal allowed acceleration [unit/s^2] | Double (F64) |
| Jerk | Maximal allowed jerk [unit/s^3] | Double (F64) |
| CoordSystem | Reference to the coordinate system used |
| 1 | ACS |
| 2 | MCS |
| 3 | PCS |
| BufferMode | Buffering mode | Long (I32) |
| 1 | Aborting |
| 2 | Buffered |
| 3 | Blending low |
| 4 | Blending high |
| 5 | Blending previous |
| 6 | Blending next |
**Chapter 20. MC_COORD - Motion Control - Coordinated Movement Blocks**

Transition Mode  Transition mode in blending mode  Long (I32)
1  TM None
2  TMStartVelocity
3  TMConstantVelocity
4  TMCornerDistance
5  TMMaxCornerDeviation
11  Smooth

Transition Parameter  Parameter for transition (depends on transition mode)  Double (F64)

### Outputs

- **yAxesGroup**  Axes group reference  Reference
- **Done**  Algorithm finished  Bool
- **Command Aborted**  Algorithm was aborted  Bool
- **Busy**  Algorithm not finished yet  Bool
- **Active**  The block is controlling the axis  Bool
- **Error**  Error occurred  Bool
- **ErrorID**  Result of the last operation  Error

- i  REXYGEN general error

---

**Diagrams**

[Diagram showing the process of motion control with variables like velocity, coordinates, and execution states.]

---
Sequence of two complete motions (Done>Execute)
Execute - MC_MoveLinearRelative -- Function Block 1

Active - MC_MoveLinearRelative -- Function Block 1

Done - MC_MoveLinearRelative -- Function Block 1

Execute - MC_MoveLinearRelative -- Function Block 2

Active - MC_MoveLinearRelative -- Function Block 2

Done - MC_MoveLinearRelative -- Function Block 2

Velocity AxesGroup

Position AxesGroup

Commanded position x,y
MC_MoveCircularAbsolute – Circular move to position (absolute coordinates)

Block Symbol

Licence: COORDINATED MOTION

Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

| uAxesGroup | Axes group reference | Reference |
| Execute    | The block is activated on rising edge | Bool |
| CircMode   | Specifies the meaning of the input signals AuxPoint and EndPoint | Long (132) |
|            | CircDirection         |         |
|            | 1 …… BORDER           |         |
|            | 2 …… CENTER           |         |
|            | 3 …… RADIUS           |         |
| AuxPoint   | Next coordinates to define circle (depend on CircMode) | Reference |
| EndPoint   | Target axes coordinates position | Reference |
### PathChoice

**Choice of path**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Clockwise</td>
</tr>
<tr>
<td>2</td>
<td>CounterClockwise</td>
</tr>
</tbody>
</table>

**Velocity**

Maximal allowed velocity [unit/s]

**Acceleration**

Maximal allowed acceleration [unit/s²]

**Jerk**

Maximal allowed jerk [unit/s³]

**CoordSystem**

Reference to the coordinate system used

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACS</td>
</tr>
<tr>
<td>2</td>
<td>MCS</td>
</tr>
<tr>
<td>3</td>
<td>PCS</td>
</tr>
</tbody>
</table>

**BufferMode**

Buffering mode

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aborting</td>
</tr>
<tr>
<td>2</td>
<td>Buffered</td>
</tr>
<tr>
<td>3</td>
<td>Blending low</td>
</tr>
<tr>
<td>4</td>
<td>Blending high</td>
</tr>
<tr>
<td>5</td>
<td>Blending previous</td>
</tr>
<tr>
<td>6</td>
<td>Blending next</td>
</tr>
</tbody>
</table>

**TransitionMode**

Transition mode in blending mode

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TMNone</td>
</tr>
<tr>
<td>2</td>
<td>TMStartVelocity</td>
</tr>
<tr>
<td>3</td>
<td>TMConstantVelocity</td>
</tr>
<tr>
<td>4</td>
<td>TMCornerDistance</td>
</tr>
<tr>
<td>5</td>
<td>TMMaxCornerDeviation</td>
</tr>
<tr>
<td>11</td>
<td>Smooth</td>
</tr>
</tbody>
</table>

**TransitionParameter**

Parameter for transition (depends on transition mode)

Reference (I32)

### Outputs

**yAxesGroup**

Axes group reference

Reference

**Done**

Algorithm finished

Reference

**CommandAborted**

Algorithm was aborted

Reference

**Busy**

Algorithm not finished yet

Reference

**Active**

The block is controlling the axis

Reference

**Error**

Error occurred

Reference

**ErrorID**

Result of the last operation

Reference

iR EREXGEN general error
MC_MoveCircularRelative – Circular move to position (relative to execution point)

Block Symbol

Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Axes group reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>CircMode</td>
<td>Specifies the meaning of the input signals AuxPoint and EndPoint.</td>
</tr>
<tr>
<td></td>
<td>Long (132)</td>
</tr>
<tr>
<td>CircDirection</td>
<td>1 ...... BORDER</td>
</tr>
<tr>
<td></td>
<td>2 ...... CENTER</td>
</tr>
<tr>
<td></td>
<td>3 ...... RADIUS</td>
</tr>
<tr>
<td>AuxPoint</td>
<td>Next coordinates to define circle (depend on CircMode)</td>
</tr>
<tr>
<td>EndPoint</td>
<td>Target axes coordinates position</td>
</tr>
</tbody>
</table>

Reference
### Outputs

<table>
<thead>
<tr>
<th>yAxesGroup</th>
<th>Axes group reference</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>Bool</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
<td>Bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>Bool</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
<td>Bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>Bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>Error</td>
</tr>
<tr>
<td>i</td>
<td>REXYGEN general error</td>
<td></td>
</tr>
</tbody>
</table>
MC_MoveCircularRelative - Example
MC_MoveDirectAbsolute – Direct move to position (absolute coordinates)

Block Symbol  Licence: COORDINATED MOTION

Function Description
The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

<table>
<thead>
<tr>
<th>uAxesGroup</th>
<th>Axes group reference</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>Bool</td>
</tr>
<tr>
<td>Position</td>
<td>Array of coordinates (positions and orientations)</td>
<td>Reference</td>
</tr>
<tr>
<td>CoordSystem</td>
<td>Reference to the coordinate system used</td>
<td>Long (I32)</td>
</tr>
<tr>
<td></td>
<td>1 ...... ACS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ...... MCS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 ...... PCS</td>
<td></td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
<td>Long (I32)</td>
</tr>
<tr>
<td></td>
<td>1 ...... Aborting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ...... Buffered</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 ...... Blending low</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 ...... Blending high</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 ...... Blending previous</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6 ...... Blending next</td>
<td></td>
</tr>
<tr>
<td>TransitionMode</td>
<td>Transition mode in blending mode</td>
<td>Long (I32)</td>
</tr>
<tr>
<td></td>
<td>1 ...... TMNone</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ...... TMStartVelocity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 ...... TMConstantVelocity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 ...... TMCornerDistance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 ...... TMMaxCornerDeviation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11 ...... Smooth</td>
<td></td>
</tr>
<tr>
<td>TransitionParameter</td>
<td>Parameter for transition (depends on transition mode)</td>
<td>Double (F64)</td>
</tr>
</tbody>
</table>
Outputs

yAxesGroup  Axes group reference

Done  Algorithm finished

CommandAborted  Algorithm was aborted

Busy  Algorithm not finished yet

Active  The block is controlling the axis

Error  Error occurred

ErrorID  Result of the last operation

i . . . . . REXYGEN general error
**MC_MoveDirectRelative** – **Direct move to position** (relative to execution point)

<table>
<thead>
<tr>
<th>Block Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Licence: <strong>COORDINATED MOTION</strong></td>
</tr>
</tbody>
</table>

**Function Description**

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

**Inputs**

- **uAxesGroup** Axes group reference
- **Execute** The block is activated on rising edge
- **Distance** Array of coordinates (relative distances and orientations)
- **CoordSystem** Reference to the coordinate system used
  - 1 ..... ACS
  - 2 ..... MCS
  - 3 ..... PCS
- **BufferMode** Buffering mode
  - 1 ..... Aborting
  - 2 ..... Buffered
  - 3 ..... Blending low
  - 4 ..... Blending high
  - 5 ..... Blending previous
  - 6 ..... Blending next
- **TransitionMode** Transition mode in blending mode
  - 1 ..... TMNone
  - 2 ..... TMStartVelocity
  - 3 ..... TMConstantVelocity
  - 4 ..... TMCornerDistance
  - 5 ..... TMMaxCornerDeviation
  - 11 ..... Smooth
- **TransitionParameter** Parameter for transition (depends on transition mode)

**Outputs**

- **Done**
- **Active**
- **Busy**
- **CommandAborted**
- **Error**
- **ErrorID**
Outputs

| yAxesGroup | Axes group reference | Reference |
| CommandAborted | Algorithm was aborted | Bool |
| Busy | Algorithm not finished yet | Bool |
| Active | The block is controlling the axis | Bool |
| Error | Error occurred | Bool |
| ErrorID | Result of the last operation | Error |

i . . . . . REXYGEN general error
MC_MovePath – General spatial trajectory generation

Block Symbol

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Axes group reference</td>
<td>Reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>Bool</td>
</tr>
<tr>
<td>TotalTime</td>
<td>Time [s] for whole move</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>RampTime</td>
<td>Time [s] for acceleration/deceleration</td>
<td>Double (F64)</td>
</tr>
<tr>
<td>CoordSystem</td>
<td>Reference to the coordinate system used</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
<td>Long (I32)</td>
</tr>
<tr>
<td>TransitionMode</td>
<td>Transition mode in blending mode</td>
<td>Long (I32)</td>
</tr>
</tbody>
</table>

Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

- **uAxesGroup**: Axes group reference
- **Execute**: The block is activated on rising edge
- **TotalTime**: Time [s] for whole move
- **RampTime**: Time [s] for acceleration/deceleration
- **CoordSystem**: Reference to the coordinate system used
  
  1. ACS
  2. MCS
  3. PCS
- **BufferMode**: Buffering mode
  
  1. Aborting
  2. Buffered
  3. Blending low
  4. Blending high
  5. Blending previous
  6. Blending next
- **TransitionMode**: Transition mode in blending mode
  
  1. TMNone
  2. TMStartVelocity
  3. TMConstantVelocity
  4. TMCornerDistance
  5. TMMaxCornerDeviation
  6. Smooth
**Chapter 20. MC_COORD - Motion Control - Coordinated Movement Blocks**

**TransitionParameter**  Parameter for transition (depends on transition mode)  Double (F64)

**RampIn**  RampIn factor (0 = RampIn mode not used)  Double (F64)

**Parameters**

- **pc**  Control-points matrix  Double (F64)
  
  \[ [0.0 \ 1.0 \ 2.0; \ 0.0 \ 1.0 \ 1.0; \ 0.0 \ 1.0 \ 0.0] \]

- **pk**  Knot-points vector  Double (F64)
  
  \[ [0.0 \ 0.0 \ 0.0 \ 0.0 \ 0.5 \ 1.0 \ 1.0] \]

- **pw**  Weighting vector  Double (F64)
  
  \[ [1.0 \ 1.0 \ 1.0] \]

- **pv**  Polynomials for feedrate definition  Double (F64)
  
  \[ [0.0 \ 0.05 \ 0.95; \ 0.0 \ 0.1 \ 0.1; \ 0.0 \ 0.0 \ 0.0; \ 0.1 \ 0.0 \ -0.1; \ -0.05 \ 0.0 \ 0.05; \ 0.0 \ 0.0 \ 0.0] \]

- **pt**  Knot-points (time [s]) for feedrate  Double (F64)
  
  \[ [0.0 \ 1.0 \ 10.0 \ 11.0] \]

- **user**  Only for special edit  Double (F64)
  
  \[ [0.0 \ 1.0 \ 2.0 \ 3.0] \]

**Outputs**

- **yAxesGroup**  Axes group reference  Reference

- **Done**  Algorithm finished  Bool

- **CommandAborted**  Algorithm was aborted  Bool

- **Busy**  Algorithm not finished yet  Bool

- **Active**  The block is controlling the axis  Bool

- **Error**  Error occurred  Bool

- **ErrorID**  Result of the last operation  Error

  \[ i \dots \ \text{REXYGEN general error} \]
MC_GroupSetOverride – Set group override factors

Block Symbol

Function Description
The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

| uAxesGroup | Axes group reference | Reference |
| Enable | Block function is enabled | Bool |
| VelFactor | Velocity multiplication factor | Double (F64) |
| AccFactor | Acceleration/deceleration multiplication factor | Double (F64) |
| JerkFactor | Jerk multiplication factor | Double (F64) |

Parameter

diff | Deadband (difference for recalculation) | 0.05 | Double (F64) |

Outputs

| yAxesGroup | Axes group reference | Reference |
| Enabled | Signal that the override faktor are set successfully | Bool |
| Busy | Algorithm not finished yet | Bool |
| Error | Error occurred | Bool |
| ErrorID | Result of the last operation | Error |
| i | REXYGEN general error |
Chapter 21

CanDrv – Communication via CAN bus

Contents

CanItem – Secondary received CAN message .................. 640
CanRecv – Receive CAN message ............................... 641
CanSend – Send CAN message ................................. 643
CanItem – Secondary received CAN message

Block Symbol

Licence: CANDRV

Function Description

The block is used with the CanRecv block. The uRef input of the CanItem block must be connected to the itemRef output of some CanRecv block or to the yRef output of another CanItem block.

This block shows the previous message that has passed the filter in the CanRecv block.

If more than one CanItem block is connected (directly or indirectly through the yRef output of the CanItem block already connected to the CanRecv block) then the first executed CanItem block shows the first message before the last received message (which is shown by the CanRecv block), the second executed CanItem block shows the second message before the last received message (which is shown by the CanRecv block) etc. It is strongly recommended to connect the CanItem blocks in a daisy chain. Unexpected ordering of messages may occur if the blocks are connected in a tree-like structure.

If no message has been received since start of the CAN driver, the data outputs have fallback values msgId = -1 and length = -1.

The DRDY output is set to DRDY = on if the message has been received during the last period, i.e. after previous execution of the CanItem block. At the same moment, the outputs msgId, data and length are updated. If there is no new data, DRDY output is set to DRDY = off and the data values are kept on the other outputs (msgId, data and length).

Input

- **uRef**: Secondary received packet reference

Output

- **yRef**: Secondary received packet reference
- **msgId**: CAN message ID (COB-ID)
- **data**: Message data (8 bytes maximum, LSB first)
- **length**: Message length (number of bytes of data)
- **DRDY**: Received message in the last period flag

- **Reference**: Reference
- **Long (I32)**: Long (I32)
- **Large (I64)**: Large (I64)
- **↓-9.22337E+18 †9.22337E+18**: ↓-9.22337E+18 †9.22337E+18
- **↓0 ↑8**: ↓0 ↑8
- **Bool**: Bool
CanRecv – Receive CAN message

Block Symbol

Licence: CANDRV

Function Description

The CanRecv block receives message via CAN bus. The message is defined by the msgId, data and length inputs and the RTR and EXT parameters.

Number of messages received in the current task period (i.e. since the previous execution) is indicated by the nDRDY output.

The data from the last received message is available at the msgId, data and length outputs. Previous messages (with respect to the nmax parameter) are available using the CanItem block(s) linked to the itemRef output.

The block must be linked with the CANDrv driver. The driver must be configured to use the simple CAN mode (i.e. the parameter NodeMode = 256).

The block’s name must be in the form <DRV>_--<blkname> (see e.g. OUTQUAD or OUTOCT blocks for details about referencing data from I/O drivers). The <blkname> part of the name has no special meaning in this case and it is recommended to keep the original CanRecv.

The block supports short (11-bit) and long (29-bit) message IDs (see the EXT parameter) and RequestToReceive messages (see the RTR parameter). FD mode which allows up to 64 data bytes in a single message is not supported.

Outputs

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>itemRef</td>
<td>Secondary received packet reference</td>
<td>Reference</td>
</tr>
<tr>
<td>msgId</td>
<td>CAN message ID (COB-ID)</td>
<td>Long (132)</td>
</tr>
<tr>
<td>data</td>
<td>Message data (8 bytes maximum, LSB first)</td>
<td>Large (164)</td>
</tr>
<tr>
<td>length</td>
<td>Message length (number of bytes of data)</td>
<td>↓0 ↑8 Long (132)</td>
</tr>
<tr>
<td>nDRDY</td>
<td>Number of received messages in the last period</td>
<td>↑255 Word (016)</td>
</tr>
<tr>
<td>iErr</td>
<td>Error code</td>
<td>Error</td>
</tr>
<tr>
<td>age</td>
<td>Elapsed time since the last received message [s]</td>
<td>↓0.0 Double (F64)</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Default Value</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filterId</code></td>
<td>Message ID of packets to receive by this block</td>
<td>Long (I32)</td>
<td>0</td>
<td>0</td>
<td>536870911</td>
</tr>
<tr>
<td><code>filterIdMask</code></td>
<td>Mask for the <code>filterId</code> parameter (marks valid bits)</td>
<td>Long (I32)</td>
<td>0</td>
<td>0</td>
<td>536870911</td>
</tr>
<tr>
<td><code>filterLength</code></td>
<td>Data length of packets to receive by this block (-1 allows all lengths)</td>
<td>Long (I32)</td>
<td>-1</td>
<td>-8</td>
<td>8</td>
</tr>
<tr>
<td><code>RTR</code></td>
<td>Request To Receive flag</td>
<td>Bool</td>
<td>on</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>EXT</code></td>
<td>Extended message ID (29 bits)</td>
<td>Bool</td>
<td>on</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>Error is indicated if no packet is received within the timeout interval [s]</td>
<td>Double (F64)</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td><code>nmax</code></td>
<td>Maximum number of received messages in one period</td>
<td>Long (I32)</td>
<td>1</td>
<td>1</td>
<td>255</td>
</tr>
</tbody>
</table>
CanSend – Send CAN message

Block Symbol

Function Description

The CanSend block sends message via CAN bus. The message content is defined by the msgId, data and length inputs and the RTR and EXT parameters. Message is sent only if the input RUN is set to RUN = on.

The block must be linked with the CanDrv driver. The driver must be configured to use the simple CAN mode (i.e. the parameter NodeMode = 256).

The block’s name must be in the form <DRV>__<blkname> (see e.g. OUTQUAD or OUTOCT blocks for details about referencing data from I/O drivers). The <blkname> part of the name has no special meaning in this case and it is recommended to keep the original CanSend.

The block supports short (11-bit) and long (29-bit) message IDs (see the EXT parameter) and RequestToReceive messages (see the RTR parameter). FD mode which allows up to 64 data bytes in a single message is not supported.

Inputs

| msgId     | CAN message ID (COB-ID) | ↓0 ↑536870911 | Long (132) |
| length    | Message length (number of bytes of data) | ↓0 ↑18 | Long (132) |
| data      | Message data (8 bytes maximum, LSB first) | ↓-9.22337E+18 ↑9.22337E+18 | Large (164) |
| RUN       | Sending message is enabled | Bool |

Output

<table>
<thead>
<tr>
<th>iErr</th>
<th>Error code</th>
<th>Error</th>
</tr>
</thead>
</table>

Parameters

<table>
<thead>
<tr>
<th>RTR</th>
<th>Request To Receive flag</th>
<th>on</th>
<th>Bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXT</td>
<td>Extended message ID (29bits)</td>
<td>on</td>
<td>Bool</td>
</tr>
</tbody>
</table>
Chapter 22

OpcUaDrv – Communication using OPC UA

Contents

OpcUaReadValue – Read value from OPC UA Server . . . . . . . . 646
OpcUaServerValue – Expose value as an OPC UA Node . . . . . . 648
OpcUaWriteValue – Write value to OPC UA Server . . . . . . . . 650
OpcUaReadValue – Read value from OPC UA Server

Block Symbol

Licence: ADVANCED

Function Description

This function block depends on the OpcUa driver. Please read the OpcUaDrv manual [12] before use.

The OpcUaReadValue block reads value of an OPC UA Node through a connection established by the OPC UA client driver.

The first two parameters are NodeId and NodeId_type. The NodeId_type specifies what type of information it is expected to be entered as the NodeId parameter. If the value is string, numeric, guid than the NodeId parameter should contain the id of the actual OPC UA Node on the server prefixed with the index of the namespace declared in the configuration of the driver separated by a colon (e.g. 1:myNode).

If the value of the NodeId_type parameter is set to path than the NodeId parameter should contain the path to the desired Node in the server structure. Every segment of the path is composed from the attribute BrowserName of the node and the BrowserName is similarly with regular NodeId types prefixed with the index of the namespace declared in the configuration of the driver separated by a colon (e.g. /1:myDevice/1:myNode). The path is relative to the Objects folder in the OPC UA server structure.

The parameter type specifies the expected Node’s value data type. The block converts the Node’s value to the specified type and sets the value output signal in case of success or it sets the errId to the resulting error code.

Input

READ Enable execution Bool

Parameters

<table>
<thead>
<tr>
<th>NodeId</th>
<th>OPC UA Node Id</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeId_type</td>
<td>Type of Node ID</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>string</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>numeric</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>guid</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>path</td>
<td>4</td>
</tr>
</tbody>
</table>

646 CHAPTER 22. OPCADRV – COMMUNICATION USING OPC UA
<table>
<thead>
<tr>
<th>type</th>
<th>Expected type of incoming data</th>
<th>⊙1</th>
<th>Long (132)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>double</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>bool</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>value</th>
<th>Output signal</th>
<th>Unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSY</td>
<td>Busy flag</td>
<td>Bool</td>
</tr>
<tr>
<td>DONE</td>
<td>Indicator of finished transaction</td>
<td>Bool</td>
</tr>
<tr>
<td>errId</td>
<td>Error code</td>
<td>Error</td>
</tr>
</tbody>
</table>
OpcUaServerValue – Expose value as an OPC UA Node

Block Symbol

Function Description

This function block depends on the OpcUa driver. Please read the OpcUaDrv manual [12] before use.

The OpcUaServerValue block exposes an OPC UA Node through OPC UA server driver.

The first two parameters are NodeId and NodeId_type. The NodeId_type specifies how the value entered as the NodeId parameter should be treated. The parameter NodeId specifies the NodeId that the OPC UA Node represented by the block should be exposed with.

The input DISABLE controls whether the OPC UA Node is exposed on the server or not. When the SET input is set to on the value on the input uValue port is set to the OPC UA Node’s value. If the parameter READONLY is set to off the Node’s value can also be changed from outside of the algorithm through the OPC UA communication protocol.

The output signal yValue is set to the Node’s value on every tick. The parameter type specifies the Node’s value data type, the data type of the uValue input and yValue output.

Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uValue</td>
<td>Input signal</td>
<td>Unknown</td>
</tr>
<tr>
<td>SET</td>
<td>Set the input value to OPC UA Node value</td>
<td>Bool</td>
</tr>
<tr>
<td>DISABLE</td>
<td>Disable OPC UA Node</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeId</td>
<td>OPC UA Node Id</td>
<td>String</td>
</tr>
<tr>
<td>NodeId_type</td>
<td>OPC UA Node Id type</td>
<td>guid</td>
</tr>
<tr>
<td></td>
<td>1 ... string</td>
<td>guid</td>
</tr>
<tr>
<td></td>
<td>2 ... numeric</td>
<td>guid</td>
</tr>
<tr>
<td></td>
<td>3 ... guid</td>
<td>guid</td>
</tr>
</tbody>
</table>
type | Value data type
---|---
1 | string
2 | double
3 | long
4 | bool

**BrowseName** | OPC UA Node Browse name | String
**Description** | OPC UA Node description | String
**DisplayName** | OPC UA Node display name | String
**READONLY** | Set OPC Node value as read only | On | Off

**Outputs**

| yValue | Output signal | Unknown
---|---|---
**CHANGED** | Value of the node changed through the OPC UA protocol | Bool
**errId** | Error code | Error
OpcUaWriteValue – Write value to OPC UA Server

Block Symbol

Licence: ADVANCED

Function Description

This function block depends on the OpcUa driver. Please read the OpcUadrv manual [12] before use.

The OpcUaWriteValue block writes value to the OPC UA Node through a connection established by the OPC UA client driver.

The first two parameters are NodeId and NodeId_type. The NodeId_type specifies what type of information it is expected to be entered as the NodeId parameter. If the value is string, numeric, guid then the NodeId parameter should contain the id of the actual OPC UA Node on the server prefixed with the index of the namespace declared in the configuration of the driver separated by a colon (e.g. 1:myNode).

If the value of the NodeId_type parameter is set to path then the NodeId parameter should contain the path to the desired Node in the server structure. Every segment of the path is composed from the attribute BrowserName of the node and the BrowserName is similarly with regular NodeId types prefixed with the index of the namespace declared in the configuration of the driver separated by a colon (e.g. /1:myDevice/1:myNode). The path is relative to the Objects folder in the OPC UA server structure.

The parameter type specifies the expected Node’s value data type. The input signal value is converted to the specified type and is then written to the Node’s value attribute.

When the process of writing the value is finished the result code defined by OPC UA is set to the code output and it’s textual representation is set to the status output.

Inputs

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Input signal</td>
<td>Unknown</td>
</tr>
<tr>
<td>WRITE</td>
<td>Enable execution</td>
<td>Bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeId</td>
<td>OPC UA Node Id</td>
<td>String</td>
</tr>
<tr>
<td>NodeId_type</td>
<td>Type of Node ID</td>
<td>Type</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------</td>
<td>------</td>
</tr>
<tr>
<td>1</td>
<td>string</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>numeric</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>guid</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>path</td>
<td>4</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>BUSY</th>
<th>Busy flag</th>
<th>Bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>DONE</td>
<td>Indicator of finished transaction</td>
<td>Bool</td>
</tr>
<tr>
<td>errId</td>
<td>Error code</td>
<td>Error</td>
</tr>
<tr>
<td>code</td>
<td>OPC UA result status code</td>
<td>DWord (U32)</td>
</tr>
<tr>
<td>status</td>
<td>OPC UA result status string</td>
<td>String</td>
</tr>
</tbody>
</table>
Appendix A

Licensing options

From the licensing point of view, there are several versions of the RexCore runtime module to provide maximum flexibility for individual projects. The table below compares the individual variants.

The function blocks are divided into several licensing groups. The STANDARD function blocks are always available, the other groups require activation by a corresponding licence.

<table>
<thead>
<tr>
<th>Function blocks</th>
<th>RexCore DEMO</th>
<th>RexCore Starter</th>
<th>RexCore Plus</th>
<th>RexCore Professional</th>
<th>RexCore Ultimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>STANDARD</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>ADVANCED</td>
<td>•</td>
<td>–</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>REXLANG</td>
<td>•</td>
<td>–</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>MOTION CONTROL</td>
<td>•</td>
<td>–</td>
<td>○</td>
<td>○</td>
<td>•</td>
</tr>
<tr>
<td>COORDINATED MOTION</td>
<td>•</td>
<td>–</td>
<td>○</td>
<td>○</td>
<td>•</td>
</tr>
<tr>
<td>AUTOTUNING</td>
<td>–</td>
<td>–</td>
<td>○</td>
<td>○</td>
<td>•</td>
</tr>
<tr>
<td>MATRIX</td>
<td>•</td>
<td>–</td>
<td>○</td>
<td>○</td>
<td>•</td>
</tr>
<tr>
<td>I/O drivers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic I/O drivers</td>
<td>•</td>
<td>–</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Additional I/O drivers</td>
<td>•</td>
<td>○</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>

(• ... included, ○ ... optional, – ... not available)

See Appendix B for details about licensing of individual function blocks.
Appendix B

Licensing of individual function blocks

To maximize flexibility for individual projects, function blocks of the RE<sub>XYGEN</sub> system are divided into several licensing groups. The table below shows the groups the function blocks belong to. See Appendix A for detailed information about the individual licensing options.

<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STANDARD</td>
</tr>
<tr>
<td>ABS_</td>
<td></td>
</tr>
<tr>
<td>ABSROT</td>
<td></td>
</tr>
<tr>
<td>ACD</td>
<td>●</td>
</tr>
<tr>
<td>ADD</td>
<td>●</td>
</tr>
<tr>
<td>ADDHEXD</td>
<td>●</td>
</tr>
<tr>
<td>ADDOCT</td>
<td>●</td>
</tr>
<tr>
<td>ADDQUAD</td>
<td>●</td>
</tr>
<tr>
<td>AFLUSH</td>
<td>●</td>
</tr>
<tr>
<td>ALB</td>
<td>●</td>
</tr>
<tr>
<td>ALBI</td>
<td>●</td>
</tr>
<tr>
<td>ALN</td>
<td>●</td>
</tr>
<tr>
<td>ALNI</td>
<td>●</td>
</tr>
<tr>
<td>AND_</td>
<td>●</td>
</tr>
<tr>
<td>ANDHEXD</td>
<td>●</td>
</tr>
<tr>
<td>ANDOCT</td>
<td>●</td>
</tr>
<tr>
<td>ANDQUAD</td>
<td>●</td>
</tr>
<tr>
<td>ANLS</td>
<td>●</td>
</tr>
<tr>
<td>ARC</td>
<td>●</td>
</tr>
<tr>
<td>ARLY</td>
<td>●</td>
</tr>
<tr>
<td>ARS</td>
<td>●</td>
</tr>
</tbody>
</table>

The list continues on the next page...
## Appendix B. Licensing of Individual Function Blocks

<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASW</td>
<td>STANDARD</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>ATMT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BDHEXD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BDQCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BINS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BISR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BITOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BMHEXD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BMQCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BPF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CanItem</td>
<td>CANDRV</td>
<td></td>
</tr>
<tr>
<td>CanRecv</td>
<td>CANDRV</td>
<td></td>
</tr>
<tr>
<td>CanSend</td>
<td>CANDRV</td>
<td></td>
</tr>
<tr>
<td>CDELSMM</td>
<td></td>
<td>ADVANCED</td>
</tr>
<tr>
<td>CMP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNDR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONCAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COUNT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSSM</td>
<td></td>
<td>ADVANCED</td>
</tr>
<tr>
<td>DATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATETIME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDELSMM</td>
<td></td>
<td>ADVANCED</td>
</tr>
<tr>
<td>DEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DFRIR</td>
<td></td>
<td>ADVANCED</td>
</tr>
<tr>
<td>DIF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Display</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIV</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*The list continues on the next page...*
<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STANDARD</td>
</tr>
<tr>
<td>DSSM</td>
<td></td>
</tr>
<tr>
<td>EAS</td>
<td>●</td>
</tr>
<tr>
<td>EATMT</td>
<td></td>
</tr>
<tr>
<td>EDGE_</td>
<td>●</td>
</tr>
<tr>
<td>EKF</td>
<td></td>
</tr>
<tr>
<td>EMD</td>
<td>●</td>
</tr>
<tr>
<td>EPC</td>
<td></td>
</tr>
<tr>
<td>EQ</td>
<td>●</td>
</tr>
<tr>
<td>EVAR</td>
<td>●</td>
</tr>
<tr>
<td>EXEC</td>
<td>●</td>
</tr>
<tr>
<td>FIND</td>
<td>●</td>
</tr>
<tr>
<td>FLDCU</td>
<td></td>
</tr>
<tr>
<td>FXH</td>
<td>●</td>
</tr>
<tr>
<td>FNXY</td>
<td>●</td>
</tr>
<tr>
<td>FOPDT</td>
<td>●</td>
</tr>
<tr>
<td>FRID</td>
<td></td>
</tr>
<tr>
<td>From</td>
<td>●</td>
</tr>
<tr>
<td>GAIN</td>
<td>●</td>
</tr>
<tr>
<td>GETPA</td>
<td>●</td>
</tr>
<tr>
<td>GETPB</td>
<td>●</td>
</tr>
<tr>
<td>GETPI</td>
<td>●</td>
</tr>
<tr>
<td>GETPR</td>
<td>●</td>
</tr>
<tr>
<td>GETPS</td>
<td>●</td>
</tr>
<tr>
<td>Goto</td>
<td>●</td>
</tr>
<tr>
<td>GotoTagVisibility</td>
<td>●</td>
</tr>
<tr>
<td>GRADS</td>
<td></td>
</tr>
<tr>
<td>HMI</td>
<td>●</td>
</tr>
<tr>
<td>HTTP</td>
<td></td>
</tr>
<tr>
<td>HTTP2</td>
<td></td>
</tr>
<tr>
<td>I3PM</td>
<td></td>
</tr>
<tr>
<td>IADD</td>
<td>●</td>
</tr>
<tr>
<td>IDIV</td>
<td>●</td>
</tr>
<tr>
<td>IMOD</td>
<td>●</td>
</tr>
<tr>
<td>IMUL</td>
<td>●</td>
</tr>
<tr>
<td>INFO</td>
<td>●</td>
</tr>
<tr>
<td>INHEXD</td>
<td>●</td>
</tr>
<tr>
<td>INOCT</td>
<td>●</td>
</tr>
<tr>
<td>Import</td>
<td>●</td>
</tr>
<tr>
<td>INQUAD</td>
<td>●</td>
</tr>
</tbody>
</table>

*The list continues on the next page...*
<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTD</td>
<td>•</td>
</tr>
<tr>
<td>INTE</td>
<td>•</td>
</tr>
<tr>
<td>INTSM</td>
<td>•</td>
</tr>
<tr>
<td>IODRV</td>
<td>•</td>
</tr>
<tr>
<td>IOTASK</td>
<td>•</td>
</tr>
<tr>
<td>ISSW</td>
<td>•</td>
</tr>
<tr>
<td>ISUB</td>
<td>•</td>
</tr>
<tr>
<td>ITOI</td>
<td>•</td>
</tr>
<tr>
<td>ITOS</td>
<td>•</td>
</tr>
<tr>
<td>KDER</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>LC</td>
<td>•</td>
</tr>
<tr>
<td>LEN</td>
<td>•</td>
</tr>
<tr>
<td>LIN</td>
<td>•</td>
</tr>
<tr>
<td>LLC</td>
<td>•</td>
</tr>
<tr>
<td>LPBK</td>
<td>•</td>
</tr>
<tr>
<td>LPF</td>
<td>•</td>
</tr>
<tr>
<td>MC_AccelerationProfile</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_AddAxisToGroup</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_CamIn</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_CamOut</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_CombineAxes</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GearIn</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GearInPos</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GearOut</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GroupContinue</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_GroupDisable</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_GroupEnable</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_GroupHalt</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_GroupInterrupt</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_GroupReadActualAcceleration</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_GroupReadActualPosition</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_GroupReadActualVelocity</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_GroupReadError</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_GroupReadStatus</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_GroupReset</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_GroupSetOverride</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_GroupSetPosition</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_GroupStop</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_Halt</td>
<td>MOTION CONTROL</td>
</tr>
</tbody>
</table>

The list continues on the next page...
<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC_HaltSuperimposed</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_Home</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveAbsolute</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveAdditive</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveCircularAbsolute</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_MoveCircularRelative</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_MoveContinuousAbsolute</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveContinuousRelative</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveDirectAbsolute</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_MoveDirectRelative</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_MoveLinearAbsolute</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_MoveLinearRelative</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_MovePath</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_MovePath_PH</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_MoveRelative</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveSuperimposed</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveVelocity</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_PhasingAbsolute</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_PhasingRelative</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_PositionProfile</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_Power</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_ReadActualPosition</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_ReadAxisError</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_ReadBoolParameter</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_ReadCartesianTransform</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_ReadParameter</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_ReadStatus</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_Reset</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_SetCartesianTransform</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_SetOverride</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_Stop</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_TorqueControl</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_UngroupAllAxes</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_VelocityProfile</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_WriteBoolParameter</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_WriteParameter</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_AccelerationProfile</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_CamIn</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_CamTableSelect</td>
<td>MOTION CONTROL</td>
</tr>
</tbody>
</table>

The list continues on the next page...
## APPENDIX B. LICENSING OF INDIVIDUAL FUNCTION BLOCKS

<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCP_CombineAxes</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_GearIn</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_GearInPos</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_GroupHalt</td>
<td>COORDINATED MOTION</td>
<td></td>
</tr>
<tr>
<td>MCP_GroupInterrupt</td>
<td>COORDINATED MOTION</td>
<td></td>
</tr>
<tr>
<td>MCP_GroupSetOverride</td>
<td>COORDINATED MOTION</td>
<td></td>
</tr>
<tr>
<td>MCP_GroupSetPosition</td>
<td>COORDINATED MOTION</td>
<td></td>
</tr>
<tr>
<td>MCP_GroupStop</td>
<td>COORDINATED MOTION</td>
<td></td>
</tr>
<tr>
<td>MCP_Halt</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_HaltSuperimposed</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_Home</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_MoveAbsolute</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_MoveAdditive</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_MoveCircularAbsolute</td>
<td>COORDINATED MOTION</td>
<td></td>
</tr>
<tr>
<td>MCP_MoveCircularRelative</td>
<td>COORDINATED MOTION</td>
<td></td>
</tr>
<tr>
<td>MCP_MoveContinuousAbsolute</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_MoveContinuousRelative</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_MoveDirectAbsolute</td>
<td>COORDINATED MOTION</td>
<td></td>
</tr>
<tr>
<td>MCP_MoveDirectRelative</td>
<td>COORDINATED MOTION</td>
<td></td>
</tr>
<tr>
<td>MCP_MoveLinearAbsolute</td>
<td>COORDINATED MOTION</td>
<td></td>
</tr>
<tr>
<td>MCP_MoveLinearRelative</td>
<td>COORDINATED MOTION</td>
<td></td>
</tr>
<tr>
<td>MCP_MovePath</td>
<td>COORDINATED MOTION</td>
<td></td>
</tr>
<tr>
<td>MCP_MovePath_PH</td>
<td>COORDINATED MOTION</td>
<td></td>
</tr>
<tr>
<td>MCP_MoveRelative</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_MoveSuperimposed</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_MoveVelocity</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_PhasingAbsolute</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_PhasingRelative</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_PositionProfile</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_SetCartesianTransform</td>
<td>COORDINATED MOTION</td>
<td></td>
</tr>
<tr>
<td>MCP_SetKinTransform_Arm</td>
<td>COORDINATED MOTION</td>
<td></td>
</tr>
<tr>
<td>MCP_SetOverride</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_Stop</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_TorqueControl</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCP_VelocityProfile</td>
<td>MOTION CONTROL</td>
<td></td>
</tr>
<tr>
<td>MCU</td>
<td>⋆</td>
<td></td>
</tr>
<tr>
<td>MDL</td>
<td>⋆</td>
<td></td>
</tr>
<tr>
<td>MDLI</td>
<td>⋆</td>
<td></td>
</tr>
<tr>
<td>MID</td>
<td>⋆</td>
<td></td>
</tr>
</tbody>
</table>

*The list continues on the next page...*
<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINMAX</td>
<td>●</td>
</tr>
<tr>
<td>MODULE</td>
<td>●</td>
</tr>
<tr>
<td>MP</td>
<td>●</td>
</tr>
<tr>
<td>MqttPublish</td>
<td>MQTTDRV</td>
</tr>
<tr>
<td>MqttSubscribe</td>
<td>MQTTDRV</td>
</tr>
<tr>
<td>MUL</td>
<td>●</td>
</tr>
<tr>
<td>MVD</td>
<td>●</td>
</tr>
<tr>
<td>NOT_</td>
<td>●</td>
</tr>
<tr>
<td>NSCL</td>
<td>●</td>
</tr>
<tr>
<td>NSSM</td>
<td>MODEL</td>
</tr>
<tr>
<td>OpcUaReadValue</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>OpcUaServerValue</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>OpcUaWriteValue</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>OR_</td>
<td>●</td>
</tr>
<tr>
<td>ORHEXD</td>
<td>●</td>
</tr>
<tr>
<td>OROCT</td>
<td>●</td>
</tr>
<tr>
<td>ORQUAD</td>
<td>●</td>
</tr>
<tr>
<td>OSCALL</td>
<td>●</td>
</tr>
<tr>
<td>OUTHEXD</td>
<td>●</td>
</tr>
<tr>
<td>OUTOCT</td>
<td>●</td>
</tr>
<tr>
<td>Outport</td>
<td>●</td>
</tr>
<tr>
<td>OUTQUAD</td>
<td>●</td>
</tr>
<tr>
<td>OUTRHEXD</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>OUTROCT</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>OUTRQUAD</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>OUTRSTD</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>OUTSTD</td>
<td>●</td>
</tr>
<tr>
<td>PARA</td>
<td>●</td>
</tr>
<tr>
<td>PARB</td>
<td>●</td>
</tr>
<tr>
<td>PARE</td>
<td>●</td>
</tr>
<tr>
<td>PARI</td>
<td>●</td>
</tr>
<tr>
<td>PARR</td>
<td>●</td>
</tr>
<tr>
<td>PARS</td>
<td>●</td>
</tr>
<tr>
<td>PGAVR</td>
<td>●</td>
</tr>
<tr>
<td>PGBAT</td>
<td>●</td>
</tr>
<tr>
<td>PGBUS</td>
<td>●</td>
</tr>
<tr>
<td>PGC</td>
<td>●</td>
</tr>
<tr>
<td>PGENG</td>
<td>●</td>
</tr>
<tr>
<td>PGEN</td>
<td>●</td>
</tr>
</tbody>
</table>

*The list continues on the next page...*
### APPENDIX B. LICENSING OF INDIVIDUAL FUNCTION BLOCKS

<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGGS</td>
<td></td>
</tr>
<tr>
<td>PGINV</td>
<td></td>
</tr>
<tr>
<td>PGLOAD</td>
<td></td>
</tr>
<tr>
<td>PGMAINS</td>
<td></td>
</tr>
<tr>
<td>PGSENS</td>
<td></td>
</tr>
<tr>
<td>PGSIM</td>
<td></td>
</tr>
<tr>
<td>PGSLAR</td>
<td></td>
</tr>
<tr>
<td>PGWIND</td>
<td></td>
</tr>
<tr>
<td>PIDAT</td>
<td>AUTOTUNING</td>
</tr>
<tr>
<td>PIDE</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>PIDGS</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>PIDMA</td>
<td>AUTOTUNING</td>
</tr>
<tr>
<td>PIDU</td>
<td>•</td>
</tr>
<tr>
<td>PIDUI</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>PJROCT</td>
<td>•</td>
</tr>
<tr>
<td>PJSEXGCT</td>
<td>•</td>
</tr>
<tr>
<td>PJSEXGCT</td>
<td>•</td>
</tr>
<tr>
<td>PJSOCT</td>
<td>•</td>
</tr>
<tr>
<td>PCL</td>
<td></td>
</tr>
<tr>
<td>POUT</td>
<td>•</td>
</tr>
<tr>
<td>PRBS</td>
<td>•</td>
</tr>
<tr>
<td>PRGM</td>
<td>•</td>
</tr>
<tr>
<td>PROJECT</td>
<td>•</td>
</tr>
<tr>
<td>PSMPC</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>PWM</td>
<td></td>
</tr>
<tr>
<td>PYTHON</td>
<td>REXLANG</td>
</tr>
<tr>
<td>QFC</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>QFD</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>QP_GASES</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>QTASK</td>
<td></td>
</tr>
<tr>
<td>RDC</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>REC</td>
<td>•</td>
</tr>
<tr>
<td>REGEXP</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>REL</td>
<td>•</td>
</tr>
<tr>
<td>REPLACE</td>
<td>•</td>
</tr>
<tr>
<td>REXLANG</td>
<td>REXLANG</td>
</tr>
<tr>
<td>RLIM</td>
<td>•</td>
</tr>
<tr>
<td>RLY</td>
<td></td>
</tr>
</tbody>
</table>

*The list continues on the next page...*
<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM_AxesGroup</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>RM_Axis</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>RM_AxisOut</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>RM_AxisSpline</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>RM_DirectTorque</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>RM_DirectVelocity</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>RM_DriveMode</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>RM_Feed</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>RM_Gcode</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>RM_GroupTrack</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>RM_HomeOffset</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>RM_Track</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>S_AND</td>
<td></td>
</tr>
<tr>
<td>S_BC</td>
<td></td>
</tr>
<tr>
<td>S_CMP</td>
<td></td>
</tr>
<tr>
<td>S_CTS</td>
<td></td>
</tr>
<tr>
<td>S_LB</td>
<td></td>
</tr>
<tr>
<td>S_NUT</td>
<td></td>
</tr>
<tr>
<td>S_OR</td>
<td></td>
</tr>
<tr>
<td>S_PULS</td>
<td></td>
</tr>
<tr>
<td>S_PV</td>
<td></td>
</tr>
<tr>
<td>S_RS</td>
<td></td>
</tr>
<tr>
<td>S_SEL</td>
<td></td>
</tr>
<tr>
<td>S_SELVAL</td>
<td></td>
</tr>
<tr>
<td>S_SR</td>
<td></td>
</tr>
<tr>
<td>S_SUMC</td>
<td></td>
</tr>
<tr>
<td>S_TDE</td>
<td></td>
</tr>
<tr>
<td>S_TDR</td>
<td></td>
</tr>
<tr>
<td>S_TLATCH</td>
<td></td>
</tr>
<tr>
<td>S.VALB</td>
<td></td>
</tr>
<tr>
<td>S_VALC</td>
<td></td>
</tr>
<tr>
<td>S1OF2</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>SAI</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>SAT</td>
<td></td>
</tr>
<tr>
<td>SC2FA</td>
<td>AUTOTUNING</td>
</tr>
</tbody>
</table>

The list continues on the next page...
## Function block name | Licensing group
--- | ---
SCU | STANDARD
SCUV | STANDARD
SEL | STANDARD
SELHEXD | STANDARD
SELOCT | STANDARD
SELQUAD | STANDARD
SELSOCT | STANDARD
SELU | STANDARD
SETPA | STANDARD
SETPB | STANDARD
SETPI | STANDARD
SETPR | STANDARD
SETPS | STANDARD
SG | STANDARD
SGI | STANDARD
SGSLP | ADVANCED
SHFTOCT | STANDARD
SHLD | STANDARD
SILO | STANDARD
SILGS | STANDARD
SINT | STANDARD
SLEEP | STANDARD
SMHCC | ADVANCED
SMHCCA | AUTOTUNING
SMTP | ADVANCED
SOPDT | STANDARD
SPIKE | ADVANCED
SQR | STANDARD
SQRTH | STANDARD
SR | STANDARD
SRTF | ADVANCED
SSW | STANDARD
STEAM | STANDARD
STOR | STANDARD
SUB | STANDARD
SubSystem | STANDARD
SWR | STANDARD
SWU | STANDARD
SWVMR | STANDARD

The list continues on the next page...
<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STANDARD</td>
</tr>
<tr>
<td>TASK</td>
<td>●</td>
</tr>
<tr>
<td>TIME</td>
<td>●</td>
</tr>
<tr>
<td>TIMER</td>
<td>●</td>
</tr>
<tr>
<td>TIODRV</td>
<td>●</td>
</tr>
<tr>
<td>TRND</td>
<td>●</td>
</tr>
<tr>
<td>TRNDV</td>
<td>●</td>
</tr>
<tr>
<td>TSE</td>
<td>●</td>
</tr>
<tr>
<td>UTOI</td>
<td>●</td>
</tr>
<tr>
<td>VDEL</td>
<td>●</td>
</tr>
<tr>
<td>VIN</td>
<td></td>
</tr>
<tr>
<td>VOUT</td>
<td></td>
</tr>
<tr>
<td>VTOR</td>
<td>●</td>
</tr>
<tr>
<td>WSM</td>
<td></td>
</tr>
<tr>
<td>WCH</td>
<td>●</td>
</tr>
<tr>
<td>WWW</td>
<td>●</td>
</tr>
<tr>
<td>ZV4IS</td>
<td></td>
</tr>
</tbody>
</table>
Appendix C

Error codes of the REXYGEN system

Success codes

- 0 .......... Success
- -1 ....... False
- -2 ....... First value is greater
- -3 ....... Second value is greater
- -4 ....... Parameter changed
- -5 ....... Success, no server transaction done
- -6 ....... Value too big
- -7 ....... Value too small
- -8 ....... Operation in progress
- -9 ....... REXYGEN I/O driver warning
- -10 ..... No more archive items
- -11 ..... Object is array
- -12 ..... Closed
- -13 ..... End of file
- -14 ..... Parameter may be incorrect

General failure codes

- -100 .... Not enough memory
- -101 .... Assertion failure
- -102 .... Timeout
- -103 .... General input variable error
- -104 .... Invalid configuration version
- -105 .... Not implemented
- -106 .... Invalid parameter
- -107 .... COM/OLE error
- -108 .... REXYGEN Module error - some driver or block is not installed or licensed
-109 . . . REXYGEN I/O driver error
-110 . . . Task creation error
-111 . . . Operating system call error
-112 . . . Invalid operating system version
-113 . . . Access denied by operating system
-114 . . . Block period has not been set
-115 . . . Initialization failed
-116 . . . REXYGEN configuration is being changed
-117 . . . Invalid target device
-118 . . . Access denied by REXYGEN security mechanism
-119 . . . Block or object is not installed or licensed
-120 . . . Checksum mismatch
-121 . . . Object already exists
-122 . . . Object doesn’t exist
-123 . . . System user doesn’t belong to any REXYGEN group
-124 . . . Password mismatch
-125 . . . Bad user name or password
-126 . . . Target device is not compatible
-127 . . . Resource is locked by another module and can not be used
-128 . . . String is not valid in UTF8 codepage
-129 . . . Start of executive not allowed
-130 . . . Some resource reach limit
-133 . . . Block execution halted due to runtime error

Class registration, symbol and validation error codes

-200 . . . Class not registered
-201 . . . Class already registered
-202 . . . Not enough space for registry
-203 . . . Registry index out of range
-204 . . . Invalid context
-205 . . . Invalid identifier
-206 . . . Invalid input flag
-207 . . . Invalid input mask
-208 . . . Invalid object type
-209 . . . Invalid variable type
-210 . . . Invalid object workspace
-211 . . . Symbol not found
-212 . . . Symbol is ambiguous
-213 . . . Range check error
-214 . . . Not enough search space
-215 . . . Write to read-only variable denied
-216 . . . Data not ready
-217 . . . Value out of range
-218 . . . Input connection error
-219     Loop of type UNKNOWN detected
-220     REXLANG compilation error

Stream and file system codes
-300     Stream overflow
-301     Stream underflow
-302     Stream send error
-303     Stream receive error
-304     Stream download error
-305     Stream upload error
-306     File creation error
-307     File open error
-308     File close error
-309     File read error
-310     File write error
-311     Invalid format
-312     Unable to compress files
-313     Unable to extract files

Communication errors
-400     Network communication failure
-401     Communication already initialized
-402     Communication finished successfully
-403     Communication closed unexpectedly
-404     Unknown command
-405     Unexpected command
-406     Communication closed unexpectedly, probably "Too many clients"
-407     Communication timeout
-408     Target device not found
-409     Link failed
-410     REXGEN configuration has been changed
-411     REXGEN executive is being terminated
-412     REXGEN executive was terminated
-413     Connection refused
-414     Target device is unreachable
-415     Unable to resolve target in DNS
-416     Error reading from socket
-417     Error writing to socket
-418     Invalid operation on socket
-419     Reserved for socket 1
-420     Reserved for socket 2
-421     Reserved for socket 3
-422     Reserved for socket 4
-423     Reserved for socket 5
-424 . . . Unable to create SSL context
-425 . . . Unable to load certificate
-426 . . . SSL handshake error
-427 . . . Certificate verification error
-428 . . . Reserved for SSL 2
-429 . . . Reserved for SSL 3
-430 . . . Reserved for SSL 4
-431 . . . Reserved for SSL 5
-432 . . . Relay rejected
-433 . . . STARTTLS rejected
-434 . . . Authentication method rejected
-435 . . . Authentication failed
-436 . . . Send operation failed
-437 . . . Receive operation failed
-438 . . . Communication command failed
-439 . . . Receiving buffer too small
-440 . . . Sending buffer too small
-441 . . . Invalid header
-442 . . . HTTP server responded with error
-443 . . . HTTP server responded with redirect
-444 . . . Operation would block
-445 . . . Invalid operation
-446 . . . Communication closed
-447 . . . Connection cancelled

**Numerical error codes**

-500 . . . General numeric error
-501 . . . Division by zero
-502 . . . Numeric stack overflow
-503 . . . Invalid numeric instruction
-504 . . . Invalid numeric address
-505 . . . Invalid numeric type
-506 . . . Not initialized numeric value
-507 . . . Numeric argument overflow/underflow
-508 . . . Numeric range check error
-509 . . . Invalid subvector/submatrix range
-510 . . . Numeric value too close to zero

**Archive system codes**

-600 . . . Archive seek underflow
-601 . . . Archive semaphore fatal error
-602 . . . Archive cleared
-603 . . . Archive reconstructed from saved vars
-604 . . . Archive reconstructed from normal vars
-605 . . .  Archive check sum err error
-606 . . .  Archive integrity error
-607 . . .  Archive sizes changed
-608 . . .  Maximum size of disk archive file exceeded

Motion control codes

-700 . . .  MC - Invalid parameter
-701 . . .  MC - Out of range
-702 . . .  MC - Position not reachable
-703 . . .  MC - Invalid axis state
-704 . . .  MC - Torque limit exceeded
-705 . . .  MC - Time limit exceeded
-706 . . .  MC - Distance limit exceeded
-707 . . .  MC - Step change in position or velocity
-708 . . .  MC - Base axis error or invalid state
-709 . . .  MC - Stopped by HALT input
-710 . . .  MC - Stopped by POSITION limit
-711 . . .  MC - Stopped by VELOCITY limit
-712 . . .  MC - Stopped by ACCELERATION limit
-713 . . .  MC - Stopped by LIMITSWITCH
-714 . . .  MC - Stopped by position LAG
-715 . . .  MC - Axis disabled during motion
-716 . . .  MC - Transition failed
-717 . . .  MC - Not used
-718 . . .  MC - Not used
-719 . . .  MC - Not used
-720 . . .  MC - General failure
-721 . . .  MC - Not implemented
-722 . . .  MC - Command is aborted
-723 . . .  MC - Conflict in block and axis periods
-724 . . .  MC - Busy, waiting for activation

Licensing codes

-800 . . .  Unable to identify Ethernet interface
-801 . . .  Unable to identify CPU
-802 . . .  Unable to identify HDD
-803 . . .  Invalid device code
-804 . . .  Invalid licensing key
-805 . . .  Not licensed

Webserver-related errors

-900 . . .  Web request too large
-901 . . .  Web reply too large
-902 . . .  Invalid format
- 903 . . . Invalid parameter

RexVision-related errors
- 1000 . . . Result is not evaluated
- 1001 . . . The searched object / pattern can not be found
- 1002 . . . The search criterion returned more corresponding objects

FMI standard related errors
- 1100 . . . FMI Context allocation failure
- 1101 . . . Invalid FMU version
- 1102 . . . FMI XML parsing error
- 1103 . . . FMI Model Exchange kind required
- 1104 . . . FMI Co-Simulation kind required
- 1105 . . . Could not create FMU loading mechanism
- 1106 . . . Instantiation of FMU failed
- 1107 . . . Termination of FMU failed
- 1108 . . . FMU reset failed
- 1109 . . . FMU Experiment setup failed
- 1110 . . . Entering FMU initialization mode failed
- 1111 . . . Exiting FMU initialization mode failed
- 1112 . . . Error getting FMU variable list
- 1113 . . . Error getting FMU real variable
- 1114 . . . Error setting FMU real variable
- 1115 . . . Error getting FMU integer variable
- 1116 . . . Error setting FMU integer variable
- 1117 . . . Error getting FMU boolean variable
- 1118 . . . Error setting FMU boolean variable
- 1119 . . . Doing a FMU simulation step failed
- 1120 . . . FMU has too many inputs
- 1121 . . . FMU has too many outputs
- 1122 . . . FMU has too many parameters
Bibliography


Index

REXYGEN Compiler
  compiler, 30
REXYGEN Compiler compiler, 22
TODO
  SRTF DLOG, 36

ABS, 65, 655
absolute
  position sensor, 101
absolute value, 65
ABSROT, 101, 655
ACD, 276, 655
ADD, 66, 67, 655
ADDHEXD, 67, 655
addition, 66
  extended, 74
  integer, 83
ADDOCT, 66, 67, 98, 655
ADDUQUAD, 67, 655
AFLUSH, 282, 655
alarm
  Boolean value, 269
  numerical value, 271
ALB, 269, 655
ALBI, 269, 655
algebraic loop, 30
ALN, 271, 655
ALNI, 271, 655
analog input
  safety, 129
AND, 232, 233, 655
ANDHEXD, 233, 655
ANDOCT, 232, 233, 655
ANDQUAD, 233, 655
ANLS, 150, 655
application
  of the REXYGEN System, 22
ARC, 20, 23, 270, 272, 277, 280, 282, 283, 655
architecture
  open, 31
archiv, 268
archive, 20
  back-up memory, 268
  configuration, 20
  disk, 268
  RAM memory, 268
archiving
  delta criterion, 276
ARLY, 163, 655
ARS, 274, 655
ASW, 103, 656
ATMT, 14, 234, 242, 304, 314, 318, 656
automaton
  finite-state, 234, 242
average
  moving, 105
AVG, 105, 656
AVS, 14, 106, 656
band
  frequency transmission, 107
band-pass filter, 107
bandwidth, 120
BDHEXD, 237, 242, 656
BDOCT, 237, 242, 656
Bessel filter, 120
binary number
  transformation, 250
binary sequence
  generator, 152, 154
BINS, 152, 154, 656
BIS, 152, 154, 156, 656
BISR, 656
BITOP, 238, 656
block
description, 15
description format, 15
execution, 35
inputs, 15
mathematic, 13
matrix, 14
modeling, 14
outputs, 15
parameters, 15
symbol, 15
vector, 14
blocks
analog signal processing, 14
data archiving, 14
generators, 14
input-output, 13
logic control, 14
parameter-related, 14
regulation, 14
special, 15
BMHEXD, 239, 242, 656
BMQCT, 239, 242, 656
Boolean complementation, 251
BPF, 107, 656
Butterworth filter, 120
CanItem, 640, 641, 656
CanRecy, 640, 641, 656
CanSend, 643, 656
CDELSSM, 326, 656
circuit
flip-flop Reset-Set, 254
flip-flop Set-Reset, 255
CMP, 108, 656
CNA, 344, 656
CNB, 68, 656
CNDR, 109, 656
CNE, 69, 656
CNI, 70, 656
CNR, 71, 656
cns, 286, 656
coefficient
relative damping, 107, 120
comparator, 108
compatibility
REXYGEN and Simulink, 30
compensator
lead, 170
lead-lag, 171
nonlinearity, 109
simple nonlinearity, 122
compiler
REXYGEN Compiler, 22, 30
compression, 276
CONCAT, 287, 656
COND, 656
conditioner
nonlinear, 109
configuration
REXYGEN System, 22
archives, 22
computation task, 22
input-output drivers, 22
modules, 22
constant
Boolean, 68
integer, 70
logic, 68
real, 71
control
motion, 14, 15
sequential, 234, 242
control unit
manual, 172
controller
fuzzy logic, 164
PID, 187
PID with gain scheduling, 179
PID with input-defined parameters, 190
PID with relay autotuner, 174
PID with static error, 177
step, 211, 214
with frequency autotuner, 204
correction
INDEX
INDEX

real to integer, 95
COUNT, 17, 240, 656
counter
    controlled, 240
CSSM, 329, 656
data
    remote connection, 443
data storing, 278, 281
data types, 16
DATE, 260, 261, 656
DATETIME, 260, 261, 263, 656
DDELSSM, 331, 656
dead time, 335, 339
DEL, 111, 656
delay
    transport, 112
    variable, 143
    with initialization, 111
DELM, 112, 656
delta criterion, 276
demultiplexer
    bitwise, 237
denominator, 75
DER, 113, 656
derivation, 113, 118
detection
    edge, 245
deviation
    standard, 115
DFIR, 656
DIF, 72, 656
difference, 72
Display, 44, 656
DIV, 73, 656
division
    extended, 75
    integer, 88, 89
    remainder, 89
    two signals, 73
driver
    REXYGEN System, 27
    .rio file extension, 27
    configuration data, 27
    input-output, 13
    input/output, 27
    input/output with tasks, 40
    user manual, 29
drivers
    REXYGEN system, 13
dSSM, 333, 657
EAS, 74, 657
EATNT, 242, 657
dge detection, 245
EDGE, 135, 245, 657
EXF, 657
element
    three state, 230
END, 75, 657
EPC, 37, 434, 657
EQ, 246, 657
error
    fatal, 33
EVAR, 115, 657
EXEC, 20, 22, 27–29, 31, 33, 34, 38–41, 657
executive
    configuration, 13, 19
    real-time, 22
    RexCore program, 13
external program, 434
feedback loop, 30
filter
    band-pass, 107
    Bessel, 120
    Butterworth, 120
    low-pass, 120
    moving average, 105
    nonlinear, 139
    spike, 139
filtering, 113, 118
    digital, 33
FIND, 288, 657
finite-state machine, 234, 242
first order system, 339
FLCU, 14, 164, 657
flip-flop circuit
Reset-Set, 254
Set-Reset, 255
FNX, 76, 657
FNXY, 78, 657
FOPDT, 339
FOPDT, 335, 657
frequency transmission band, 107
FRID, 166, 657
From, 45, 48, 49, 657
function
  operating system, 37
  single variable, 76
  two variables, 78
fuzzy logic, 164
GAIN, 80, 657
gain, 80
generator
  binary sequence, 152, 154
  piecewise linear function, 150
  signal, 159
  time function, 193
GETPA, 302, 657
GETPB, 304, 657
GETPI, 304, 456, 457, 657
GETPR, 304, 318, 657
GETPS, 306, 657
Goto, 45–47, 49, 657
GotoTagVisibility, 48, 49, 657
GRADS, 81, 657
HMI, 24, 657
HTTP, 437, 657
HTTP2, 439, 657
hysteresis, 108
I3PM, 168, 657
IADD, 83, 657
identification
  three parameter model, 168
IDIV, 88, 657
IMOD, 89, 657
IMUL, 86, 657
INFO, 26, 657
INHEXD, 54, 657

INDEX
INDC, 54, 657
Imports, 50, 52, 657
INQUAD, 54, 657
INSTD, 45, 54, 55, 658
INTE, 116, 138, 658
integer
  division, 88
integer number
  transformation, 250
integer signal
  switching, 248
integrator
  controlled, 116
  simple, 138
interpolation
  linear, 90
INTSM, 247, 249, 658
IODRV, 23, 27, 45, 47, 658
IOTASK, 29, 36, 40, 302, 304, 312, 314, 326, 329, 658
ISSW, 248, 658
ISUB, 85, 658
ITOI, 250, 658
ITOS, 289, 658
KDER, 118, 658
LC, 170, 658
least squares method, 113
LEN, 290, 658
LIN, 90, 658
linear
  interpolation, 90
linear function
  generator, 150
LLC, 171, 339, 658
logical sum, 252
loop
  algebraic, 30
  feedback, 30
low-pass filter, 120
LPBRK, 30, 103, 658
LPF, 120, 658
LSM, 113
INDEX

maximum, 121
MB_DA SUM, 345
MB DA XPY, 346
MB DCOPY, 348
MB_DDGT, 350
MB_DGMEM, 352
MB_DGER, 356
MB_DNM2, 358, 396
MB_DROT, 359
MB_DSCAL, 361
MB_DSWAP, 362
MB_DTRMV, 364
MB_DTRSV, 366
MC_AccelerationProfile, 489, 521, 522, 541, 542, 658
MC_AddAxisToGroup, 588, 658
MC_CamIn, 552, 559, 563, 573, 576, 658
MC_CamOut, 552, 556, 658
MC_CombineAxes, 560, 658
MC_GearIn, 563, 566, 571, 573, 576, 658
MC_GearInPos, 566, 658
MC_GearOut, 571, 658
MC_GroupContinue, 608, 609, 658
MC_GroupDisable, 591, 658
MC_GroupEnable, 590, 658
MC_GroupHalt, 603, 658
MC_GroupInterrupt, 608, 609, 658
MC_GroupReadActualAcceleration, 599, 658
MC_GroupReadActualPosition, 597, 658
MC_GroupReadActualVelocity, 598, 658
MC_GroupReadError, 611, 658
MC_GroupReadStatus, 610, 658
MC_GroupReset, 612, 658
MC_GroupSetOverride, 637, 658
MC_GroupSetPosition, 595, 658
MC_GroupStop, 600, 658
MC_Halt, 493, 658
MC_HaltSuperimposed, 494, 659
MC_Home, 495, 526, 659
MC_MoveAbsolute, 497, 510, 534, 549, 567, 659
MC_MoveAdditive, 497, 501, 659
MC_MoveCircularAbsolute, 621, 659
MC_MoveCircularRelative, 625, 659
MC_MoveContinuousAbsolute, 510, 659
MC_MoveContinuousRelative, 513, 659
MC_MoveDirectAbsolute, 629, 659
MC_MoveDirectRelative, 632, 659
MC_MoveLinearAbsolute, 613, 659
MC_MoveLinearRelative, 617, 659
MC_MovePath, 635, 659
MC_MovePath_PH, 659
MC_MoveRelative, 497, 504, 507, 513, 659
MC_MoveSuperimposed, 497, 507, 573, 576, 659
MC_MoveVelocity, 483, 517, 549, 659
MC_PhasingAbsolute, 573, 659
MC_PhasingRelative, 576, 659
MC_PositionProfile, 489, 490, 521, 541, 542, 549, 558, 659
MC_Power, 525, 659
MC_ReadActualPosition, 526, 659
MC_ReadAxisError, 527, 659
MC_ReadBoolParameter, 528, 659
MC_ReadCartesianTransform, 594, 659
MC_ReadParameter, 529, 659
MC_ReadStatus, 531, 659
MC_Reset, 533, 612, 659
MC_SetCartesianTransform, 592, 659
MC_SetOverride, 534, 659
MC_SetPosition, 495
MC_Stop, 493, 536, 659
MC_TorqueControl, 538, 659
MC_UngroupAllAxes, 589, 659
MC_VelocityProfile, 489, 490, 521, 522, 541, 659
MC_WriteBoolParameter, 545, 659
MC_WriteParameter, 546, 659
MCP_AccelerationProfile, 489, 659
MCP_CamIn, 552, 659
MCP_CamTableSelect, 552, 553, 558, 659
MCP_CombineAxes, 560, 660
MCP_GearIn, 563, 660
MCP_GearInPos, 566, 660
MCP_GroupHalt, 660
MCP_GroupInterrupt, 608, 660
MCP_GroupSetOverride, 660
MCP_GroupSetPosition, 595, 660
MCP_GroupStop, 660
MCP_Halt, 493, 660
MCP_HaltSuperimposed, 494, 660
MCP_Home, 495, 660
MCP_MoveAbsolute, 497, 660
MCP_MoveAdditive, 501, 660
MCP_MoveCircularAbsolute, 660
MCP_MoveCircularRelative, 660
MCP_MoveContinuousAbsolute, 510, 660
MCP_MoveContinuousRelative, 513, 660
MCP_MoveDirectAbsolute, 660
MCP_MoveDirectRelative, 660
MCP_MoveLinearAbsolute, 660
MCP_MoveLinearRelative, 660
MCP_MovePath, 660
MCP_MovePath_PH, 660
MCP_MoveRelative, 504, 660
MCP_MoveSuperimposed, 507, 660
MCP_MoveVelocity, 517, 660
MCP_PhasingAbsolute, 573, 660
MCP_PhasingRelative, 576, 660
MCP_PositionProfile, 521, 660
MCP_SetCartesianTransform, 660
MCP_SetKinTransform_Arm, 660
MCP_SetOverride, 534, 660
MCP_Stop, 536, 660
MCP_TorqueControl, 538, 660
MCP_VelocityProfile, 541, 660
MCU, 172, 229, 660
MDL, 336, 337, 660
MDLI, 337, 660
mean value, 115
MID, 291, 660
minimum, 121
MINMAX, 121, 661
ML_DGEBAK, 372
ML_DGEBAL, 374
ML_DGEBRD, 376
ML_DGECON, 378
ML_DGEE, 380
ML_DGEEV, 382
ML_DGEH, 384
ML_DGELQ, 386
ML_DGELSD, 388
ML_DGEQRF, 390
ML_DGESDD, 392
ML_DLACPY, 394
ML_DLANG.E, 358, 396
ML_DLASET, 398
ML_DTRSYL, 400
model
  first order, 335
  FOPDT, 339
  process, 336, 337
  second order, 339
  state space
    continuous, 329
    continuous with time delay, 326
    discrete, 333
    discrete with time delay, 331
  modulation
    pulse width, 199
  MODULE, 23, 27, 31, 661
  module, 31
    extending the REXYGEN System, 31
    extension, 27
  motion control, 14, 15, 106
  moving average, 105
  MP, 156, 661
  MPC, 195
  MqttPublish, 476, 661
  MqttSubscribe, 478, 661
  MUL, 91, 661
  multiplexer
    bitwise, 239
  multiplication
    by a constant, 80
    extended, 75
    two signals, 91
  MVD, 338, 661
  MX_AT, 402
  MX_ATSET, 403
  MX_CNADD, 404
  MX_CMUL, 405
  MX_CTDPA, 406
  MX_DIM, 408
INDEX 681

MX_DIMSET, 409
MX_DSAGET, 411
MX_DSAREF, 413
MX_DSASET, 415
MX_DTNSP, 417
MX_DTNSQ, 419
MX_FILL, 420
MX_MAT, 421
MX_RAND, 422
MX_REFCOPY, 424
MX_SLFS, 425
MX_VEC, 427
MX_WRITE, 428

negation, 251

nonlinear transformation
    simple, 122
NOT_, 251, 661
NSCL, 122, 661
NSSM, 661

OPC server, 447
OpcUsaReadValue, 646, 661
OpcUsaServerValue, 648, 661
OpcUsaWriteValue, 650, 661
operating system, 37

operation
    binary, 94
    bitwise, 238

operator
    relational, 94

optimization
    gradient based, 81
OR_, 252, 253, 661

order
    driver execution, 27
    driver initialization, 27
    module initialization, 31
    of task execution, 38
    of task initialization, 38
ORHEXD, 253, 661
ORCT, 252, 253, 661
ORQUAD, 253, 661
OSCALL, 37, 436, 661

OUTHEXD, 55, 57, 661
OUTOCT, 55, 57, 641, 643, 661
Outport, 50, 53, 661
output
    pulse, 192
    three state, 230
output saturation, 202
OUTQUAD, 55, 57, 641, 643, 661
OUTRHEXD, 57, 661
OUTROCT, 57, 661
OUTRQUAD, 57, 661
OUTRSTD, 58, 661
OUTSTD, 47, 54, 55, 58, 661

overhead
    control system core, 22

PARA, 307, 661
parameter
    tick, 22
    input-defined, 309
    remote, 302, 304, 312, 314
    remote acquisition, 302, 304
PARB, 309, 661
PARC, 308, 661
PARI, 309, 661
PARR, 137, 309, 661
PARS, 311, 661

path
    full, 35

period
    of quick task execution, 33
    of task execution, 38
PGAVR, 661
PGBAT, 661
PGBUS, 661
PGCB, 661
PGENG, 661
PGGEN, 661
PGGS, 662
PGINV, 662
PGLOAD, 662
PGMAINS, 662
PGSENS, 662
PGSG, 662
PGSIM, 662
PGSOLAR, 662
PGWIND, 662
PID
   autotuning, 181
   controller, 187
   with gain scheduling, 179
   with input-defined parameters, 190
   with moment autotuner, 181
   with relay autotuner, 174
   with static error, 177
PIDAT, 14, 174, 662
PIDE, 177, 662
PIDGS, 14, 179, 662
PIDMA, 14, 181, 446, 662
PIDU, 174, 177, 179, 181, 187, 190, 229, 446, 484, 662
PIDUI, 190, 662
PJROCT, 292, 662
PJSEXDOCT, 662
PJSO OCT, 293, 662
POL, 92, 662
polynomial
   evaluation, 92
position sensor
   absolute, 101
POUT, 192, 662
PRES, 157, 662
prediction, 113
predictive control, 195
PRGM, 193, 662
priority
   dependency on the operating system, 22
   logic, 22
   logical, 27, 33
   of tasks, 38
process
   model, 336
   model with variable parameters, 337
product
   Boolean, 232, 233
   logical, 232, 233
program
   REXYGEN Studio, 27
   REXYGEN Diagnostics, 20, 27
   REXYGEN Diagnostics, Enable checkbox, 35
   REXYGEN Diagnostics, Halt/Run button, 35
   REXYGEN Diagnostics, Reset button, 35
   external, 434
   programmable block, 448
   programmable block in Python, 468
   programme
      weekly, 264
   programmer, 193
   PROJECT, 32, 662
   project
      main file, 22, 27
   protocol
      UDP/IP, 443
   PSMPC, 14, 195, 662
   pulses, 192
   pulse
      manually generated, 156
   pulse counting
      bidirectional, 240
   pulse output, 192
   pulse width modulation, 199
   PWM, 180, 186, 189, 191, 199, 218, 222, 223, 662
   PYTHON, 468, 662
   QFC, 59, 60, 455, 662
   QFD, 57–60, 455, 662
   QP_DATES, 662
   QTASK, 22, 23, 29, 33, 36, 38, 326, 329, 439, 441, 662
   quotient, 73
      integer, 88
   rate limiter, 125
   rate monotonic scheduling, 23
   RDC, 15, 443, 662
   RDFT, 123
   real-time
INDEX

executive, 19, 22
REC, 93, 662
reciprocal value, 93
REGEXP, 294, 662
REL, 94, 662
relative damping coefficient, 107, 120
relay
  advance, 163
  with hysteresis, 201
remote
data connection, 443
parameter, 302, 304
REPLACE, 297, 662
REXLANG, 15, 448, 662
RLIM, 125, 662
RLY, 163, 201, 662
RM_AxesGroup, 582, 588, 611, 663
RM_Axis, 431, 483, 484, 495, 525–529, 536,
  547, 548, 588, 663
RM_AxisOut, 547, 548, 663
RM_AxisSpline, 548, 663
RM_DirectTorque, 663
RM_DirectVelocity, 663
RM_DriveMode, 663
RM_Feed, 583, 663
RM_Gcode, 586, 663
RM_GroupTrack, 663
RM_HomeOffset, 663
RM_Track, 549, 663
root
  square, 97
RS, 254, 663
RT0I, 95, 663
RTOS, 298, 663
RTUV, 430, 434, 435, 663
S1OF2, 126, 663
S_AND, 663
S_BC, 663
S_CMP, 663
S_CTS, 663
S_LB, 663
S_NOT, 663
S_OR, 663
S_PULS, 663
S_PV, 663
S_RS, 663
S_SEL, 663
S_SELVAL, 663
S_SR, 663
S_SUMC, 663
S_TDE, 663
S_TDR, 663
S_TLATCH, 663
S_VALB, 663
S_VALC, 663
safety analog input, 129
safety selector, 126
SAI, 126, 128, 129, 663
sample and hold, 137
SAT, 202, 663
SC2FA, 204, 663
schedule
  weekly, 264
SCU, 180, 186, 189, 191, 211, 664
SCUV, 180, 186, 187, 189, 191, 214, 664
SEL, 132, 664
selector
  active controller, 217
  analog signal, 126, 132, 133
  safety, 126
  with ramp, 142
SELHEXD, 133, 664
SELOCT, 133, 664
SELQUAD, 132, 133, 664
SELSOCT, 299, 664
SELV, 217, 431, 664
sensor
  absolute position, 101
sequence
  pseudo-random binary, 157
  sequential control, 234, 242
SETPA, 312, 664
SETPB, 314, 664
SETPI, 314, 457, 664
setpoint, 193
SETPR, 314, 318, 664
SETPS, 316, 664
SG, 159, 446, 664
SGI, 159, 664
SGSLP, 317, 321, 664
SHIFTCT, 135, 664
SHLD, 137, 309, 664
signal generator, 159
SILO, 319, 321, 664
SILOS, 323, 664
simulation
  parameters, 34
  real-time, 34
Simulink, 34
SINT, 116, 138, 664
SLEEP, 34, 664
SMHCC, 218, 664
SMHCCA, 222, 664
SMTP, 441, 664
SOPDT, 339, 664
SPIKE, 129–131, 139, 664
SQR, 96, 664
SQRTE, 97, 664
square root, 97
square value, 96
SR, 255, 664
SRTF, 35, 664
SSW, 141, 431, 446, 664
stack
  size, 27
standard deviation, 115
starting unit, 106
state machine, 234, 242
state space model, 329, 333
  with time delay, 326, 331
STEAM, 664
step controller
  with position feedback, 211
  with velocity output, 214
STOR, 300, 664
SUB, 67, 98, 664
SubSystem, 46, 48, 52, 54, 55, 664
subsystem
  archiving, 267
  execution, 35
subtraction
  extended, 74
  two signals, 98
sum, 66
  Boolean, 253
  logical, 252, 253
switch, 132, 133
  integer signals, 248
  simple, 141
  unit, 229
  with automatic selection of input, 103
SWR, 142, 431, 664
SWU, 217, 229, 664
SWMR, 431, 664
system
  first order, 335
  second order, 339
TASK, 22, 23, 29, 33, 36, 38, 326, 329, 665
task
  driver-triggered, 29
  execution, 35
  execution period, 38
  priority, 38
  quick, 33
  quick, execution period, 33
  standard, 38
TIME, 261, 263, 665
time delay, 112, 335, 339
  variable, 143
timer
  system, 29
  weekly, 264
TIMER, 256, 665
TIODRV, 23, 29, 40, 665
trajectory
  time-optimal, 106
transformation
  binary number, 250
  integer number, 250
transport delay, 112
trend
  recording, 278, 281
TRND, 17, 278, 281, 446, 665
TRNDV, 281, 665
INDEX

TSE, 211, 214, 230, 665

type
  input, 16
  output, 16
  parameter, 16

types
  of variables, 16

user programmable block, 448
user programmable block in Python, 468
UTOI, 665

value
  default, 16
  maximal, 16
  mean, 115
  minimal, 16
  reciprocal, 93
  substitute, 73, 75, 76, 78, 88, 89, 93, 97

valve
  motor driven, 338
  servo, 338

variance, 115

VDEL, 143, 665

VIN, 57, 58, 60, 61, 455, 665

VOUT, 59, 62, 455, 665

VTOR, 123, 432, 435, 665

WASM, 665

weekly
  schedule, 264

WSCH, 264, 665

WWW, 42, 665

ZV4IS, 144, 665