

Function Blocks of REXYGEN

Reference manual

REX Controls s.r.o.

Version 3.0.2

2024-09-16

Plzeň (Pilsen), Czech Republic

Contents

1	Introduction	17
1.1	How to use this manual	17
1.2	The function block description format	23
1.3	Conventions for variables, blocks and subsystems naming	24
1.4	Signal Quality Corresponding with OPC	25
2	EXEC – Real-time executive configuration	27
	ALARMS – REXYGEN alarms list	29
	ARC – REXYGEN archive	32
	EXEC – Real-time executive	34
	HMI – HMI configuration	36
	INFO – Additional project information	38
	IODRV – REXYGEN input/output driver	39
	IOTASK – REXYGEN driver-triggered task	41
	LPBRK – Loop break	42
	MODULE – REXYGEN extension module	43
	OSCALL – Operating system calls	44
	PROJECT – Additional project settings	45
	QTASK – REXYGEN quick task	46
	SLEEP – Timing in Simulink	47
	SRTF – Set run-time flags	48
	STATELOAD – Load multiple block states and parameters	50
	STATESAVE – Save multiple block states and parameters	52
	SYSEVENT – Read system log	54
	SYSLOG – Write system log	56
	TASK – REXYGEN standard task	57
	TIODRV – REXYGEN input/output driver with tasks	59
	WWW – Internal webserver content	61
3	INOUT – Input and output blocks	63
	Display – Numeric display of input values	65
	FromFile – From File	67
	FromWorkspace – From Workspace	68

GotoTagVisibility – Visibility of the signal source	69
INCONN – Block for remote value acquirement	70
INQUAD, INOCT, INHEXD – Multi-input blocks	71
From, INSTD – Signal connection or input	73
IOASYNC – Asynchronous reading and writing	75
OUTCONN – Block for remote value setting	76
OUTQUAD, OUTOCT, OUTHEXD – Multi-output blocks	77
OUTRQUAD, OUTROCT, OUTRHEXD – Multi-output blocks with verification	79
OUTRSTD – Output block with verification	80
Goto, OUTSTD – Signal source or output	81
Inport, Outport – Input and output port	83
QFC – Quality flags coding	85
QFD – Quality flags decoding	86
SubSystem – Subsystem block	87
ToFile – To File	89
ToWorkspace – To Workspace	90
VIN – Validation of the input signal	91
VOU – Validation of the output signal	92
4 MATH – Math blocks	93
ABS – Absolute value	95
ADD – Addition of two signals	96
ADDQUAD, ADDOCT, ADDHEXD – Multi-input addition	97
CNB – Boolean (logic) constant	98
CNE – Enumeration constant	99
CNI – Integer constant	100
CNR – Real constant	101
DIF – Difference	102
DIV – Division of two signals	103
EAS – Extended addition and subtraction	104
EMD – Extended multiplication and division	105
FNX – Evaluation of single-variable function	106
FNXY – Evaluation of two-variables function	109
GAIN – Multiplication by a constant	111
GRADS – Gradient search optimization	112
IADD – Integer addition	114
IDIV – Integer division	116
IMOD – Remainder after integer division	117
IMUL – Integer multiplication	118
ISUB – Integer subtraction	120
LIN – Linear interpolation	122
MUL – Multiplication of two signals	123
NANINF – Block for checking NaN and Inf values	124
POL – Polynomial evaluation	126

REC – Reciprocal value	127
REL – Relational operator	128
RTOI – Real to integer number conversion	129
SQR – Square value	131
SQRT – Square root	132
SUB – Subtraction of two signals	133
UTOI – Unsigned to signed integer number conversion	134
5 ANALOG – Analog signal processing	135
ABSR0T – Absolute rotation (multiturn extension of the position sensor)	137
ASW – Switch with automatic selection of input	139
AVG – Moving average filter	141
AVS – Motion control unit	142
AVSI – Smooth trajectory interpolation	143
BPF – Band-pass filter	146
CMP – Comparator with hysteresis	147
CNDR – Nonlinear conditioner	148
DEL – Delay with initialization	150
DELM – Time delay	151
DER – Derivation, filtering and prediction from the last n+1 samples	152
EVAR – Moving mean value and standard deviation	154
INTE – Controlled integrator	155
KDER – Derivation and filtering of the input signal	157
LPF – Low-pass filter	159
MINMAX – Running minimum and maximum	161
NSCL – Nonlinear scaling factor	162
OSD – One Step Delay	163
RDFT – Running discrete Fourier transform	164
RLIM – Rate limiter	166
S10F2 – One of two analog signals selector	167
SAI – Safety analog input	170
SEL – Selector switch for analog signals	173
SELQUAD, SELOCT, SELHEXD – Selector switch for analog signals	174
SHIFTOCT – Data shift register	176
SHLD – Sample and hold	178
SINT – Simple integrator	179
SPIKE – Spike filter	180
SSW – Simple switch	182
SWR – Selector with ramp	183
VDEL – Variable time delay	184
ZV4IS – Zero vibration input shaper	185

6 GEN – Signal generators	189
ANLS – Controlled generator of piecewise linear function	190
BINS – Controlled binary sequence generator	192
BIS – Binary sequence generator	194
BISR – Binary sequence generator with reset	196
MP – Manual pulse generator	198
PRBS – Pseudo-random binary sequence generator	199
SG, SGI – Signal generators	201
7 REG – Function blocks for control	203
ARLY – Advance relay	205
FIWR – Frequency Identification With Reconstructor	206
FLCU – Fuzzy logic controller unit	209
FRID – Frequency response identification	211
I3PM – Identification of a three parameter model	215
LC – Lead compensator	217
LLC – Lead-lag compensator	218
LPI – Loop performance index	219
MCU – Manual control unit	221
PIDAT – PID controller with relay autotuner	223
PIDE – PID controller with defined static error	226
PIDGS – PID controller with gain scheduling	228
PIDMA – PID controller with moment autotuner	230
PIDU – PID controller unit	237
PIDUI – PID controller unit with variable parameters	241
POUT – Pulse output	243
PRGM – Setpoint programmer	244
PSMPC – Pulse-step model predictive controller	246
PWM – Pulse width modulation	250
RLY – Relay with hysteresis	252
SAT – Saturation with variable limits	253
SC2FA – State controller for 2nd order system with frequency autotuner	255
SCU – Step controller with position feedback	262
SCUV – Step controller unit with velocity input	265
SELU – Controller selector unit	268
SMHCC – Sliding mode heating/cooling controller	270
SMHCCA – Sliding mode heating/cooling controller with autotuner	274
SWU – Switch unit	281
TSE – Three-state element	282
8 LOGIC – Logic control	283
AND – Logical product of two signals	284
ANDQUAD, ANDOCT, ANDHEXD – Multi-input logical product	285
ATMT – Finite-state automaton	286

BDOCT, BDHEXD – Bitwise demultiplexers	288
BITOP – Bitwise operation	289
BMOCT, BMHEXD – Bitwise multiplexers	290
COUNT – Controlled counter	291
EATMT – Extended finite-state automaton	293
EDGE – Falling/rising edge detection in a binary signal	296
EQ – Equivalence two signals	297
INTSM – Integer number bit shift and mask	298
ISSW – Simple switch for integer signals	299
ITOI – Transformation of integer and binary numbers	300
NOT – Boolean complementation	301
OR – Logical sum of two signals	302
ORQUAD, OROCT, ORHEXD – Multi-input logical sum	303
RS – Reset-set flip-flop circuit	304
SR – Set-reset flip-flop circuit	305
TIMER – Multipurpose timer	306
9 TIME – Blocks for handling time	309
DATE – Current date	310
DATETIME – Get, set and convert time	311
TC – Timer control and status	314
TIME – Current time	316
TS – Current timestamp	317
TS2NS – Timestamp difference in nanoseconds	319
WSCH – Week scheduler	320
10 ARC – Data archiving	323
ACD – Archive compression using Delta criterion	325
ACLEAR – Forced archive purge	327
AFLUSH – Forced archive flushing	328
ALB, ALBI – Alarms for Boolean value	329
ALM, ALMI – Alarm store value	331
ALN, ALNI – Alarms for numerical value	332
ARS – Archive store value	335
TRND – Real-time trend recording	337
TRNDV – Real-time trend recording (for vector signals)	340
11 STRING – Blocks for string operations	343
CNS – String constant	344
CONCAT – Concat string by pattern	345
FIND – Find substring	346
ITOS – Integer number to string conversion	347
LEN – String length	348
MID – Substring extraction	349

PJROCT – Parse JSON string (real output)	350
PJSEXOCT – Parse JSON string (string output)	352
PJSOCT – Parse JSON string (string output)	354
REEXP – Regular expression parser	356
REPLACE – Replace substring	358
RTOS – Real number to string conversion	359
SELSOCT – Selector switch for string signals	360
STOR – String to real number conversion	362
TRIM – Remove leading and trailing whitechar	363
12 PARAM – Blocks for parameter handling	365
GETPA – Block for remote array parameter acquirement	366
GETPB, GETPI, GETPR – Blocks for remote parameter acquirement	368
GETPS – Block for remote string parameter acquirement	370
GETPX – Block for remote parameter acquirement	371
PARA – Block with input-defined array parameter	373
PARE – Block with input-defined enumeration parameter	375
PARB, PARI, PARR – Blocks with input-defined parameter	376
PARS – Block with input-defined string parameter	378
PARX – Block with input-defined parameter	379
SETPA – Block for remote array parameter setting	381
SETPB, SETPI, SETPR – Blocks for remote parameter setting	383
SETPS – Block for remote string parameter setting	385
SETPX – Block for remote parameter setting	386
SGSLP – Set, get, save and load parameters	388
SILO – Save input value, load output value	392
SILOS – Save input string, load output string	394
13 MODEL – Dynamic systems simulation	397
CDELSSM – Continuous state space model with time delay	399
CSSM – Continuous state space model	402
DDELSSM – Discrete state space model with time delay	405
DFIR – Discrete finite input response filter	407
DSSM – Discrete state space model	408
EKF – Extended (nonlinear) Kalman filter	411
FOPDT – First order plus dead-time model	414
IPEN2, IPEN3 – N-link inverted pendulum on cart - Physical parameters	415
IPEN2pu, IPEN3pu – N-link inverted pendulum on cart - Dynamic parameters	418
MDL – Process model	421
MDLI – Process model with input-defined parameters	422
MVD – Motorized valve drive	423
NSSM – Nonlinear State-Space Model	424
NUREACT – Model of nuclear reactor	427
QCOPT – Model of quadrucopter	428

SGEN – Synchronous generator model	430
SGENTX – Synchronous generator model	432
SOPDT – Second order plus dead-time model	434
STMGEN – Model of steam generator	436
STURB – Steam turbine model	438
14 MATRIX – Blocks for matrix and vector operations	441
CNA – Array (vector/matrix) constant	444
MB_DASUM – Sum of the absolute values	446
MB_DAXPY – Performs $y := a*x + y$ for vectors x, y	448
MB_DCOPY – Copies vector x to vector y	450
MB_DDOT – Dot product of two vectors	452
MB_DGEMM – Performs $C := \alpha*op(A)*op(B) + \beta*C$, where $op(X) = X$ or $op(X) = X^T$	454
MB_DGEMV – Performs $y := \alpha*A*x + \beta*y$ or $y := \alpha*A^T*x + \beta*y$	456
MB_DGER – Performs $A := \alpha*x*y^T + A$	458
MB_DNRM2 – Euclidean norm of a vector	460
MB_DRROT – Plain rotation of a vector	462
MB_DSCAL – Scales a vector by a constant	464
MB_DSWAP – Interchanges two vectors	466
MB_DTRMM – Performs $B := \alpha*op(A)*B$ or $B := \alpha*B*op(A)$, where $op(X) = X$ or $op(X) = X^T$ for triangular matrix A	468
MB_DTRMV – Performs $x := A*x$ or $x := A^T*x$ for triangular matrix A	470
MB_DTRSV – Solves one of the system of equations $A*x = b$ or $A^T*x = b$ for triangular matrix A	472
ML_DGEBAK – Backward transformation to ML_DGEBAL of left or right eigenvectors	474
ML_DGEBAL – Balancing of a general real matrix	476
ML_DGEBRD – Reduces a general real matrix to bidiagonal form by an orthogonal transformation	478
ML_DGECON – Estimates the reciprocal of the condition number of a general real matrix	480
ML_DGEES – Computes the eigenvalues, the Schur form, and, optionally, the matrix of Schur vectors	483
ML_DGEEV – Computes the eigenvalues and, optionally, the left and/or right eigenvectors	485
ML_DGEHRD – Reduces a real general matrix A to upper Hessenberg form	487
ML_DGELQF – Computes an LQ factorization of a real M-by-N matrix A	489
ML_DGELSD – Computes the minimum-norm solution to a real linear least squares problem	491
ML_DGEQRF – Computes an QR factorization of a real M-by-N matrix A	493
ML_DGESDD – Computes the singular value decomposition (SVD) of a real M-by-N matrix A	495

ML_DLACPY – Copies all or part of one matrix to another matrix	497
ML_DLANGE – Computes one of the matrix norms of a general matrix	499
ML_DLASET – Initalizes the off-diagonal elements and the diagonal elements of a matrix to given values	501
ML_DTRSYL – Solves the real Sylvester matrix equation for quasi-triangular matrices A and B	503
MX_AT – Get Matrix/Vector element	505
MX_ATSET – Set Matrix/Vector element	506
MX_CNADD – Add scalar to each Matrix/Vector element	507
MX_CNMUL – Multiply a Matrix/Vector by a scalar	508
MX_CTODPA – Discretizes continuous model given by (A,B) to (Ad,Bd) using Pade approximations	509
MX_DIM – Matrix/Vector dimensions	511
MX_DIMSET – Set Matrix/Vector dimensions	512
MX_DSAGET – Set subarray of A into B	514
MX_DSAREF – Set reference to subarray of A into B	516
MX_DSASET – Set A into subarray of B	518
MX_DTRNSP – General matrix transposition: $B := \alpha A^T$	520
MX_DTRNSQ – Square matrix in-place transposition: $A := \alpha A^T$	522
MX_FILL – Fill real matrix or vector	524
MX_MAT – Matrix data storage block	525
MX_RAND – Randomly generated matrix or vector	526
MX_REFCOPY – Copies input references of matrices A and B to their output references	528
MX_SLFS – Save or load a Matrix/Vector into file or string	529
MX_VEC – Vector data storage block	532
MX_WRITE – Write a Matrix/Vector to the console/system log	533
RTOV – Vector multiplexer	535
SWVMR – Vector/matrix/reference signal switch	537
VTOR – Vector demultiplexer	538
15 OPTIM – Optimization blocks	539
QP_MPC2QP – Conversion of MPC problem to quadratic programming	540
QP_OASES – Quadratic programming using active set method	547
QP_UPDATE – Update matrices/vectors of quadratic programming	552
SOLNP – Nonlinear optimization solver	557
16 SPEC – Special blocks	561
EPC – External program call	562
HTTP – Block for generating HTTP GET or POST requests (obsolete)	565
HTTP2 – Block for generating HTTP requests	567
RDC – Remote data connection	569
REXLANG – User programmable block	573
SMTP – Block for sending e-mail alerts via SMTP	593

STEAM – Steam and water properties	595
UART – UART communication block	600
17 LANG – Language blocks	603
LUA, LUAQUAD, LUAOCT, LUAHEXD – User programmable blocks in Lua	604
PYTHON – User programmable block in Python	608
18 DSP – Digital Signal Processing blocks	613
BSFIFO – Binary Structure - Queueing serialize and deserialize	614
BSGET, BSGETOCT – Binary Structure - Get a single value of given type . .	616
BSGETV, BSGETOCTV – Binary Structure - Get matrix (all values of same given type)	618
BSSET, BSSETOCT – Binary Structure - Set a single value of given type . .	620
BSSETV, BSSETOCTV – Binary Structure - Set matrix (all values of same given type)	621
FFT – Fast Fourier Transform	622
MOSS – Motion smart sensor	624
PCI – PCI Bus Memory Access	626
PSD – Power Spectral Density	627
19 MQTTDrv – Communication via MQTT protocol	633
MqttPublish – Publish MQTT message	634
MqttSubscribe – Subscribe to MQTT topic	636
20 MC_SINGLE – Motion control - single axis blocks	639
MCP_AccelerationProfile – * Acceleration profile	641
MCP_Halt – * Stopping a movement (interruptible)	643
MCP_HaltSuperimposed – * Stopping a movement (superimposed and in- terruptible)	644
MCP_Home – * Homing	645
MCP_MoveAbsolute – * Move to position (absolute coordinate)	647
MCP_MoveAdditive – * Move to position (relative to previous motion) . .	649
MCP_MoveContinuousAbsolute – * Move to position (absolute coordinate)	651
MCP_MoveContinuousRelative – * Move to position (relative to previous motion)	653
MCP_MoveRelative – * Move to position (relative to execution point) . .	655
MCP_MoveSuperimposed – * Superimposed move	657
MCP_MoveVelocity – * Move with constant velocity	658
MCP_PositionProfile – * Position profile	660
MCP_SetOverride – * Set override factors	662
MCP_Stop – * Stopping a movement	663
MCP_TorqueControl – * Torque/force control	664
MCP_VelocityProfile – * Velocity profile	666

MC_AccelerationProfile, MCP_AccelerationProfile – Acceleration profile	668
MC_Halt, MCP_Halt – Stopping a movement (interruptible)	672
MC_HaltSuperimposed, MCP_HaltSuperimposed – Stopping a movement (superimposed and interruptible)	673
MC_Home, MCP_Home – Homing	674
MC_MoveAbsolute, MCP_MoveAbsolute – Move to position (absolute coordinate)	676
MC_MoveAdditive, MCP_MoveAdditive – Move to position (relative to previous motion)	680
MC_MoveContinuousAbsolute, MCP_MoveContinuousAbsolute – Move to position (absolute coordinate)	683
MC_MoveContinuousRelative, MCP_MoveContinuousRelative – Move to position (relative to previous motion)	686
MC_MoveRelative, MCP_MoveRelative – Move to position (relative to execution point)	690
MC_MoveSuperimposed, MCP_MoveSuperimposed – Superimposed move	693
MC_MoveVelocity, MCP_MoveVelocity – Move with constant velocity	696
MC_PositionProfile, MCP_PositionProfile – Position profile	700
MC_Power – Axis activation (power on/off)	704
MC_ReadActualPosition – Read actual position	705
MC_ReadAxisError – Read axis error	706
MC_ReadBoolParameter – Read axis parameter (bool)	707
MC_ReadParameter – Read axis parameter	708
MC_ReadStatus – Read axis status	710
MC_Reset – Reset axis errors	712
MC_SetOverride, MCP_SetOverride – Set override factors	713
MC_Stop, MCP_Stop – Stopping a movement	715
MC_TorqueControl, MCP_TorqueControl – Torque/force control	717
MC_VelocityProfile, MCP_VelocityProfile – Velocity profile	720
MC_WriteBoolParameter – Write axis parameter (bool)	724
MC_WriteParameter – Write axis parameter	725
RM_Axis – Motion control axis	726
RM_AxisOut – Axis output	733
RM_AxisSpline – Commanded values interpolation	734
RM_HomeOffset – * Homing by setting offset	739
RM_Track – Tracking and inching	740
21 MC_MULTI – Motion control - multi axis blocks	743
MCP_CamIn – * Engage the cam	745
MCP_CamTableSelect – Cam definition	747
MCP_CombineAxes – * Combine the motion of 2 axes into a third axis	749
MCP_GearIn – * Engage the master/slave velocity ratio	751
MCP_GearInPos – * Engage the master/slave velocity ratio in defined position	753

MCP_PhasingAbsolute – * Create phase shift (absolute coordinate)	755
MCP_PhasingRelative – * Create phase shift (relative to previous motion)	757
MC_CamIn, MCP_CamIn – Engage the cam	759
MC_CamOut – Disengage the cam	763
MC_CombineAxes, MCP_CombineAxes – Combine the motion of 2 axes into a third axis	765
MC_GearIn, MCP_GearIn – Engage the master/slave velocity ratio	768
MC_GearInPos, MCP_GearInPos – Engage the master/slave velocity ratio in defined position	771
MC_GearOut – Disengage the master/slave velocity ratio	776
MC_PhasingAbsolute, MCP_PhasingAbsolute – Phase shift in synchro- nized motion (absolute coordinates)	778
MC_PhasingRelative, MCP_PhasingRelative – Phase shift in synchro- nized motion (relative coordinates)	781
22 MC_COORD – Motion control - coordinated movement blocks	783
MCP_GroupHalt – * Stopping a group movement (interruptible)	787
MCP_GroupInterrupt – * Read a group interrupt	788
MCP_GroupSetOverride – * Set group override factors	789
MCP_GroupSetPosition – * Sets the position of all axes in a group	790
MCP_GroupStop – * Stopping a group movement	791
MCP_MoveCircularAbsolute – * Circular move to position (absolute co- ordinates)	792
MCP_MoveCircularRelative – * Circular move to position (relative to execution point)	794
MCP_MoveDirectAbsolute – * Direct move to position (absolute coordinates)	796
MCP_MoveDirectRelative – * Direct move to position (relative to execu- tion point)	798
MCP_MoveLinearAbsolute – * Linear move to position (absolute coordinates)	800
MCP_MoveLinearRelative – * Linear move to position (relative to execu- tion point)	802
MCP_MovePath – * General spatial trajectory generation	804
MCP_MovePath_PH – * General spatial trajectory generation PH	806
MCP_SetCartesianTransform – * Sets Cartesian transformation	808
MCP_SetKinTransform_Arm – * Kinematic transformation robot ARM	810
MCP_SetKinTransform_UR – * Kinematic transformation for UR robot	812
MC_AddAxisToGroup – Adds one axis to a group	814
MC_GroupContinue – Continuation of interrupted movement	815
MC_GroupDisable – Changes the state of a group to GroupDisabled	816
MC_GroupEnable – Changes the state of a group to GroupEnable	817
MC_GroupHalt – Stopping a group movement (interruptible)	818
MC_GroupInterrupt, MCP_GroupInterrupt – Read a group interrupt	823
MC_GroupReadActualAcceleration – Read actual acceleration in the se- lected coordinate system	824

MC_GroupReadActualPosition – Read actual position in the selected coordinate system	825
MC_GroupReadActualVelocity – Read actual velocity in the selected coordinate system	826
MC_GroupReadError – Read a group error	827
MC_GroupReadStatus – Read a group status	828
MC_GroupReset – Reset axes errors	829
MC_GroupSetOverride – Set group override factors	830
MC_GroupSetPosition, MCP_GroupSetPosition – Sets the position of all axes in a group	831
MC_GroupStop – Stopping a group movement	833
MC_MoveCircularAbsolute – Circular move to position (absolute coordinates)	836
MC_MoveCircularRelative – Circular move to position (relative to execution point)	840
MC_MoveDirectAbsolute – Direct move to position (absolute coordinates)	844
MC_MoveDirectRelative – Direct move to position (relative to execution point)	847
MC_MoveLinearAbsolute – Linear move to position (absolute coordinates)	850
MC_MoveLinearRelative – Linear move to position (relative to execution point)	854
MC_MovePath – General spatial trajectory generation	858
MC_MovePath_PH – * General spatial trajectory generation PH	860
MC_ReadCartesianTransform – Reads the parameter of the cartesian transformation	862
MC_SetCartesianTransform – Sets Cartesian transformation	863
MC_UngroupAllAxes – Removes all axes from the group	865
RM_AxesGroup – Axes group for coordinated motion control	866
RM_Feed – * MC Feeder ???	869
RM_Gcode – * CNC motion control	870
RM_GroupTrack – T	872
23 CanDrv – Communication via CAN bus	875
CanItem – Secondary received CAN message	876
CanRecv – Receive CAN message	878
CanSend – Send CAN message	880
24 OpcUaDrv – Communication using OPC UA	881
OpcUaReadValue – Read value from OPC UA Server	882
OpcUaServerValue – Expose value as an OPC UA Node	884
OpcUaWriteValue – Write value to OPC UA Server	886

25 UNIPI – Communication blocks for Unipi	889
IM201CNT – Iris IM201CNT, 4 digital counters	891
IM201DI – Iris IM201DI, 4 digital inputs	892
IM203DO – Iris IM203DO, 8 digital outputs	893
IM203PWM – Iris IM203PWM, 8 PWM outputs	894
IM204CNT – Iris IM204CNT, 16 digital counters	895
IM204DI – Iris IM204DI, 16 digital inputs	897
IM205DO – Iris IM205DO, 16 digital outputs	898
IM205PWM – Iris IM205PWM, 16 PWM outputs	899
IM301CNT – Iris IM301CNT, 2 digital counters	901
IM301DI – Iris IM301DI, 2 digital inputs	902
IM301DO – Iris IM301DO, 2 digital outputs	903
IM502AO – Iris IM502AO, 4 analog outputs	904
IM503AI – Iris IM503AI, 8 analog inputs	905
IM504RI – Iris IM504RI, 8 resistance or temperature inputs	907
IM506AI – Iris IM506AI, 2 analog inputs	909
IM506AO – Iris IM506AO, 1 analog output	911
IRIS_MODULE – Iris - Module description info	912
UNIPI_CHANNEL – Iris module or Patron section info	913
UNIPI_PRODUCT – Product description info	915
UNIPI_S1AI – Patron section 1, analog input	916
UNIPI_S1AOR – Patron section 1, analog output or resistance input	917
UNIPI_S1CNT – Patron section 1, counters	918
UNIPI_S1DI – Patron section 1, digital inputs	919
UNIPI_S1DO – Patron section 1, digital outputs	920
UNIPI_S1LED – Patron section 1, LED outputs	921
UNIPI_S1PWM – Patron section 1, PWM outputs	922
UNIPI_S2AI – Patron section 2, analog inputs	923
UNIPI_S2AO – Patron section 2, analog outputs	924
UNIPI_S2CNT – Patron section 2, counters	925
UNIPI_S2DI – Patron section 2, digital inputs	926
UNIPI_S2RO – Patron section 2, relay outputs	927
 A Licensing options	 929
 B Licensing of individual function blocks	 931
 C Error codes of the REXYGEN system	 945
 D Special signals of the REXYGEN system	 951
 Bibliography	 953

Index**955**

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* (sub-libraries). 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 as follows:

1 Introduction

This introductory chapter familiarizes readers with the content and ordering of the manual. A convention used for individual function block descriptions is presented.

2 EXEC – Real-time executive configuration

The EXEC library is essential for setting up the real-time executive in the REXYGEN system and includes key blocks like **EXEC**, **TASK**, **QTASK**, and **HMI**. These blocks are fundamental for managing task execution, determining process priorities, and interacting with user interfaces, significantly contributing to the efficiency and controllability of applications within the REXYGEN ecosystem.

3 INOUT – Input and output blocks

The INOUT library serves as a crucial interface in the REXYGEN system, enabling smooth interaction with input/output drivers. It is designed for efficient simultaneous signal processing, essential for fast control tasks. This library simplifies the connection between control algorithms and hardware, ensuring minimal latency. Additionally, it provides advanced features, such as virtual linking (flags) of signals for increased clarity of diagrams and flexibility of subsystems.

4 MATH – Math blocks

The MATH library offers a comprehensive collection of mathematical operations and functions. It includes basic arithmetic blocks like [ADD](#), [SUB](#), [MUL](#), and [DIV](#) for standard calculations, and more specialized blocks such as [ABS](#) for absolute values, [SQRT](#) for square roots, and [SQR](#) for squaring. Advanced functionalities are provided by blocks like [LIN](#) for linear transformations, [POL](#) for polynomial evaluations, and [FNX](#), [FNXY](#) for customizable mathematical functions. The library also features integer-specific operations through blocks like [IADD](#), [IMUL](#), [IDIV](#), and [IMOD](#).

5 ANALOG – Analog signal processing

Library presents a versatile range of functional blocks, designed for control and signal processing applications. It includes blocks like [ASW](#), [AVG](#), [BPF](#), and [DEL](#), which provide functionalities from signal manipulation and averaging to filtering and complex conditional operations, catering to a broad spectrum of system requirements and scenarios.

6 GEN – Signal generators

The GEN library is specialized in signal generation. It includes blocks like [ANLS](#) for generating a piecewise linear function of time or binary sequence generators [BINS](#), [BIS](#), [BISR](#). The library also features [MP](#) for manual pulse signal generation, [PRBS](#) for pseudo-random binary sequence generation, and [SG](#) for periodic signals generation. This library provides essential tools for creating and manipulating various signal types.

7 REG – Function blocks for control

The control function blocks form the most extensive sub-library 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 sub-library. The involved controllers include the [PIDGS](#) 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

The LOGIC library encompasses a range of blocks for executing logical and sequential operations. It includes basic Boolean blocks like [AND](#), [OR](#), [NOT](#) for fundamental logical operations, and advanced blocks like [ATMT](#) for finite state machines. Blocks like [COUNT](#) and [TIMER](#) extend functionality to bidirectional pulse counting and time-based operations. Additional elements like [BITOP](#), [BMOCT](#), and [BDOCT](#) offer bitwise operations and multiplexing/demultiplexing capabilities, enhancing the library's versatility in handling combinational and sequential logic control.

9 TIME – Blocks for handling time

The TIME library is specialized for time-based operations and scheduling in REXY-

GEN system. It includes blocks like [DATE](#), [TIME](#) and [DATETIME](#) for handling date and datetime, providing essential tools for working with temporal data. The library features [TC](#) for internal timer control. Additionally, [WSCH](#) is used for scheduling, enabling efficient management of time-dependent tasks. This library is particularly valuable for systems requiring precise time management and scheduling capabilities.

10 **ARC – Data archiving**

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.

11 **STRING – Blocks for string operations**

The STRING library is dedicated to string manipulation and analysis in REXYGEN system. It includes blocks like [CONCAT](#) for concatenating strings, [FIND](#) for searching within strings, and [REPLACE](#) for replacing string segments. The library offers [LEN](#) and [MID](#) for determining string length and extracting substrings, respectively. Advanced pattern matching is provided by [REGEXP](#). Conversion blocks such as [ITOS](#), [STOR](#) and [RTOS](#) convert integers and real numbers to strings, while a simple [CNS](#) block defines a string constant. Additionally, the library features blocks like [PJROCT](#) for JSON parsing. This collection of blocks is essential for handling and processing string data in various applications.

12 **PARAM – Blocks for parameter handling**

The PARAM library is designed for parameter management and signal processing in the REXYGEN system. It includes blocks like [PARR](#) and its variants for defining and modifying various types of parameters. Blocks for getting parameters of other blocks like [GETPA](#) and [GETPS](#). Conversely, [SETPA](#), [SETPR](#) and [SETPS](#) are used to dynamically set parameter values of other blocks. Additionally, the library contains [SILO](#) and [SILOS](#) for exporting and importing values from a file. This library is crucial for systems requiring dynamic parameter manipulation and the ability to read/save values to a file.

13 **MODEL – Dynamic systems simulation**

The MODEL library is centered around system modeling and simulation. It includes blocks like [CSSM](#) and [DSSM](#) for continuous and discrete state-space models, and [DFIR](#) for digital finite impulse response filters. The library offers [EKF](#) for Extended Kalman Filter implementations, and [FOPDT](#), [SOPDT](#) for first and second order process time delay models. Additionally, it provides [FMUCS](#) and [FMUINFO](#) for interfacing with Functional Mock-up Units, and [MDL](#), [MDLI](#) for generic model interfaces. Advanced functionalities are covered by blocks like [CDELSSM](#), [DDELSSM](#) for continuous and discrete state space models of a linear system with time delay, and [MVD](#) for model variable delays, catering to a wide range of modeling requirements in REXYGEN system.

14 MATRIX – Blocks for matrix and vector operations

The MATRIX library is designed for advanced matrix computations and manipulations. It encompasses a wide range of blocks such as [MB_DGEMM](#), [MB_DTRMM](#), and [MB_DGER](#) for matrix-matrix and matrix-vector operations. The library includes functions for matrix decomposition ([ML_DGEBRD](#), [ML_DGEQRF](#)), eigenvalue problems ([ML_DGEEV](#), [ML_DGEES](#)), and singular value decomposition ([ML_DGESDD](#)). Additionally, it offers utility blocks like [MX_MAT](#), [MX_VEC](#), and [MX_FILL](#) for matrix creation and manipulation, as well as specialized blocks such as [MX_DTRNSP](#) for matrix transposition and [MX_RAND](#) for generating random matrices. This library is essential for complex mathematical operations involving matrices in various applications.

15 OPTIM – Optimization blocks

The OPTIM library is tailored for optimization algorithms and processes. It includes [QCEDPOPT](#) for Quadratic Cost Economic Dispatch Problem optimization, providing advanced tools for handling complex optimization problems. The library also features blocks like [QP_MPC2QP](#) and [QP_OASES](#) for Quadratic Programming, essential in Model Predictive Control (MPC) scenarios. Additionally, [QP_UPDATE](#) is available for updating quadratic program parameters. This library is particularly useful in systems requiring high-level optimization solutions, such as in advanced control and decision-making algorithms.

16 SPEC – Special blocks

The SPEC library encompasses a diverse set of functional blocks designed to integrate a wide range of functionalities into automation, control systems, and communication protocols. From facilitating precise thermodynamic calculations with the [STEAM](#) block to enabling seamless data communication through [UART](#) and [SMTP](#), the library serves as a comprehensive toolkit for engineers and developers. It includes specialized blocks for executing external programs ([EPC](#)), handling web-based requests ([HTTP2](#)). Additionally, it offers unique input-output solutions ([RDC](#)) and a versatile programming environment with [REXLANG](#).

17 LANG – Language blocks

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. For these purposes, the blocks from the LANG library, or the [REXLANG](#) block, can be used.

18 DSP – Digital Signal Processing blocks

The DSP library is tailored for advanced digital signal processing. It includes blocks like [FFT](#) for Fast Fourier Transform operations and [PSD](#) for Power Spectral Density analysis. The library also features [BSFIFO](#), [BSGET](#), [BSGETV](#), [BSSET](#), and [BSSETV](#) for buffer storage and retrieval, enabling efficient data handling in signal processing tasks. In addition, the library contains a [MOSS](#) block - an advanced filter for incremental sensors. This collection of blocks is essential for sophisticated signal analysis and manipulation in digital systems.

19 MQTTDrv – Communication via MQTT protocol

The MQTTDrv library is designed for IoT (Internet of Things) communication using the MQTT (Message Queuing Telemetry Transport) protocol. It consists of two primary blocks: `MqttPublish` and `MqttSubscribe`. The `MqttPublish` block is used for sending messages to an MQTT broker, enabling the publication of data to MQTT topics. Conversely, the `MqttSubscribe` block is designed for subscribing to topics and receiving messages from a broker. This library facilitates efficient and effective data communication in IoT applications, leveraging the lightweight and widely-used MQTT protocol for message exchange.

20 MC_SINGLE – Motion control - single axis blocks

The MC_SINGLE library is designed for motion control in single-axis systems. It features blocks like `MC_MoveAbsolute`, `MC_MoveRelative`, and `MC_MoveVelocity` for precise positioning and speed control. The library includes `MC_Home` for homing operations, and `MC_Power` for controlling the power state of the axis. Advanced functionalities are provided by `MC_AccelerationProfile`, `MC_PositionProfile`, and `MC_VelocityProfile` for customizing motion profiles. It also offers monitoring and parameter adjustment capabilities through `MC_ReadActualPosition`, `MC_ReadAxisError`, `MC_ReadParameter`, and `MC_WriteParameter`. Additionally, the library contains blocks like `MC_Halt`, `MC_Reset`, and `MC_Stop` for emergency and control operations. This library is essential for applications requiring precise and controlled motion in single-axis configurations.

21 MC_MULTI – Motion control - multi axis blocks

The MC_MULTI library is specialized for multi-axis motion control. It includes blocks like `MC_CombineAxes` for synchronizing multiple axes, `MC_GearIn` and `MC_GearOut` for gearing operations, and `MC_PhasingAbsolute`, `MC_PhasingRelative` for precise axis phasing. The library offers `MC_CamIn` and `MC_CamOut` for camming functionalities, allowing complex motion profiles to be followed. Additionally, `MCP_CamTableSelect` provides flexibility in selecting cam tables, and `MC_GearInPos` enables position-based gearing. This library is essential for advanced applications requiring coordinated motion control across multiple axes.

22 MC_COORD – Motion control - coordinated movement blocks

The MC_COORD library is specifically designed for the coordination of multi-axis motion control within complex systems. It encompasses a variety of blocks, including `MC_MoveLinearAbsolute` for executing precise linear movements, complemented by `MC_MoveLinearRelative` for relative linear motion. For the execution of circular motion, the library incorporates `MC_MoveCircularAbsolute` alongside `MC_MoveCircularRelative`, ensuring detailed circular trajectories. In the context of managing group axis control, this library introduces `MC_AddAxisToGroup`, which is further supported by functionalities such as `MC_GroupEnable` for activation, `MC_GroupDisable` for deactivation, and `MC_GroupHalt` for immediate stopping of grouped axes. Furthermore, the library provides `MC_MoveDirectAbsolute` and `MC_MoveDirectRelative`, enabling direct control over axis movements. For navi-

gating through complex paths, `MC_MovePath` is made available. Essential monitoring and control features are facilitated by `MC_GroupReadActualPosition` for positional data, `MC_GroupReadActualVelocity` for velocity insights, `MC_GroupReadError` for error detection, and `MC_GroupReadStatus` for status updates. Additionally, the library integrates `MC_ReadCartesianTransform` and `MC_SetCartesianTransform`, which are vital for Cartesian transformation processes. This collection of functionalities underscores the library's significance in applications that demand the synchronized control of multiple axes, particularly in the realms of robotics and automation systems.

23 CanDrv – Communication via CAN bus

The CanDrv library is dedicated to handling CAN (Controller Area Network) bus communication in REXYGEN system. It features `CanItem` for managing CAN data items, `CanRecv` for receiving messages from the bus, and `CanSend` for sending messages. This library provides essential tools for efficient and reliable communication over CAN networks, facilitating data exchange and control commands between various system components.

24 OpcUaDrv – Communication using OPC UA

The OpcUaDrv library is specialized in interfacing with OPC UA (Open Platform Communications Unified Architecture) servers for industrial automation. The first block – `OpcUaReadValue` is designed for reading data from servers, making it pivotal for data acquisition in automated systems. The `OpcUaWriteValue` block enables writing data to servers, allowing for control and command execution. Additionally, the `OpcUaServerValue` block facilitates the monitoring and management of server values. This library serves as a critical tool for seamless communication and interaction with OPC UA servers, enhancing the capabilities of automation systems.

25 UNIPI – Communication blocks for Unipi

This library is used to control and monitor Unipi devices. It includes blocks for reading and writing digital and analog inputs and outputs, blocks for controlling relays, PWM outputs, LED diodes and reading counters. For reading buses, drivers such as OwsDrv or MbDrv can be used. The blocks work in accordance with the manufacturer's documentation [1], where technical details about individual devices and their inputs and outputs can be found.

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:

<name>	[<i>nam</i>] Detailed description of the input (output, parameter)	<type>
	<name>. Mathematical symbol <i>nam</i> 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.	[⊙<def>] [↓<min>] [↑<max>]

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

Each described variable (input, output or parameter) has a default value <def> in the REXYGEN system, which is preceded by the ⊙ symbol. Also it has upper and lower limits, preceded by the symbols ↓ and ↑ respectively. All these three values are optional (marked by []). If the value ⊙<def> is not listed in the second column, it is equal to zero. If the values of ↓<min> and/or ↑<max> are missing, the limits are given by the minimum and/or maximum of the corresponding type, see table 1.1¹.

¹Precise range of the **Large** data type is -9223372036854775808 to 9223372036854775807.

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

Table 1.1: Types of variables in the REXYGEN system.

1.3 Conventions for variables, blocks and subsystems naming

Several conventions are used to simplify the use of the REXYGEN 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:

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 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 (Open Platform Communications) specification [2]. They can be represented by one byte, whose structure is explained in the table 1.2.

Bit number	7	6	5	4	3	2	1	0
Bit weight	128	64	32	16	8	4	2	1
Bit field	Quality		Substatus				Limits	
	Q	Q	S	S	S	S	L	L
BAD	0	0	S	S	S	S	L	L
UNCERTAIN	0	1	S	S	S	S	L	L
not used in OPC	1	0	S	S	S	S	L	L
GOOD	1	1	S	S	S	S	L	L

Table 1.2: The quality flags structure

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 [2], chapter 6.8. The list of blocks propagating signal quality is given in table 1.3.

The principle of quality propagation between blocks operates as follows: The lowest quality among all *data* inputs to a block is determined and applied to all *data* outputs. Any unconnected input is considered as good quality (**GOOD**). Quality on *control* inputs is not tracked; however, *control* inputs can influence the propagation of quality from *data* inputs. On *status* outputs, the quality is invariably set to good (**GOOD**).

For instance, in a **DEL** block, the input *u* is considered *data* because it carries operational data. The input *R1* is classified as *control* since it controls the block's operation. The output *y* is *data* because it conveys the block's output information. The *RDY* output is *status* as it indicates the operational state of the block.

Except for certain cases (**SAI**, **VIN**, **S10F2**), the quality does not influence the block's algorithm (i.e., the actual values at the outputs). Some blocks may assign a lower quality (**UNCERTAIN**, **BAD**) as a result of their algorithm (e.g., **DEL** prior to buffer filling or **DIV**

ABSROT	ABS	ADDHEXD	ADD	ANDHEXD	AND	ARLY	ASW
ATMT	AVG	AVSI	AVS	BDHEXD	BITOP	BMHEXD	BPF
CDELSSM	CMP	CNA	CNB	CNDR	CNI	CNR	CNS
CONCAT	COUNT	CSSM	DDELSSM	DELM	DEL	DER	DFIR
DIF	DIV	DSSM	EAS	EATMT	EDGE	EMD	EQ
EVAR	FIND	FIWR	FLCU	FNXY	FNX	FOPDT	FRID
GAIN	GETPA	GETPR	GETPS	GETPX	IADD	IDIV	IMOD
IMUL	INTE	INTSM	IOASYNC	ISSW	ISUB	ITOI	ITOS
KDER	LC	LEN	LIN	LLC	LPBRK	LPF	LPI
MCU	MDLI	MDL	MID	MINMAX	MUL	MVD	NANINF
NOT	NSCL	ORHEXD	OR	OSD	PIDAT	PIDE	PIDGS
PIDMA	PIDUI	PIDU	POL	POUT	PSMPC	PWM	RDFT
REC	REL	REPLACE	RLIM	RLY	RS	RTOI	RTOS
RTOV	SAT	SC2FA	SCUV	SCU	SELHEXD	SELSOCT	SELU
SETPA	SETPR	SETPS	SETPX	SHIFTOCT	SHLD	SINT	SMHCCA
SMHCC	SOPDT	SPIKE	SQRT	SQR	SR	SSW	STOR
SUB	SWR	SWU	TIMER	TRIM	TSE	UTOI	VDEL
VIN	VTOR	ZV4IS					

Table 1.3: The list of blocks prapagating signal quality

when dividing by zero).

Chapter 2

EXEC – Real-time executive configuration

Contents

ALARMS – REXYGEN alarms list	29
ARC – REXYGEN archive	32
EXEC – Real-time executive	34
HMI – HMI configuration	36
INFO – Additional project information	38
IODRV – REXYGEN input/output driver	39
IOTASK – REXYGEN driver-triggered task	41
LPBRK – Loop break	42
MODULE – REXYGEN extension module	43
OSCALL – Operating system calls	44
PROJECT – Additional project settings	45
QTASK – REXYGEN quick task	46
SLEEP – Timing in Simulink	47
SRTF – Set run-time flags	48
STATELOAD – Load multiple block states and parameters	50
STATESAVE – Save multiple block states and parameters	52
SYSEVENT – Read system log	54
SYSLOG – Write system log	56
TASK – REXYGEN standard task	57
TIODRV – REXYGEN input/output driver with tasks	59
WWW – Internal webserver content	61

The EXEC library is essential for setting up the real-time executive in the REXYGEN system and includes key blocks like [EXEC](#), [TASK](#), [QTASK](#), and [HMI](#). These blocks are fundamental for managing task execution, determining process priorities, and interacting with user interfaces, significantly contributing to the efficiency and controllability of applications within the REXYGEN ecosystem.

ALARMS – REXYGEN alarms list

Block Symbol

Licence: [STANDARD](#)



Function Description

The **ALARMS** is placed in the *main project file* and allows the user to configure list of alarms. Alarms are activated by the [ALM](#) or [ALMI](#) blocks. Alarms are defined in a **.csv** (Comma separated variable) file. The **afile** parameter contains the file name of the **.csv** file. An alarm could be activated also by the [ALB](#), [ALBI](#), [ALN](#), [ALNI](#) blocks, but these blocks not use definitions in the **ALARMS** block.

The configuration file has the following columns:

id	... Unique alarm reference number. The number is used in the ALM block, in archive records, etc.
level	... The value stored into an archive record (in the level field).
archives	... Bit field – identifies archives for recording events associated with the alarm (alarm starts, ends, acknowledges). E.g. 0 = not stored in the archive, 1 = stored in the 1st archive, 2 = stored in the 2nd archive, 4 = stored in the 3rd archive, 3 = stored in the 1st and 2nd archives, etc.
group	... Reserved for future use, now some number (or bitfield up to 64 bits) to filter alarm's list in HMI.
name	... Name of the alarm; can be used as the alarm identifier, so it should be unique.
description	... A text description for the alarm, allowing for multilingual text formatting and the incorporation of values related to the alarm within the description.

Multilingual support

REXYGEN supports multilingual alarm description. The description field must be in the form:

```
<lang1_ID>:<lang1 text>|<lang2_ID>:<lang2 text>|<lang3_ID>:<lang3 text>
```

Number of languages is not limited, but total size of the field is limited to 32765 bytes (english characters). The lang1 (language 1) is used if the user sets unsupported language. If the user sets an empty language (i.e. ""), the entire text will be displayed, i.e. all languages including formatting marks.

Example: Let's expect the description field in the form: **cz:Přepětí|en:High voltage alarm**. The user will see **High voltage alarm** if the language is set to **en**. The user will

see **Přepětí** if the language is set to **cz**. The user will see **Přepětí** in all other cases (for example if the language is set to **de**, **cze**, **EN**, **en-us**, etc.).

Associated values

The description field can contain special marks that is replaced by values from control algorithm – so-called associated values. The mark has the form:

```
%<value number>[<format>][:<number of characters>[:<precision>]]
```

where the format is one of the following characters:

- b**, **B** ... binary value (string **on** or **off** is shown)
- d**, **D** ... integer number shown as decimal string, the default value for integer types
- x**, **X** ... integer number shown as hexadecimal string
- f**, **F** ... real number in fix point form, the precision of it is a number of digits behind decimal point (if precision is specified)
- e**, **E** ... real number in exponential (scientific) form
- g**, **G** ... the same as **F** or **E** (depends on actual value), the default format for real number types
- s**, **S** ... text string

The default type is used if the format is not specified or if the type of the value is not compatible with the specified format. More characters than it is specified is used if it is necessary to show the correct value.

Format Examples:

- %2** ... value of 2nd variable (e.g. **av2** in the **ALM** or **ALMI** block)
- %1:8:2** ... value of 1st variable (e.g. **av1** in the **ALM** or **ALMI** block), 2 characters behind decimal point, total 8 characters (leading spaces are used if necessary)
- %1e** ... value of 1st variable (e.g. **av1** in the **ALM** or **ALMI** block) in exponential form

The **ALB**, **ALBI** blocks not use associated values. The **ALN** and **ALNI** maps it this way:

- 1 ... value of the **u** input
- 2 ... value of the **h** parameter (input)
- 3 ... value of the **hh** parameter (input)
- 4 ... value of the **l** parameter (input)
- 5 ... value of the **ll** parameter (input)
- 6 ... value of the **tout** parameter (input)

Remarks:

- It is possible to use comma or semicolon as a separator in the **.csv** file. The first row with column names is optional.
- Alarms (lines) in the file must be in the ascending order respect to the **id**.
- The **id** must be unique including other alarming/archiving blocks (**TRND**, **ALB**, **ALN**, ...).

- It is possible to use the internal editor (the **Configure** button in parametric dialog) or external tool. Internal editor generates a correct example if the `.csv` file does not exist.
- The blocks `ALB`, `ALBI`, `ALN` and `ALNI` regard `lvl > 127` as an event, where only its begin (nor end nor acknowledge) is stored into archives. The blocks `ALM`, `ALMI` do not implement this event function.
- Alarm's associated values are stored into alarm's value when alarm is triggered (begin). Later changes of the associated values are not updated in an alarm window in HMI.
- Alarm window in HMI can show also alarm name. It is the name of the block (without block type if it prefixes the block name) that is connected to the alarm.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

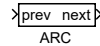
Parameter

<code>afile</code>	Alarms definition CSV filename	String
--------------------	--------------------------------	--------

ARC – REXYGEN 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 **Diagnostics** section of the **REXYGEN Studio** program after successful connecting to the target device. The following options are usually available:

- **1: RAM memory:** The archive is allocated in RAM memory (it is irretrievably lost after restarting **RexCore**).
- **2: Permanent memory:** The archive is allocated in backup memory, e.g. CMOS (remains after restarting **RexCore**).
- **3: Disk:** Archive is saved to disk (remains after restarting **RexCore**).

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.

The frequency of writing values can be influenced by the **period** parameter. For devices using flash memory or SD cards as a disk, it is not suitable to write values too often, therefore it is appropriate to set this parameter to a value in the order of minutes. Furthermore, it is possible to select a suitable source of time stamps with the **timesrc** parameter.

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

`prev` Input for chaining archives Long (I32)

Parameter

<code>atype</code>	Archive type	⊙1	Long (I32)
	1 RAM memory		
	2 Permanent memory		
	3 Disk		
<code>asize</code>	Archive size [bytes]	↓256 ⊙102400	Long (I32)
<code>nmarks</code>	Number of time stamps	↓2 ⊙720	Long (I32)
<code>ldaymax</code>	Maximum size of archive per day [bytes]		Large (I64)
		↓1000 ↑2147480000 ⊙1048576	
<code>period</code>	Period of writing data to disk [s]	⊙60.0	Double (F64)
<code>timesrc</code>	Source of timestamps	⊙1	Long (I32)
	1 CORETIMER		
	2 CORETIMER (precise)		
	3 RTC (UTC)		
	4 RTC (localtime)		
	5 PFC		
	6 TSC		

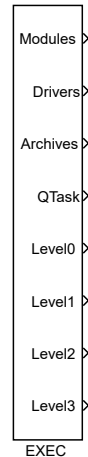
Output

`next` Output for chaining archives Long (I32)

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!

On multi-core CPUs, the parameters `cpu_rt` and `cpu_other` can be used to reserve a core for real-time tasks (e.g. **Drivers** and **Levels**) and a core for other REXYGEN tasks (e.g. diagnostics or web server). After filling the parameter on a given CPU, only the defined REXYGEN tasks will run. Furthermore, the parameters `cpu0` to `cpu3` can be used to assign a core to a given **Level**. The CPUs are numbered starting from 0, where -1 means the default setting. The parameters `cpu0-3` have priority over `cpu_rt` and `cpu_other`.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Parameter

target	target	⊙Generic target device	String
tick	Base tick (period) of the runtime core [s]	⊙0.05	Double (F64)
ntick0.. <code>ntick3</code>	Period of tasks in Level0 (tick*ntick0)	↓1 ⊙10	Long (I32)
pri0.. <code>pri3</code>	Priority of tasks in Level0	↓3 ↑31 ⊙5	Long (I32)
cpu_rt	Default CPU core for real-time tasks (-1=default, 0=core 0, 1=core 1, ...)	↓-1 ↑127 ⊙-1	Long (I32)
cpu_other	Default CPU core for non real-time tasks (-1=default, 0=core 0, 1=core 1, ...)	↓-1 ↑127 ⊙-1	Long (I32)
cpu0.. <code>cpu3</code>	Level0 tasks CPU core (-1=default, 0=core 0, 1=core 1, ...)	↓-1 ↑127 ⊙-1	Long (I32)

Output

Modules	Anchor for chain of modules	Long (I32)
Drivers	Anchor for chain of I/O drivers	Long (I32)
Archives	Anchor for chain of archives	Long (I32)
QTask	Anchor for connecting a quick task	Long (I32)
Level0..<code>Level3</code>	Anchor for chain of Level0 tasks	Long (I32)

HMI – HMI 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 and so it belongs to the **EXEC** category.

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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Parameter

<code>IncludeHMI</code>	Include HMI files in the project	<input type="radio"/> on	Bool
<code>HmiDir</code>	Output folder for HMI files	<input type="radio"/> hmi	String
<code>SourceDir</code>	Source directory	<input type="radio"/> hmisrc	String
<code>GenerateWebWatch</code>	Generate WebWatch HMI source files from MDL files	<input type="radio"/> on	Bool
<code>GenerateRexHMI</code>	Build HMI from SVG and JS files when compiling project	<input type="radio"/> on	Bool
<code>RedirectToHMI</code>	Webserver will automatically redirect to HMI webpage	<input type="radio"/> on	Bool
<code>Compression</code>	Enable data compression		Bool

INFO – Additional project information

Block Symbol

Licence: [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 behaviour.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

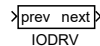
Parameter

Title	Project title	String
Author	Project author	String
Description	Brief description of the project	String
Customer	Information about the customer	String

IODRV – REXYGEN 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, see the [MODULE](#) block for more details.

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 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 **Diagnostics** section of the REXYGEN Studio

program. The `cpu` parameter can be used to specify where the driver thread should run on multi-CPU devices.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>prev</code>	Input for chaining I/O drivers	Long (I32)
-------------------	--------------------------------	------------

Parameter

<code>module</code>	Module name		String
<code>classname</code>	I/O driver class name	⊙DrvClass	String
<code>cfgname</code>	Configuration file name	⊙iodrv.rio	String
<code>factor</code>	Execution factor	↓1 ⊙10	Long (I32)
<code>stack</code>	Stack size [bytes]	↓1024 ⊙10240	Long (I32)
<code>pri</code>	Driver thread logical priority	↓1 ↑31 ⊙3	Long (I32)
<code>cpu</code>	CPU core assigned to driver thread (-1=default, 0=core 0, 1=core 1, ...)	↓-1 ↑127 ⊙-1	Long (I32)

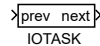
Output

<code>next</code>	Output for chaining I/O drivers	Long (I32)
-------------------	---------------------------------	------------

IOTASK – REXYGEN driver-triggered task

Block Symbol

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 [TIODRV](#) block. The user manual of the given driver provides more details about the possibility and conditions of using the above mentioned approach. **Note:** The parameter `MDLOPEN` is intended for the internal needs of the REXYGEN system and cannot be changed manually.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>prev</code>	Input for chaining I/O tasks	Long (I32)
-------------------	------------------------------	------------

Parameter

<code>factor</code>	Execution factor	⊙1	Long (I32)
<code>stack</code>	Stack size [bytes]	⊙10240	Long (I32)
<code>filename</code>	Corresponding MDL file		String
<code>MDLOPEN</code>	Is the corresponding MDL file open?		Bool

Output

<code>next</code>	Output for chaining I/O tasks	Long (I32)
-------------------	-------------------------------	------------

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 behaviour 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** 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.

Note: Behaviour of the **LPBRK** block has been changed since the version 3.0. The block is not removed by the **REXYGEN Compiler** but is present in the algorithm and clears the quality flag of the **y** output. This change is useful and necessary due to the quality propagation in function blocks. Original behaviour (e.g. the block is removed from the algorithm) can be forced by the **RB = on** parameter. The main function of the block (indication of the feedback signal) remains unchanged in all cases.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	Input signal	Double (F64)
----------	--------------	--------------

Parameter

RB	Remove block from configuration (loopback indicator only)	Bool
-----------	---	------

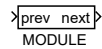
Output

y	Output signal	Double (F64)
----------	---------------	--------------

MODULE – REXYGEN extension module

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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

prev	Input for chaining modules	Long (I32)
-------------	----------------------------	------------

Output

next	Output for chaining modules	Long (I32)
-------------	-----------------------------	------------

OSCALL – Operating system calls

Block Symbol

Licence: [STANDARD](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>TRG</code>	Trigger of the selected action	<code>Bool</code>
------------------	--------------------------------	-------------------

Parameter

<code>action</code>	System function to perform	⊙1 <code>Long (I32)</code>
	1 Reboot system	
	2 System shutdown	
	3 System halt	
	4 Flush disc caches	
	5 Lock system partition	
	6 Unlock system partition	
	7 Disable internal webserver	
	8 Enable internal webserver	
<code>cmd</code>	Reserved for internal use	<code>String</code>

Output

<code>E</code>	Error indicator	<code>Bool</code>
	<code>off</code> ... No error	
	<code>on</code> ... An error occurred	
<code>iE</code>	Error code	<code>Long (I32)</code>

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 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 behaviour.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Parameter

CompileParams	RexComp command-line options		String
SourcesOnTarget	Store source files on target device	 on	Bool
TargetURL	Default URL address of the target device		String
LibraryPath	Path to libraries referenced in the project		String
PreBuild	Command executed (by operating system) before compilation		String
PostBuild	Command executed (by operating system) after compilation		String

QTASK – REXYGEN quick task

Block Symbol

Licence: [STANDARD](#)



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 `.mdl` 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 block's execution time is shown in the **Diagnostics** section of the **REXYGEN Studio** program.

Note: The parameter **MDLOPEN** is intended for the internal needs of the **REXYGEN** system and cannot be changed manually.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

prev	Connection to EXEC block	Long (I32)
-------------	---------------------------------	------------

Parameter

factor	Execution factor	⊙1	Long (I32)
stack	Stack size [bytes]	⊙10240	Long (I32)
filename	Corresponding MDL file		String
MDLOPEN	Is the corresponding MDL file open?		Bool

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).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

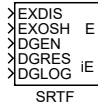
Parameter

ts	Sleep time [s]	⊙0.1 Double (F64)
-----------	----------------	-------------------

SRTF – Set run-time flags

Block Symbol

Licence: [ADVANCED](#)



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 **bname** 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 full path to the block (object) is case sensitive. Individual layers are separated by dots, object names except tasks ([TASK](#), [QTASK](#)) begin with one of the following special characters:

- **^** – computational level (Level), e.g. **^0** for **Level0**
- **&** – input-output driver (I/O Driver), e.g. **&WcnDrv**

The name of the task executed by the I/O controller ([IOTASK](#)) is entered in the form **&<driver_name>.<task_name>**.

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 **Diagnostics** of the **REXYGEN** Studio program.
- **One-shot execution** of the object. If the object execution is disabled by the **EXDIS = on** input or by the **Diagnostics** section of the **REXYGEN** Studio 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 **Diagnostics** section of the corresponding tab (**I/O Driver**, **Level**, **Quick Task**, **Task**, **I/O Task**, **Sequence**) of the **REXYGEN** Studio program.

- **Reset diagnostic data** of the given object by **DGRES = on**. The same flag can be set by the **Reset** button in the **Diagnostics** section of the corresponding tab in the **REXYGEN Studio** 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.

Object type	EXDIS	EXOSH	DGEN	DGRES
I/O Driver	✓	✓	✓	✓
Level	✓	×	✓	✓
Task	✓	✓	✓	✓
Quick Task	✓	✓	✓	✓
I/O Task	✓	✓	✓	✓
Sequence, subsystem	✓	×	✓	✓
Block	✓	×	×	×

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

EXDIS	Disable execution	Bool
EXOSH	One-shot execution	Bool
DGEN	Enable diagnostics	Bool
DGRES	Reset diagnostic data	Bool
DGLOG	Enable verbose diagnostics	Bool

Parameter

bname	Full path to the controlled task or block	String
-------	---	--------

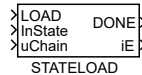
Output

E	Error indicator	Bool
	off ... No error	
	on An error occurred	
iE	Error code	Long (I32)
	0 No error	
	1 Object not found (invalid bname parameter)	
	2 REXYGEN internal error (invalid pointers)	
	3 Flag could not be set (timeout)	

STATELOAD – Load multiple block states and parameters

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **STATELOAD** block reloads values of state and parameters from a file or string. The file is specified by the **filename** parameter and must be in JSON format, usually stored by the [STATESAVE](#) block. It is also possible to read data from the **InState** input, which is a JSON string in the same format as the input file. The **InState** input is used if the **filename** parameter is empty.

All values configured by the parameters **blocks**, **depth**, and **mask** that are stored in the file are loaded. The **blocks** parameter contains relative paths (starting with a dot) to the loaded blocks separated by semicolons. If **blocks** is empty, all blocks of the current subsystem are loaded. If the loaded block is a subsystem, the **depth** parameter specifies how many levels are also loaded:

- 0: current level only,
- n: current level and blocks in subsystems of other **n** levels.

Furthermore, you can use **mask** to specify which objects will be loaded. Each bit of a number means:

- 1: inputs,
- 2: outputs,
- 4: parameters,
- 8: internal states,
- 16: array parameters,
- 32: array states,
- 64: cyclic (trend) buffers,
- 256: metadata (**STATESAVE** only).

If the parameter **Strict** is set to **on**, the block checks if the configured blocks and values match those stored in the file, and the file is refused if there is a mismatch.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>LOAD</code>	Trigger to load the state	<code>Bool</code>
<code>InState</code>	JSON string to load if the filename parameter is empty	<code>String</code>
<code>uChain</code>	Useful for placing the block in the correct execution order	<code>Long (I32)</code>

Parameter

<code>filename</code>	Filename from which to load	<code>String</code>
<code>blocks</code>	List of blocks to load	<code>String</code>
<code>depth</code>	Specifies how many levels are loaded	$\downarrow 0 \uparrow 65535$ <code>Long (I32)</code>
<code>mask</code>	Select which objects are loaded	$\downarrow 0 \uparrow 65535 \oplus 65535$ <code>Long (I32)</code>
<code>LoadOnInit</code>	The file is loaded during the configuration initialization	$\odot \text{on}$ <code>Bool</code>
<code>STRICT</code>	The file is checked against the current configuration	$\odot \text{on}$ <code>Bool</code>

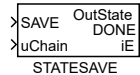
Output

<code>DONE</code>	State has been loaded	<code>Bool</code>
<code>iE</code>	Error code	<code>Error</code>

STATESAVE – Save multiple block states and parameters

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **STATESAVE** block stores the values of states and parameters in a file. The file is specified by the **filename** parameter and is in JSON format, which can usually be reloaded by the **STATELOAD** block. It is also possible to store data in the **OutState** output, which is a JSON string in the same format as the file output. The **OutState** output is used if the **filename** parameter is empty.

All values configured by the parameters **blocks**, **depth**, and **mask** are stored. The **blocks** parameter contains relative paths (starting with a dot) to the stored blocks separated by semicolons. If **blocks** is empty, all blocks of the current subsystem are saved. If the stored block is a subsystem, the **depth** parameter specifies how many levels are also stored:

- 0: current level only,
- n: current level and blocks in subsystems of other **n** levels.

Furthermore, you can use **mask** to specify which objects will be stored. Each bit of a number means:

- 1: inputs,
- 2: outputs,
- 4: parameters,
- 8: internal states,
- 16: array parameters,
- 32: array states,
- 64: cyclic (trend) buffers,
- 256: metadata (**STATESAVE** only).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

SAVE	Trigger to save the state	Bool
uChain	Useful for placing the block in the correct execution order	Long (I32)

Parameter

filename	Filename where to store	String
blocks	List of blocks to store	String
depth	Specifies how many levels are stored	↓0 ↑65535 Long (I32)
mask	Select which objects are saved	↓0 ↑65535 ⊙65535 Long (I32)
SaveOnExit	The file is stored when the configuration is terminated	⊙on Bool

Output

OutState	JSON string where values are stored if the filename parameter is empty	String
DONE	State has been saved	Bool
iE	Error code	Error

SYSEVENT – Read system log

Block Symbol

Licence: [STANDARD](#)



Function Description

This block serves to read records from the system log or archive. The archive to be read is selected by the **arc** parameter. Not all entries are displayed, but only those that pass through the filter. It is possible to filter by item ID (in the case of the system log it is meaningless - currently, all entries have **id=1**), by the level of alarm/event (in the case of the system log, categories are encoded there), and in the case of a text item, also by value.

The filter by ID is set using the **idfrom** and **idto** parameters, which select the interval to be displayed. If both values are the same, only one **id** is displayed, and if **idfrom>idto**, filtering by **id** is turned off and all **ids** are displayed.

The filter by level is set using the **lvlfrom** and **lvlto** parameters, with the same rules as in the previous case applying.

The filter by value applies only to text items (in the system log, these are all entries). An item is displayed only if it contains the text from the **filter** parameter. If the parameter is empty, all items are displayed. This parameter has no effect on other than text items, and they are always displayed (if they meet the settings of other filters).

As long as there are items in the archive that meet the filter, they are displayed so that one item is on the output in each tick (in the order they are stored in the archive) and the **VALID** output is set to **on**. When there are no more items, the outputs have the values corresponding to the last read item, but **VALID** is set to **off**.

The output **sVal** contains the value of the text item (for other types of items, it is empty). The output **iVal** contains the value of the integer item (for other types of items, it is 0). In all cases, all parameters (including the value) are stored in JSON format on the **sEvent** output. To retrieve the required values, the **PJSOCT** block, or possibly the **PJROCT** block, can be used.

Note: If multiple **sysevent** blocks are used, each goes through the respective archive separately. Depending on the set filter, it can happen that a certain item from the archive is output by both blocks, but usually at a different moment.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Parameter

<code>arc</code>	Archive to read (0=system log)	↓0 ↑16	Long (I32)
<code>filter</code>	String that item must contain		String
<code>idfrom</code>	Minimum item ID to show	↓0 ↑65535	Long (I32)
<code>idto</code>	Maximum item ID to show	↓0 ↑65535 ⊙65655	Long (I32)
<code>lvlfrom</code>	Minimum item level to show	↓0 ↑255	Long (I32)
<code>lvlto</code>	Maximum item level to show	↓0 ↑255 ⊙255	Long (I32)

Output

<code>VALID</code>	Output data are valid (actual)	Bool
<code>sEvent</code>	Whole archive item in JSON	String
<code>sVal</code>	Archive item value (string)	String
<code>iVal</code>	Archive item value (integer)	Long (I32)

SYSLOG – Write system log

Block Symbol

Licence: [STANDARD](#)



Function Description

The `SYSLOG` block is intended for writing any messages to the `REXYGEN` system log. It can be used for basic logging of user events. To write, it is necessary to have messages of the given level enabled in the System Logs Configuration (Target -> System Logs Configuration -> Function block messages).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>msg</code>	String writing into system log	String
<code>lvl</code>	Message level	Long (I32)
	0 Error	
	1 Warning	
	2 Information	
	3 Verbose	

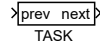
Parameter

<code>RUN</code>	Enable writing message	Bool
------------------	------------------------	------

TASK – REXYGEN standard task

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 **Level0** to **Level3** 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 **Level0** to **Level3** (the [QTASK](#) block precedes **Level0**).

All tasks at the given level *<i>* are executed with the same priority, which is determined by the **pri<i>** parameter of the [EXEC](#) block. The execution period of the task is calculated as a multiple of the **factor** parameter and the base tick of the **ntick<i>*tick** in the [EXEC](#) block.

The time allocated for task execution starts at the **start** tick and ends at the **stop** tick. The **start** and **stop** values can be fixed or left to be controlled by the **RexCore**. For **RexCore** control, the parameters can be filled in as follows:

- **start** = -1: The execution begins as soon as the previous **Task** ends.
- **start** = -2: The execution starts on the next **tick** after the completion of the previous task.
- **stop** = -1: The task execution must finish before the end of **ntick<i>*tick**.
- **stop** = -2: The task execution must finish in the next **tick**.

For fixed execution times, **start** and **stop** should be a non-negative integer.

The REXYGEN Compiler additionally verifies that the **stop** parameter of the preceding task is less than or equal to the **stop** parameter of the succeeding task. This ensures that the allocated time intervals for individual tasks do not overlap. If the timing of individual levels is inappropriate, tasks may be interrupted by tasks and other events with higher priority. In such cases, execution is not aborted but delayed (in contrast to the [QTASK](#) block). The **Diagnostics** section of the REXYGEN Studio program assesses whether the execution delay is occasional or permanent (the **Level** and **Task** tabs).

Note: The parameter `MDLOPEN` is intended for the internal needs of the REXYGEN system and cannot be changed manually.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>prev</code>	Input for chaining tasks	Long (I32)
-------------------	--------------------------	------------

Parameter

<code>factor</code>	Execution factor	⊙1	Long (I32)
<code>start</code>	Start tick		Long (I32)
<code>stop</code>	Stop tick	⊙1	Long (I32)
<code>stack</code>	Stack size [bytes]	⊙10240	Long (I32)
<code>filename</code>	Corresponding MDL file		String
<code>MDLOPEN</code>	Is the corresponding MDL file open?		Bool

Output

<code>next</code>	Output for chaining tasks	Long (I32)
-------------------	---------------------------	------------

TIODRV – REXYGEN input/output driver with tasks

Block Symbol

Licence: [STANDARD](#)



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 **prev** input of the **IOTASK** block must be connected with the **Tasks** output of the **TIODRV** block. If the driver allows so, the **next** output of a **TIODRV** block which is already included in the configuration can be used to add more tasks. The **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 **Diagnostics** section of the **REXYGEN Studio** program. The **cpu** parameter can be used to specify where the driver thread should run on multi-CPU devices.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

prev	Input for chaining drivers (with tasks)	Long (I32)
-------------	---	------------

Parameter

module	Module name		String
classname	I/O driver class name	⊙DrvClass	String
cfgname	Configuration file name	⊙iodrv.rio	String
factor	Execution factor	↓1 ⊙10	Long (I32)
stack	Stack size [bytes]	↓1024 ⊙10240	Long (I32)
pri	Driver thread logical priority	↓1 ↑31 ⊙3	Long (I32)
cpu	CPU core assigned to driver thread (-1=default, 0=core 0, 1=core 1, ...)	↓-1 ↑127 ⊙-1	Long (I32)

Output

<code>next</code>	Output for chaining drivers (with tasks)	Long (I32)
<code>Tasks</code>	Anchor for chain of I/O tasks	Long (I32)

WWW – Internal webserver 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 and 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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Parameter

Source	Source directory	String
Target	Target directory	String
Compression	Enable data compression	Bool

Chapter 3

INOUT – Input and output blocks

Contents

Display – Numeric display of input values	65
FromFile – From File	67
FromWorkspace – From Workspace	68
GotoTagVisibility – Visibility of the signal source	69
INCONN – Block for remote value acquirement	70
INQUAD, INOCT, INHEXD – Multi-input blocks	71
From, INSTD – Signal connection or input	73
IOASYNC – Asynchronous reading and writing	75
OUTCONN – Block for remote value setting	76
OUTQUAD, OUTOCT, OUTHEXD – Multi-output blocks	77
OUTRQUAD, OUTROCT, OUTRHEXD – Multi-output blocks with verification	79
OUTRSTD – Output block with verification	80
Goto, OUTSTD – Signal source or output	81
Inport, Outport – Input and output port	83
QFC – Quality flags coding	85
QFD – Quality flags decoding	86
SubSystem – Subsystem block	87
ToFile – To File	89
ToWorkspace – To Workspace	90
VIN – Validation of the input signal	91
VOUT – Validation of the output signal	92

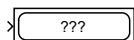
The INOUT library serves as a crucial interface in the REXYGEN system, enabling smooth interaction with input/output drivers. It is designed for efficient simultaneous signal processing, essential for fast control tasks. This library simplifies the connection between control algorithms and hardware, ensuring minimal latency. Additionally, it

provides advanced features, such as virtual linking (flags) of signals for increased clarity of diagrams and flexibility of subsystems.

Display – Numeric display of input values

Block Symbol

Licence: [STANDARD](#)



Function Description

The **Display** block shows input value in a selected format. A text suffix may be appended to the value by specifying the **suffix** parameter. An actual value is immediately displayed in **REXYGEN Studio** after turning on *Watch* mode. The conversion of the input into its text representation is performed on the target device at each **Decimation** period, so the displayed value can also be retrieved via the *REST* interface or used in the user interface.

The **Format** parameter allows selecting from various value display styles. The display modes differ for non-integer and integer values. The offered display styles include: **For non-integer values:**

- **1: Best fit:** Default display of most numbers in the "Full Precision" style, switches to "Scientific Long" style for extremely small or large values.
- **2: short:** Displays values with a maximum of 3 decimal places.
- **3: long:** Displays values with the maximum number of decimal places (up to 15).
- **4: short_e:** Exponential (scientific) display of values with a maximum of 3 decimal places.
- **5: long_e:** Exponential (scientific) display of values with the maximum number of decimal places (up to 15).
- **6: bank:** Displays values with 2 decimal places.

For integer values:

- **7: hex:** The number in hexadecimal format.
- **8: bin:** The number in binary format.
- **9: dec:** Standard display of numbers in decimal format.
- **10: oct:** The number in octal format.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

u	Input signal	Any
---	--------------	-----

Parameter

Format	Format of displayed value	⊙1	Long (I32)
	Best fit		
	short		
	long ..		
	short_e		
	long_e		
	bank ..		
	hex ...		
	bin ...		
	det ...		

Decimation	Value is evaluated in each <Decimation> period	Long (I32)
	↓1 ↑100000 ⊙1	

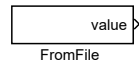
Suffix	A string to append to the value	String
--------	---------------------------------	--------

DispValue	Displayed value	String
-----------	-----------------	--------

FromFile — From File

Block Symbol

Licence: [STANDARD](#)



Function Description

This block corresponds to the block of the same name in Matlab/Simulink. For more information, see the Matlab/Simulink documentation.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Parameter

FileName	MAT file name or path to MAT file	String
-----------------	-----------------------------------	---------------

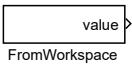
Output

value	Output signal	Any
--------------	---------------	------------

FromWorkspace – **From Workspace**

Block Symbol

Licence: [STANDARD](#)



Function Description

This block corresponds to the block of the same name in Matlab/Simulink. For more information, see the Matlab/Simulink documentation.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Parameter

VariableName	Name of the variable to load	String
--------------	------------------------------	--------

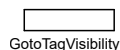
Output

value	Output signal	Any
-------	---------------	-----

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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

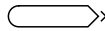
Parameter

GotoTag	Goto tag	String
----------------	----------	--------

INCONN – Block for remote value acquirement

Block Symbol

Licence: [STANDARD](#)



Function Description

The **INCONN** block allows for remote reading of parameter values (as well as inputs and outputs) of other blocks, similar to [GETPR](#), [GETPS](#), [GETPA](#), and others. Its use is particularly suitable in situations requiring a quick response, such as in time-critical tasks.

The **sc** parameter, which specifies the path to the target parameter, must be given as either a *relative path* or a *relative to task path* (see the documentation for the [GETPR](#) block). This means the **INCONN** block must be placed within the same task as the target parameter, regardless of their hierarchical levels. Unlike the **GETP** blocks, the **sc** parameter cannot be modified during runtime.

The data type of the **val** output is determined by the type of the read value. Unlike the **GETP** blocks, the **INCONN** block does not allow for a one-time read setting. The value is refreshed in every execution period.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Parameter

sc	String connection to the parameter	String
-----------	------------------------------------	---------------

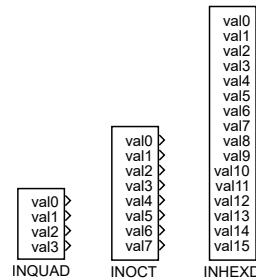
Output

val	Parameter value	Any
------------	-----------------	------------

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 `MBM__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 `MBM__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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Output

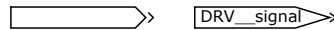
`val0..val15` Signal coming from I/O driver

Any

From, INSTD – Signal connection or input

Block Symbols

Licence: [STANDARD](#)



Function Description

The **From** (signal connection) and **INSTD** (standard input) blocks share the same symbol and are used to connect an input signal to the control algorithm at the **value** output. The output type is determined by the type of the input signal.

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** 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** 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** 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.

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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Output

<code>value</code>	Signal coming from I/O driver or Goto block	<code>Any</code>
--------------------	---	------------------

IOASYNC – Asynchronous reading and writing

Block Symbol

Licence: [STANDARD](#)



Function Description

The **IOASYNC** block is designed for asynchronous reading and writing of inputs and outputs. Unlike the [INSTD](#), [OUTSTD](#), [INOCT](#), [OUTOCT](#) blocks and their variants, the operation of the **IOASYNC** block is not controlled by the periodic timing of the driver. The main difference is that reading and writing occur exclusively at the moment of detecting a rising edge **off**→**on** signal on the respective inputs. Writing the value from the input **u** is triggered by a rising edge on the input **WF**, while reading to the output **y** is triggered by a rising edge on the input **RF**.

Unlike other blocks, where writing can be canceled by setting a special signal **_WriteEnable** to **off** (see documentation for the respective driver), the **IOASYNC** block ensures that at the startup of the executive, not even a one-time initialization of the value to be written occurs. This initialization may be undesirable in some applications.

To establish a connection with a specific driver signal, it is necessary to rename the block instance according to the established format of driver signal names, which uses a pair of underscores, similarly to the [INSTD](#) block.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	Signal going to I/O driver	Any
RF	Read force	Bool
WF	Write force	Bool

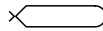
Output

y	Signal coming from I/O driver	Any
BUSY	Busy flag	Bool
DONE	Operation done	Bool

OUTCONN – Block for remote value setting

Block Symbol

Licence: [STANDARD](#)



Function Description

The **OUTCONN** block enables remote setting of parameter values of other blocks, similar to [SETPR](#), [SETPS](#), [SETPA](#), and others. Its use is particularly suitable in situations requiring a quick response, such as in time-critical tasks.

The **sc** parameter, which specifies the path to the target parameter, must be given as either a *relative path* or a *task-relative path* (see the documentation for the [SETPR](#) block). This means the **OUTCONN** block must be placed within the same task as the target parameter, regardless of their hierarchical levels. For details on formatting the path to the parameter, see the documentation for the [SETPR](#) block. Unlike the **SETP** blocks, the **sc** parameter cannot be modified during runtime.

The data type of the **val** input is determined by the type of the connected value. Unlike the **SETP** blocks, the **OUTCONN** block does not allow for a one-time setting. The value is set in every execution period.

Warning: Setting the connected input of a remote block may lead to undefined behavior. Similarly, setting a block parameter inside a subsystem when this parameter is set in the subsystem's mask.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

val	Parameter value	Any
------------	-----------------	-----

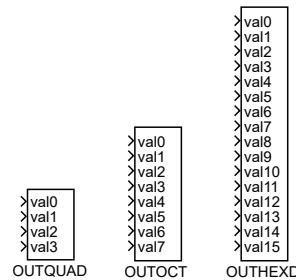
Parameter

sc	String connection to the parameter	String
-----------	------------------------------------	--------

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__D0**. 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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

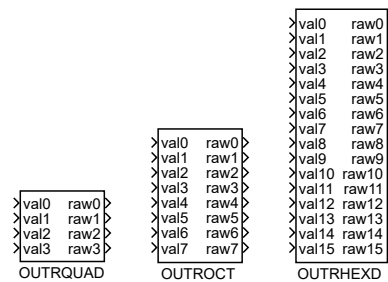
val0..val15 Signal going to I/O driver

Any

OUTRQUAD, OUTROCT, OUTRHEXD – Multi-output blocks with verification

Block Symbols

Licence: [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 **rawi** output:

- Through the value of the output, which can e.g. contain the real bit value in case of exceeding the limits of A/D 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 **rawi** 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.

The type and location of individual **val** and **raw** signals are described in the user manual of the corresponding driver.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

val0..val15 Signal going to I/O driver Any

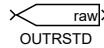
Output

raw0..raw15 Write operation result (see documentation) Any

OUTRSTD – Output block with verification

Block Symbol

Licence: [ADVANCED](#)



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 A/D 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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

value	Signal going to I/O driver	Any
--------------	----------------------------	------------

Output

raw	Write operation result (see documentation)	Any
------------	--	------------

Goto, OUTSTD – Signal source or output

Block Symbols

Licence: [STANDARD](#)



Function Description

The **Goto** (signal source) and **OUTSTD** (standard output) blocks have the same symbol and are used to connect the output signal from the control algorithm.

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 [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 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** 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 second parameter **TagVisibility** of the **Goto** block determines the visibility of the given block within the `.mdl` file. It can have the following values:

- **local**: Both blocks must be located at the same hierarchical level.
- **global**: Blocks can be placed anywhere in the given `.mdl` file.
- **scoped**: Blocks must be placed within the same subsystem or at any hierarchical level beneath the placement of the [GotoTagVisibility](#) block with the same **GotoTag** parameter.

If the given block is compiled as an `OUTSTD` block, this parameter is ignored.

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>`. E.g. the first digital output of a Modbus I/O device might be referenced by `MBM__D01`. 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__D0<id>` 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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

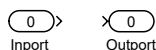
Input

value	Signal going to I/O driver or From block	Any
-------	--	-----

Inport, Outport – Input and output port

Block Symbols

Licence: [STANDARD](#)



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 (.mdl file) in this case. The corresponding **Inport** and **Outport** blocks should have the same **Block name**. The connection between 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!

In the **Inport** and **Outport** blocks, it's also possible to explicitly specify the data type of the transferred value using the **OutDataTypeStr** parameter. If no value is selected, or the option **Inherit: auto** is chosen, the value type is determined automatically.

The **Description** parameter can be used to add a textual description of the block. This description is displayed in the properties of the subsystem and library block if **Inport** or **Outport** is used to define the inputs and outputs of the subsystem.

Warning: The blocks **Inport** and **Outport** should not be used to connect arrays and other references between tasks (references often have **ref** in name and have a type **intptr** in the **Diagnostics** section of the REXYGEN Studio program). Consistence is not guaranteed in this case; incorrect value could be get and runtime code can crash in worst case scenario. Typical behaviour is that some array members are from one period of execution and other members of array from next period. The blocks [SETPA](#) and

[GETPA](#) ensure consistent read and write of the array between task. Some blocks guarantee consistence of references over task boundary (for example [RM_AxisSpline](#)). In this case, this is explicitly stated in the block manual.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

value	Value going to the output pin or Inport	Any
-------	---	-----

Parameter

OutDataTypeStr	Data type of item	String
Inherit	auto	
	double	
	single	
	uint8	
	int16	
	uint16	
	int32	
	uint32	
	boolean	
	float	
	int64	
	string	

Description	Description of the port	String
Port	Ordering of the output pins	Long (I32)

QFC – Quality flags coding

Block Symbol

Licence: [STANDARD](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

iq	Basic quality type flags	Long (I32)
is	Substatus flags	Long (I32)
il	Limits flags	Long (I32)

Output

iqf	Bit combination of the input signals	Long (I32)
------------	--------------------------------------	------------

QFD – Quality flags decoding

Block Symbol

Licence: [STANDARD](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

iqf	Quality flags to be decomposed	Long (I32)
-----	--------------------------------	------------

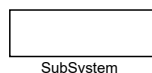
Output

iq	Basic quality type flags	Long (I32)
is	Substatus flags	Long (I32)
il	Limits flags	Long (I32)

SubSystem – Subsystem block

Block Symbol

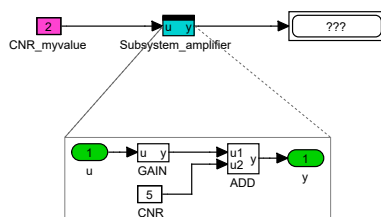
Licence: [STANDARD](#)



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.



A subsystem can be created in two ways:

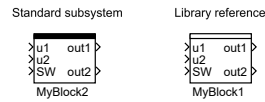
- By copying the **SubSystem** block from the **INOUT** library into the given schematic (file `.mdl`). After opening the created subsystem, blocks can be added to it, including input ports **Inport** and output ports **Outport**.
- By selecting a group of blocks and choosing the **Create Subsystem** command from the **Edit** menu. The selected blocks are replaced by a subsystem, which, when opened, shows the original blocks and **Inport** and **Outport** blocks facilitating connections with blocks at the higher (original) level.

Once the subsystem is created, it can be entered by double-clicking.

For **SubSystem**, it is possible to create a so-called subsystem mask and define parameters whose values can be used inside the subsystem. Select the subsystem and go to the menu **Edit**→**Declaration of parameters**. A dialog will appear where you can define parameters and their labels (meanings). Once a mask is defined for a subsystem, it starts behaving like a standard block – double-clicking it will open the *Block properties* dialog.

This dialog contains the parameters defined in the subsystem mask. If you need to edit the content of a masked subsystem, select it and go to the menu **Edit**→**Open Subsystem**.

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. See image below.



Please refer to [3] 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.

This block does not propagate the signal quality. More information can be found in the 1.4 section.

ToFile – To File

Block Symbol

Licence: [STANDARD](#)



Function Description

This block corresponds to the block of the same name in Matlab/Simulink. For more information, see the Matlab/Simulink documentation.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

u	Input signal	Any
---	--------------	-----

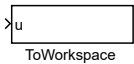
Parameter

FileName	MAT file name or path to MAT file	String
----------	-----------------------------------	--------

ToWorkspace – **To Workspace**

Block Symbol

Licence: [STANDARD](#)



Function Description

This block corresponds to the block of the same name in Matlab/Simulink. For more information, see the Matlab/Simulink documentation.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

u	Input signal	Any
---	--------------	-----

Parameter

VariableName	Name of the variable	String
--------------	----------------------	--------

VIN – Validation of the input signal

Block Symbol

Licence: [STANDARD](#)



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.

if the input signal **u** is evaluated as Quality Good **QG = on**, it is fed to the output **yg**. In case of signal quality problems, a substitute signal from the input **sv** (substitution variable) is used for the output.

This block propagates the signal quality. More information can be found in the 1.4 section.

Input

u	Analog input of the block	Any
sv	Substitute value for an error case	Any

Parameter

GU	Acceptability of UNCERTAIN quality	Bool
	off ... Uncertain quality unacceptable	
	on ... Uncertain quality acceptable	

Output

yg	Validated signal	Any
QG	Input signal acceptability indicator	Bool
iqf	Complete quality flag of the input signal	Long (I32)

VOUT – Validation of the output signal

Block Symbol

Licence: [STANDARD](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

u	Input signal requiring quality flags modification	Any
iqf	Desired quality flags	Long (I32)

Output

yq	Resulting composed signal	Any
-----------	---------------------------	------------

Chapter 4

MATH – Math blocks

Contents

ABS – Absolute value	95
ADD – Addition of two signals	96
ADDQUAD, ADDOCT, ADDHEXD – Multi-input addition	97
CNB – Boolean (logic) constant	98
CNE – Enumeration constant	99
CNI – Integer constant	100
CNR – Real constant	101
DIF – Difference	102
DIV – Division of two signals	103
EAS – Extended addition and subtraction	104
EMD – Extended multiplication and division	105
FNX – Evaluation of single-variable function	106
FNXY – Evaluation of two-variables function	109
GAIN – Multiplication by a constant	111
GRADS – Gradient search optimization	112
IADD – Integer addition	114
IDIV – Integer division	116
IMOD – Remainder after integer division	117
IMUL – Integer multiplication	118
ISUB – Integer subtraction	120
LIN – Linear interpolation	122
MUL – Multiplication of two signals	123
NANINF – Block for checking NaN and Inf values	124
POL – Polynomial evaluation	126
REC – Reciprocal value	127
REL – Relational operator	128

RTOI – Real to integer number conversion	129
SQR – Square value	131
SQRT – Square root	132
SUB – Subtraction of two signals	133
UTOI – Unsigned to signed integer number conversion	134

The MATH library offers a comprehensive collection of mathematical operations and functions. It includes basic arithmetic blocks like **ADD**, **SUB**, **MUL**, and **DIV** for standard calculations, and more specialized blocks such as **ABS** for absolute values, **SQRT** for square roots, and **SQR** for squaring. Advanced functionalities are provided by blocks like **LIN** for linear transformations, **POL** for polynomial evaluations, and **FNX**, **FNXY** for customizable mathematical functions. The library also features integer-specific operations through blocks like **IADD**, **IMUL**, **IDIV**, and **IMOD**.

ABS – Absolute value

Block Symbol

Licence: [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 **sgn** output denotes the sign of the input signal.

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

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Output

y	Absolute value of the input signal	Double (F64)
sgn	Indication of the input signal sign	Long (I32)

ADD – Addition of two signals

Block Symbol

Licence: [STANDARD](#)



Function Description

The **ADD** blocks sums two analog input signals. The output is given by

$$y = u1 + u2.$$

Consider using the [ADDOCT](#) block for addition or subtraction of multiple signals.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

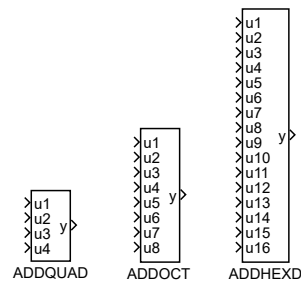
Output

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 $y = u1 + u2 + u3 + u4 + u5 + u6 + u7 + \dots + u16$. For e.g. **n1=2,5,7**, the block implements the function $y = u1 - u2 + u3 + u4 - u5 + u6 - u7 + \dots + u16$.

Note that the [ADD](#) and [SUB](#) blocks are available for simple addition and subtraction operations.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u1..u16	Analog input of the block	Double (F64)
----------------	---------------------------	--------------

Parameter

n1	List of signals to subtract	Long (I32)
-----------	-----------------------------	------------

Output

y	Sum of the input signals	Double (F64)
----------	--------------------------	--------------

CNB – Boolean (logic) constant

Block Symbol

Licence: [STANDARD](#)



Function Description

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

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Parameter

YCN	Boolean constant	on Bool
	off ... Disabled	
	on Enabled	

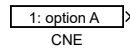
Output

Y	Logical output of the block	Bool
---	-----------------------------	------

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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Parameter

yenum	Enumeration constant	⊙1: option A	String
pupstr	Popup list definition		String
		⊙1: option A 2: option B 3: option C	

Output

iy	Integer output of the block	Long (I32)
-----------	-----------------------------	------------

CNI – Integer constant

Block Symbol

Licence: [STANDARD](#)



Function Description

The CNI block stands for an integer constant.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Parameter

icn	Integer constant	⊙1	Long (I32)
vtype	Numeric type	⊙4	Long (I32)
	2 Byte (U8)		
	3 Short (I16)		
	4 Long (I32)		
	5 Word (U16)		
	6 DWord (U32)		
	10 Large (I64)		

Output

iy	Integer output of the block	Long (I32)
----	-----------------------------	------------

CNR – Real constant

Block Symbol

Licence: [STANDARD](#)



Function Description

The **CNR** block stands for a real constant.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Parameter

<code>ycn</code>	Real constant	⊙1.0 Double (F64)
------------------	---------------	---------------------

Output

<code>y</code>	Analog output of the block	Double (F64)
----------------	----------------------------	--------------

DIF – Difference

Block Symbol

Licence: [STANDARD](#)



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).

The parameter **ISSF** sets the behavior of the block in the first cycle of task execution. If **off**, $y = u$ will be output in the first cycle. For the value **on**, the output will be $y = 0$ in the first cycle.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	Analog input of the block	Double (F64)
R1	Block reset	Bool
HLD	Hold	Bool

Parameter

ISSF	Zero output at start-up	Bool
	off ... Non-zero output in the first cycle	
	on Zero output in the first cycle	

Output

y	Difference of the input signal	Double (F64)
-----	--------------------------------	--------------

DIV – Division of two signals

Block Symbol

Licence: [STANDARD](#)



Function Description

The **DIV** block divides two analog input signals $y = u1/u2$. In case $u2 = 0$, the output **E** is set to **on** and the output **y** is substituted by **yerr**.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

yerr	Substitute value for an error case	⊙1.0 Double (F64)
-------------	------------------------------------	-------------------

Output

y	Quotient of the inputs	Double (F64)
E	Error flag - division by zero	Bool
	off ... No error	
	on An error occurred	

EAS – Extended addition and subtraction

Block Symbol

Licence: [STANDARD](#)



Function Description

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

$$y = a * u1 + b * u2 + c * u3 + d * u4 + y0.$$

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u1..u4	Analog input of the block	Double (F64)
---------------	---------------------------	---------------------

Parameter

a	Weighting coefficient of the u1 input	⊙1.0	Double (F64)
b	Weighting coefficient of the u2 input	⊙1.0	Double (F64)
c	Weighting coefficient of the u3 input	⊙1.0	Double (F64)
d	Weighting coefficient of the u4 input	⊙1.0	Double (F64)
y0	Additive constant (bias)		Double (F64)

Output

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

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

$$y = \frac{(a * u1 + a0)(b * u2 + b0)}{(c * u3 + c0)(d * u4 + d0)}. \quad (4.1)$$

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**.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u1..u4	Analog input of the block	Double (F64)
---------------	---------------------------	--------------

Parameter

a	Weighting coefficient of the u1 input	⊙1.0	Double (F64)
a0	Additive constant for u1 input		Double (F64)
b	Weighting coefficient of the u2 input	⊙1.0	Double (F64)
b0	Additive constant for u1 input		Double (F64)
c	Weighting coefficient of the u3 input	⊙1.0	Double (F64)
c0	Additive constant for u1 input		Double (F64)
d	Weighting coefficient of the u4 input	⊙1.0	Double (F64)
d0	Additive constant for u1 input		Double (F64)
yerr	Substitute value for an error case	⊙1.0	Double (F64)

Output

y	Analog output of the block	Double (F64)
E	Error flag - division by zero	Bool
	off ... No error	
	on An error occurred	

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 **ifn** parameter determines the active function.

List of functions:

ifn: shortcut	function	constraints on u
1: acos	arccosine	$u \in \langle -1.0, 1.0 \rangle$
2: asin	arcsine	$u \in \langle -1.0, 1.0 \rangle$
3: atan	arctangent	–
4: ceil	rounding towards the nearest higher integer	–
5: cos	cosine	–
6: cosh	hyperbolic cosine	–
7: exp	exponential function e^u	–
8: exp10	exponential function 10^u	–
9: fabs	absolute value	–
10: floor	rounding towards the nearest lower integer	–
11: log	logarithm	$u > 0$
12: log10	decimal logarithm	$u > 0$
13: random	arbitrary number $z \in \langle 0, 1 \rangle$ (u independent)	–
14: sin	sine	–
15: sinh	hyperbolic sine	–
16: sqr	square function	–
17: sqrt	square root	$u > 0$
18: srand	changes the seed for the random function to u	$u \in \mathbb{N}$
19: tan	tangent	–
20: tanh	hyperbolic tangent	–

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**).

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

ifn	Function type	⊙1 Long (I32)
	1 acos	
	2 asin	
	3 atan	
	4 ceil	
	5 cos	
	6 cosh	
	7 exp	
	8 exp10	
	9 fabs	
	10 floor	
	11 log	
	12 log10	
	13 random	
	14 sin	
	15 sinh	
	16 sqr	
	17 sqrt	
	18 srand	
	19 tan	
	20 tanh	
yerr	Substitute value for an error case	Double (F64)

Output

y	Result of the selected function	Double (F64)
E	Error indicator	Bool
	off ... No error	
	on An error occurred	

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 **ifn** parameter determines the active function.

List of functions:

ifn: shortcut	function	constraints on u1, u2
1: atan2	arctangent $u1/u2$	–
2: fmod	remainder after division $u1/u2$	$u2 \neq 0.0$
3: pow	exponentiation of the inputs $y = u1^{u2}$	–

The **atan2** function result belongs to the interval $\langle -\pi, \pi \rangle$. 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 $u1 = i \cdot u2 + y$, 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**).

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

ifn	Function type	⊙1 Long (I32)
	1 atan2	
	2 fmod	
	3 pow	
yerr	Substitute value for an error case	Double (F64)

Output

y	Result of the selected function	Double (F64)
E	Error indicator	Bool
	off ... No error	
	on An error occurred	

GAIN – Multiplication by a constant

Block Symbol

Licence: [STANDARD](#)



Function Description

The **GAIN** block multiplies the analog input u by a real constant k . The output is then

$$y = ku.$$

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

k	Gain	$\odot 1.0$ Double (F64)
-----	------	--------------------------

Output

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

GRADS – Gradient search optimization

Block Symbol

Licence: [ADVANCED](#)



Function Description

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

$$\begin{aligned} grad &= (\mathbf{f}_i - \mathbf{f}_{i-1}) * (dx)_{i-1} \\ (dx)_i &= -\mathbf{gamma} * grad, \end{aligned}$$

where i stands for i -th iteration. The step size is restricted to lie within the interval $\langle \mathbf{dmin}, \mathbf{dmax} \rangle$. The value of the optimized variable for the next iteration is given by

$$x_{i+1} = x_i + (dx)_i$$

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

f	Value of the optimized function $f(\cdot)$ for given \mathbf{x}	Double (F64)
x0	Optimization starting point	Double (F64)
START	Starting signal (rising edge)	Bool
BRK	Termination signal	Bool

Parameter

xmin	Lower limit for the \mathbf{x} variable	Double (F64)
xmax	Upper limit for the \mathbf{x} variable	⊙10.0 Double (F64)
gamma	Step size coefficient	⊙0.3 Double (F64)
d0	Initial step size	⊙0.05 Double (F64)

<code>dmin</code>	Minimum step size	⊙0.01	Double (F64)
<code>dmax</code>	Maximum step size	⊙1.0	Double (F64)
<code>n</code>	Iteration period (in sampling periods T_s)	⊙100	Long (I32)
<code>itermax</code>	Maximum number of iterations	⊙20	Long (I32)

Output

<code>x</code>	Current value of the optimized variable	Double (F64)
<code>xopt</code>	Resulting optimal value of the x variable	Double (F64)
<code>fopt</code>	Resulting optimal value of the function f	Double (F64)
<code>BSY</code>	Busy flag	Bool
<code>iter</code>	Number of current iteration	Long (I32)
<code>E</code>	Error indicator	Bool
	<code>off ...</code> No error	
	<code>on</code> An error occurred	
<code>iE</code>	Error code	Long (I32)
	1 x out of limits	
	2 x at the limit	

IADD – Integer addition

Block Symbol

Licence: [STANDARD](#)



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 **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 = 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 = on** and the **n** output to the nearest displayable value. For the above mentioned example we get $30000 + 2770 = 32767$.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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)

Parameter

vtype	Numeric type	⊙4	Long (I32)
	2 Byte (U8)		
	3 Short (I16)		
	4 Long (I32)		
	5 Word (U16)		
	6 DWord (U32)		
	10 Large (I64)		
SAT	Saturation (overflow) checking		Bool
	off ... Overflow is not checked		
	on Overflow is checked		

Output

n	Integer sum of the input signals	Long (I32)
---	----------------------------------	------------

E	Error indicator	Bool
	off ... No error	
	on An error occurred	

IDIV – Integer division

Block Symbol

Licence: [STANDARD](#)



Function Description

The **IDIV** block performs an integer division of two integer input signals, $n = i1 \div i2$, 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 $i2 = 0$, the output **error** is set to **on** and the output **n** is substituted by $n = nerr$.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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)

Parameter

vtype	Numeric type	⊙4	Long (I32)
	2 Byte (U8)		
	3 Short (I16)		
	4 Long (I32)		
	5 Word (U16)		
	6 DWord (U32)		
	10 Large (I64)		
nerr	Substitute value for an error case	⊙1	Long (I32)

Output

n	Integer quotient of the inputs	Long (I32)
E	Error flag - division by zero	Bool
	off ... No error	
	on ... An error occurred	

IMOD – Remainder after integer division

Block Symbol

Licence: [STANDARD](#)



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 **on** and the output **n** is substituted by $n = nerr$.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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)

Parameter

vtype	Numeric type	$\odot 4$ Long (I32)
	2 Byte (U8)	
	3 Short (I16)	
	4 Long (I32)	
	5 Word (U16)	
	6 DWord (U32)	
	10 Large (I64)	
nerr	Substitute value for an error case	$\odot 1$ Long (I32)

Output

n	Remainder after integer division	Long (I32)
E	Error flag - division by zero	Bool
	off ... No error	
	on ... An error occurred	

IMUL – Integer multiplication

Block Symbol

Licence: [STANDARD](#)



Function Description

The IMUL block multiplies 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 **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 = 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 $2000 * 20 = -25536$.

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 $2000 * 20 = 32767$.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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)

Parameter

vtype	Numeric type	⊙4	Long (I32)
	2 Byte (U8)		
	3 Short (I16)		
	4 Long (I32)		
	5 Word (U16)		
	6 DWord (U32)		
	10 Large (I64)		
SAT	Saturation (overflow) checking		Bool
	off ... Overflow is not checked		
	on Overflow is checked		

Output

n	Integer product of the input signals	Long (I32)
E	Error indicator	Bool
	off ... No error	
	on An error occurred	

ISUB – Integer subtraction

Block Symbol

Licence: [STANDARD](#)



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 **vtype** 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 $30000 - -2770 = -32766$

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 $30000 - -2770 = 32767$.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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)

Parameter

vtype	Numeric type	⊙4 Long (I32)
	2 Byte (U8)	
	3 Short (I16)	
	4 Long (I32)	
	5 Word (U16)	
	6 DWord (U32)	
	10 Large (I64)	
SAT	Saturation (overflow) checking	Bool
	off ... Overflow is not checked	
	on Overflow is checked	

Output

n	Integer difference between the input signals	Long (I32)
E	Error indicator	Bool
	off ... No error	
	on An error occurred	

LIN – Linear interpolation

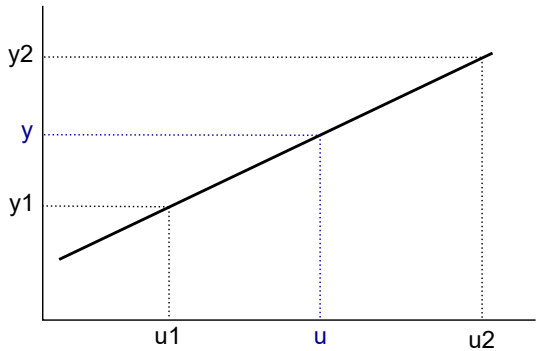
Block Symbol

Licence: [STANDARD](#)



Function Description

The LIN block performs linear interpolation. The following figure illustrates the influence of the input u and given interpolation points $[u1, y1]$ and $[u2, y2]$ on the output y .



This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

$u1$	x-coordinate of the 1st interpolation node	Double (F64)
$y1$	y-coordinate of the 1st interpolation node	Double (F64)
$u2$	x-coordinate of the 2nd interpolation node	$\odot 1.0$ Double (F64)
$y2$	y-coordinate of the 2nd interpolation node	$\odot 1.0$ Double (F64)

Output

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

MUL – Multiplication of two signals

Block Symbol

Licence: [STANDARD](#)



Function Description

The **MUL** block multiplies two analog input signals $y = u1 \cdot u2$.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Output

y	Product of the input signals	Double (F64)
---	------------------------------	--------------

NANINF – Block for checking NaN and Inf values

Block Symbol

Licence: [STANDARD](#)



Function Description

The **NANINF** block serves to detect and correct unusual values on an analog input **u**. If the input value is a standard number, it is directly forwarded to the output **y** ($y = u$), and the output **is** is set to 0. In cases where the input value is infinity ($\pm\text{Inf}$) or is of the type "not a number" (NaN), a replacement value defined in the respective parameter **infp**, **infn** or **nan** is sent to the output **y** with a bad quality (BAD) label. Simultaneously, a code indicating the type of abnormality of the input value is sent to the output **is**.

Note: From the perspective of a mathematical coprocessor, Inf or NaN values are almost normal values that can be operated with in all operations. They may arise, for example, when dividing by zero or when taking the square root of a negative number. From the perspective of the control system, however, these are nonsensical values that definitely cannot be set to a physical output. The **REXYGEN** system understands this and such values are not generated by its blocks (e.g., the [SQRT](#) block has a replacement value for negative numbers, similarly the [DIV](#) block for division by 0). However, sometimes such non-standard values occur, and then it is necessary to have the means to deal with them, which is where this block comes into play.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u1	Analog input of the block	Double (F64)
-----------	---------------------------	--------------

Parameter

infp	Substitute value for +Inf	Double (F64)
infn	Substitute value for -Inf	Double (F64)
nan	Substitute value for NaN	Double (F64)

Output

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

is **Status** (0:OK, 1:+Inf, 2:-Inf, 3:NaN)
0 **OK**
1 **+Inf**
2 **-Inf**
3 **NaN**

Long (I32)

POL – Polynomial evaluation

Block Symbol

Licence: [STANDARD](#)



Function Description

The POL block evaluates the polynomial of the form:

$$y = a_0 + a_1u + a_2u^2 + a_3u^3 + a_4u^4 + a_5u^5 + a_6u^6 + a_7u^7 + a_8u^8.$$

The polynomial is internally evaluated by using the Horner scheme to

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

a0..a8	Coefficient of the polynomial	Double (F64)
--------	-------------------------------	--------------

Output

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

REC – Reciprocal value

Block Symbol

Licence: [STANDARD](#)



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** = **on** and the output is set to the substitut-

tional

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

yerr	Substitute value for an error case	⊙1.0 Double (F64)
-------------	------------------------------------	-------------------

Output

y	Analog output of the block	Double (F64)
E	Error flag - division by zero	Bool
	off ... No error	
	on An error occurred	

REL – Relational operator

Block Symbol

Licence: [STANDARD](#)



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 = on** when relation holds, otherwise it is zero (relation does not hold). The binary operation codes are listed below.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

irel	Relation type	⊙1 Long (I32)
1 equality (==)	
2 inequality (!=)	
3 less than (<)	
4 greater than (>)	
5 less or equal (<=)	
6 greater or equal (>=)	

Output

Y	Logical output of the block	Bool
----------	-----------------------------	------

RTOI – Real to integer number conversion

Block Symbol

Licence: [STANDARD](#)



Function Description

The **RTOI** 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). This limits are valid if the **vtype** parameter has default value.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

vtype	Numeric type	⊙4 Long (I32)
	2 Byte (U8)	
	3 Short (I16)	
	4 Long (I32)	
	5 Word (U16)	
	6 DWord (U32)	
	10 Large (I64)	
SAT	Saturation (overflow) checking	⊙on Bool
	off ... Overflow is not checked	
	on Overflow is checked	

Output

i	Rounded and converted input signal	Long (I32)
---	------------------------------------	------------

SQR – Square value

Block Symbol

Licence: [STANDARD](#)



Function Description

The **SQR** block raises the input **u** to the power of 2. The output is then

$$y = u^2.$$

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Output

y	Square of the input signal	Double (F64)
----------	----------------------------	--------------

SQRT – Square root

Block Symbol

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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

yerr	Substitute value for an error case	⊙1.0 Double (F64)
-------------	------------------------------------	-------------------

Output

y	Square root of the input signal	Double (F64)
E	Error indicator	Bool
	off ... No error	
	on An error occurred	

SUB – Subtraction of two signals

Block Symbol

Licence: [STANDARD](#)



Function Description

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

$$y = u1 - u2.$$

Consider using the [ADDOCT](#) block for addition or subtraction of multiple signals.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u1	Analog input of the block	Double (F64)
u2	Analog input of the block	Double (F64)

Output

y	Difference between input signals	Double (F64)
---	----------------------------------	--------------

UTOI – Unsigned to signed integer number conversion

Block Symbol

Licence: [STANDARD](#)



Function Description

The **UTOI** block facilitates the conversion of an unsigned integer to a signed integer using two's complement representation, which is the common representation used in processors. For instance, in 8-bit representation, the number -1 is represented as 255, and in 16-bit representation as 65535. The parameter **bits** determines which bit representation is assumed.

This block is primarily used in scenarios where a value from a driver contains multiple signals extracted by masking (typically using [INTSM](#) or [BITOP](#) blocks). The result of this masking is always an unsigned (positive) number. However, if the signal from the driver is meant to be interpreted as a signed number, this block is used to obtain the correct value.

Since processors may vary in how they store multi-byte numbers (most commonly in little-endian format, where the less significant byte is stored at a lower address, but big-endian format processors also exist, where the opposite is true), the **UTOI** block offers the option to swap the byte order if it has not been handled by the driver. This adjustment is facilitated by the **SWAP** parameter.

Caution: Swapping the byte order (by setting **SWAP=on**) typically addresses issues with different byte orders in the processor only for **bits=16** or **bits=32** values.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	Unsigned input signal	↓-9.22337E+18 ↑9.22337E+18	Large (I64)
----------	-----------------------	----------------------------	-------------

Parameter

bits	Valid (LSB) bits in input signal	↓2 ↑64 ○16	Long (I32)
SWAP	Swap input byte order		Bool

Output

i	Converted (signed) input signal	Large (I64)
----------	---------------------------------	-------------

Chapter 5

ANALOG – Analog signal processing

Contents

ABSR0T – Absolute rotation (multiturn extension of the position sensor)	137
ASW – Switch with automatic selection of input	139
AVG – Moving average filter	141
AVS – Motion control unit	142
AVSI – Smooth trajectory interpolation	143
BPF – Band-pass filter	146
CMP – Comparator with hysteresis	147
CNDR – Nonlinear conditioner	148
DEL – Delay with initialization	150
DELM – Time delay	151
DER – Derivation, filtering and prediction from the last n+1 samples	152
EVAR – Moving mean value and standard deviation	154
INTE – Controlled integrator	155
KDER – Derivation and filtering of the input signal	157
LPF – Low-pass filter	159
MINMAX – Running minimum and maximum	161
NSCL – Nonlinear scaling factor	162
OSD – One Step Delay	163
RDFT – Running discrete Fourier transform	164
RLIM – Rate limiter	166
S10F2 – One of two analog signals selector	167
SAI – Safety analog input	170
SEL – Selector switch for analog signals	173

SELQUAD, SELOCT, SELHEXD – Selector switch for analog signals . . .	174
SHIFTOCT – Data shift register	176
SHLD – Sample and hold	178
SINT – Simple integrator	179
SPIKE – Spike filter	180
SSW – Simple switch	182
SWR – Selector with ramp	183
VDEL – Variable time delay	184
ZV4IS – Zero vibration input shaper	185

Library presents a versatile range of functional blocks, designed for control and signal processing applications. It includes blocks like [ASW](#), [AVG](#), [BPF](#), and [DEL](#), which provide functionalities from signal manipulation and averaging to filtering and complex conditional operations, catering to a broad spectrum of system requirements and scenarios.

ABSROT – Absolute rotation (multiturn extension of the 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 $+175^\circ$) 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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	Signal from the absolute position sensor	Double (F64)
R1	Block reset	Bool

Parameter

lolim	Lower limit of the sensor reading	$\ominus -3.14159265$	Double (F64)
hilim	Upper limit of the sensor reading	$\odot 3.14159265$	Double (F64)
tol	Tolerance for the mid-point indicator	$\odot 0.5$	Double (F64)
hys	Hysteresis for the mid-point indicator		Double (F64)

RESR	Flag for resetting the revolutions counter	Bool
	off ... Reset only the estimated position	
	on Reset also the revolutions counter	

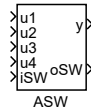
Output

y	Position output	Double (F64)
irev	Number of revolutions	Long (I32)
MPI	Mid-point indicator	Bool
OLI	Off-limits indicator	Bool

ASW – Switch with automatic selection of input

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **ASW** block copies one of the inputs $u1, \dots, u4$ or one of the parameters $p1, \dots, p4$ 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. $y = p1$ for $iSW = -1$, $y = u3$ 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 **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: $p4, p3, p2, p1, u4, u3, u2, u1$. 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 $p1, \dots, p4$. 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 $u1, \dots, u4$ 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 [LPBRK](#) block(s) must be used in appropriate positions.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

$u1..u4$	Analog input of the block	Double (F64)
iSW	Active signal or parameter selector	Long (I32)

Parameter

delta	Threshold for detecting a change	$\odot 1e-06$ Double (F64)
--------------	----------------------------------	----------------------------

p1..p4	Parameter to be selected	Double (F64)
--------	--------------------------	--------------

Output

y	The selected signal or parameter	Double (F64)
oSW	Identifier of the selected signal or parameter	Long (I32)

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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

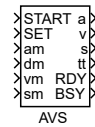
n	Number of samples for averaging	↓1 ↑100000000 ⊙10	Long (I32)
nmax	Allocated size of array	↓10 ↑100000000 ⊙100	Long (I32)

Output

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

AVS – Motion control unit

Block Symbol

Licence: [ADVANCED](#)

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**.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

START	Starting signal (rising edge)	Bool
SET	Initialize/compute the trajectory	Bool
am	Maximal allowed acceleration [m/s ²]	Double (F64)
dm	Maximal allowed deceleration [m/s ²]	Double (F64)
vm	Maximum allowed velocity [m/s]	Double (F64)
sm	Desired final position [m]	Double (F64)

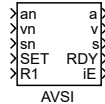
Output

a	Acceleration [m/s ²]	Double (F64)
v	Velocity [m/s]	Double (F64)
s	Position [m]	Double (F64)
tt	Time [s]	Double (F64)
RDY	Outputs valid (ready flag)	Bool
BSY	Busy flag	Bool

AVSI – Smooth trajectory interpolation

Block Symbol

Licence: [ADVANCED](#)



Function Description

The functional block **AVSI** - Acceleration (A), Velocity (V), Distance (S) Interpolation (I) - is designed for signal interpolation, especially in motion control applications. Its main purpose is to generate smooth sequences of position (distance), velocity, and acceleration based on discrete input values. This block is inspired by the functionality of [RM_AxisSpline](#) and offers similar interpolation methods.

The **AVSI** block accepts inputs **sn** (position), **vn** (velocity), and **an** (acceleration), which are generated by an external function (outside this block) with a certain period defined by the **RemTs** parameter. The values at the block's input are updated on the rising edge of the **SET** signal.

Interpolation between individual inputs is carried out with the aim of creating smooth transitions and ensuring continuous motion control or other applications requiring signal regulation and its derivatives. The block supports various interpolation methods determined by the **Mode** parameter, corresponding to the options in [RM_AxisSpline](#). The supported methods include:

- **1: linear:** Position is interpolated linearly, velocity as the derivative of position, acceleration is 0 (i.e., velocity is a piecewise constant function with jumps).
- **2: cubic spline:** Position is a 3rd order polynomial calculated based on the position and velocity at the beginning and end of the interval; velocity is the derivative of position, acceleration is the derivative of velocity.
- **3: quintic spline:** Position is a 5th order polynomial calculated based on the position, velocity, and acceleration at the beginning and end of the interval; velocity is the derivative of position, acceleration is the derivative of velocity.
- **4: cubic approximation (B-spline):** Position is a 3rd order polynomial calculated based on two positions before and two positions after the current interval; the interpolated function may not exactly pass through the given points; velocity is the derivative of position, acceleration is the derivative of velocity.
- **5: quintic approximation (B-spline):** Position is a 5th order polynomial calculated based on three positions before and three positions after the current interval;

the interpolated function may not exactly pass through the given points; velocity is the derivative of position, acceleration is the derivative of velocity.

- **6: all linear:** Position, velocity, and acceleration are independently interpolated linearly, i.e., velocity does not precisely correspond to the derivative of position, and acceleration does not precisely correspond to the derivative of velocity.
- **7: all cubic:** Both position and velocity are interpolated by a 3rd order polynomial independently, i.e., velocity does not exactly correspond to the derivative of position.
- **8:** *reserved for future use.*
- **9:** *reserved for future use.*

Due to its operating principle, the **AVSI** block introduces signal delay, where active generation of values begins only after two complete **RemTs** periods from the first rising edge of the **SET** signal. For B-spline interpolation methods, a larger number of samples is required to start interpolation.

The **AVSI** block is primarily intended for motion control applications but can also be used for other types of signals and their derivatives. Its implementation allows for more efficient and smoother transitions between individual state values without the need for complex external control.

When using it, it is important to correctly set the **RemTs** period corresponding to the input value generator and choose the appropriate **Mode** for the desired type of interpolation.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

an	Next (remote) period acceleration [m/s ²]	Double (F64)
vn	Next (remote) period velocity [m/s]	Double (F64)
sn	Next (remote) period position [m]	Double (F64)
SET	Accept input on rising edge	Bool
R1	Block reset	Bool
dm	Maximal allowed deceleration in case of error [m/s ²]	Double (F64)

Parameter

RemTs	Remote signal generator period	⊙0.0 Double (F64)
--------------	--------------------------------	-------------------

Mode	Algorithm for interpolation	⊙9 Long (I32)
1	linear	
2	cubic spline	
3	quintic spline	
4	cubic approximation (B-spline)	
5	quintic approximation (B-spline)	
6	all linear	
7	all cubic	
8	—	
9	—	

Output

a	Acceleration [m/s ²]	Double (F64)
v	Velocity [m/s]	Double (F64)
s	Position [m]	Double (F64)
RDY	Outputs valid (ready flag)	Bool
iE	Error code	Bool

BPF – Band-pass filter

Block Symbol

Licence: [STANDARD](#)



Function Description

The BPF implements a second order filter in the form

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

where a and ξ are the block parameters **fm** and **xi** respectively. The **fm** parameter defines the middle of the frequency transmission band and **xi** is the relative damping coefficient.

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**.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	Input signal to be filtered	Double (F64)
R1	Block reset	Bool
HLD	Hold	Bool

Parameter

fm	Peak frequency [Hz]	⊙1.0	Double (F64)
xi	Relative damping coefficient	⊙0.707	Double (F64)
ISSF	Steady state at start-up		Bool
	off ... Zero initial state		
	on Initial steady state		

Output

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

CMP – Comparator with hysteresis

Block Symbol

Licence: [STANDARD](#)



Function Description

The **CMP** block compares the inputs **u1** and **u2** with the hysteresis **h** as follows:

$$\begin{aligned} Y_{-1} &= 0, \\ Y_k &= \text{hyst}(e_k), \quad k = 0, 1, 2, \dots \end{aligned}$$

where

$$e_k = u1_k - u2_k$$

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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

hys	Hysteresis	↓0.0 ∅0.5 Double (F64)
------------	------------	------------------------

Output

Y	Logical output of the block	Bool
----------	-----------------------------	------

CNDR – Nonlinear conditioner

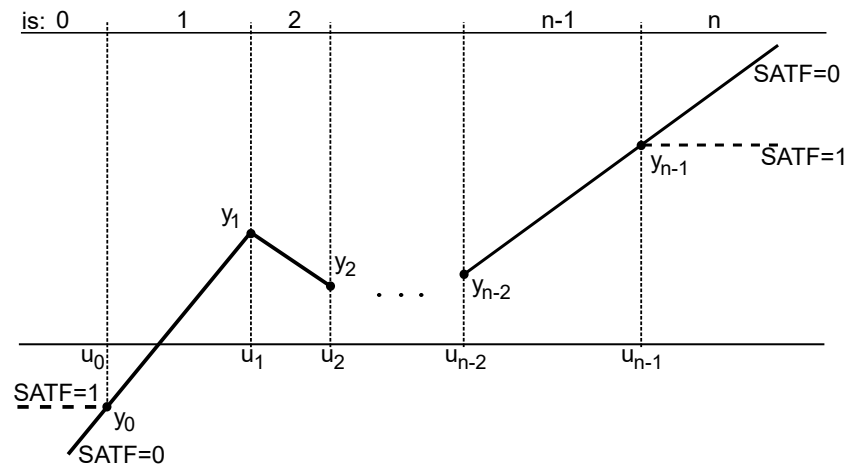
Block Symbol

Licence: [STANDARD](#)



Function Description

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



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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u Analog input of the block Double (F64)

Parameter

nmax	Allocated size of array	↓4 ⊙10	Long (I32)
SATF	Saturation flag	⊙on	Bool
	off ... Signal not limited		
	on Saturation limits active		
up	Vector of increasing u-coordinates		Double (F64)
	⊙[0.0 3.9 3.9 9.0 14.5 20.0]		

yp	Vector of y-coordinates	⊙[0.0 0.0 15.8 38.4 72.0 115.0]	Double (F64)
----	-------------------------	---------------------------------	--------------

Output

y	Analog output of the block	Double (F64)
is	Active sector of nonlinearity	Long (I32)

DEL – Delay with initialization

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.

$$y_k = u_{k-n}.$$

The corresponding time delay is $n \cdot T_S$, where T_S is the block trigger period.

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 $RDY = \text{off}$.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

n	Delay [samples]	↓0 ↑100000000 ⊙10	Long (I32)
n_{\max}	Allocated size of array	↓10 ↑100000000 ⊙100	Long (I32)

Output

y	Delayed input signal	Double (F64)
RDY	Outputs valid (ready flag)	Bool

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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

del	Time delay [s]	⊙1.0 Double (F64)
nmax	Allocated size of array	↓10 ↑10000000 ⊙100 Long (I32)

Output

y	Delayed input signal	Double (F64)
----------	----------------------	--------------

DER – Derivation, filtering and prediction from the last $n+1$ samples

Block Symbol

Licence: [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 **RUN = on** the outputs y and z are computed from the obtained parameters a and b of the linear interpolation as follows:

$$\begin{aligned} \text{Derivation:} \quad y &= a \\ \text{Filtering:} \quad z &= b, \text{ for } t_p = 0 \\ \text{Prediction:} \quad z &= at_p + b, \text{ for } t_p > 0 \\ \text{Retrodiction:} \quad z &= at_p + b, \text{ for } t_p < 0 \end{aligned}$$

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

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	Analog output of the block	Double (F64)
RUN	Enable execution	Bool
	off ... Tracking	
	on Filtering	
tp	Time instant for prediction/filtering	Double (F64)

Parameter

n	Number of samples for interpolation	↓1 ↑10000000 ⊙10	Long (I32)
nmax	Allocated size of array	↓10 ↑10000000 ⊙100	Long (I32)

Output

y	Estimate of input signal derivative	Double (F64)
z	Predicted/filtered input signal	Double (F64)

RDY	Outputs valid (ready flag)	Bool
-----	----------------------------	------

EVAR – Moving mean value and standard deviation

Block Symbol

Licence: [STANDARD](#)



Function Description

The **EVAR** block estimates the mean value **mu** (μ) and standard deviation **si** (σ) 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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

n	Number of samples for statistics	↓2 ↑10000000 ⊙100	Long (I32)
nmax	Allocated size of array	↓10 ↑10000000 ⊙200	Long (I32)

Output

mu	Mean value	Double (F64)
si	Standard deviation	Double (F64)

INTE – Controlled integrator

Block Symbol

Licence: [STANDARD](#)



Function Description

The **INTE** block implements a controlled integrator with variable integral time constant **ti** and two indicators of the output signal level (**ymin** a **ymax**). If **RUN** = **on** and **R1** = **off** then

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

where $C = y0$. If **RUN** = **off** and **R1** = **off** then the output **y** is frozen to the last value before the falling edge at the **RUN** input signal. If **R1** = **on** then the output **y** is set to the initial value **y0**. 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. If $T_i = 0$, the block realize summation by following equation

$$y_k = y_{k-1} + u_k.$$

If $T_i < 0$, the block behaviour is undefined.

Consider using the [SINT](#) block, whose simpler structure and functionality might be sufficient for elementary tasks.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	Analog input of the block	Double (F64)
RUN	Enable execution	Bool
	off ... Integration stopped	
	on ... Integration running	
R1	Block reset	Bool
y0	Initial output value	Double (F64)
ti	Integral time constant	Double (F64)

Parameter

ymin	Lower level definition	⊖-1.0	Double (F64)
ymax	Upper level definition	⊕1.0	Double (F64)
SAT	Limit output if level limit is reach		Bool

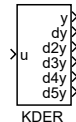
Output

y	Integrator output		Double (F64)
Q	Running integration indicator		Bool
LY	Lower saturation indicator		Bool
HY	Upper saturation indicator		Bool

KDER – Derivation and filtering of the input signal

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **KDER** block is a Kalman-type filter of the **norder**-th order aimed at estimation of derivatives of locally polynomial signals corrupted by noise. The order of derivatives ranges from 0 to **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 **pbeta** and **norder**. The **pbeta** 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 $\text{pbeta} \approx T_S \omega_0$ can be used. The frequency spectrum of the input signal **u** should be located deep down below the cutoff frequency ω_0 . But at the same time, the frequency spectrum of the noise should be as far away from the cutoff frequency ω_0 as possible. The cutoff frequency ω_0 and thus also the **pbeta** parameter must be lowered for strengthening the noise rejection.

The other parameter **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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

norder	Order of the derivative filter	↓2 ↑10 ⊙3 Long (I32)
pbeta	Bandwidth of the derivative filter	↓0.0 ⊙0.1 Double (F64)

Output

y	Filtered input signal	Double (F64)
dy	Estimated 1st order derivative	Double (F64)

d2y	Estimated 2nd order derivative	Double (F64)
d3y	Estimated 3rd order derivative	Double (F64)
d4y	Estimated 4th order derivative	Double (F64)
d5y	Estimated 5th order derivative	Double (F64)

LPF – Low-pass filter

Block Symbol

Licence: [STANDARD](#)



Function Description

The LPF block implements a second order filter in the form

$$F_s = \frac{1}{a^2 s^2 + 2\xi a s + 1},$$

where

$$a = \frac{\sqrt{\sqrt{2}\sqrt{2\xi^4 - 2\xi^2 + 1} - 2\xi^2 + 1}}{2\pi f_b}$$

and **fb** and $\xi = \mathbf{xi}$ are the block parameters. The **fb** [Hz] parameter defines the filter bandwidth and **xi** is the relative damping coefficient. Attenuation at frequency **fb** is 3 dB, at $10 \cdot \mathbf{fb}$ approximately 40 dB. For the correct function of the filter, $f_b < \frac{1}{10T_S}$ must hold, where T_S is the block triggering period. 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**.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	Input signal to be filtered	Double (F64)
R1	Block reset	Bool
HLD	Hold	Bool

Parameter

fb	Filter bandwidth [Hz]	⊙1.0	Double (F64)
xi	Relative damping coefficient	⊙0.707	Double (F64)
ISSF	Steady state at start-up		Bool
	off ... Zero initial state		
	on Initial steady state		

Output

y

Filtered output signal

Double (F64)

MINMAX – Running minimum and maximum

Block Symbol

Licence: [STANDARD](#)



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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	Analog input of the block	Double (F64)
R1	Block reset	Bool

Parameter

n	Number of samples for analysis	↓1 ↑100000000 ⊙100	Long (I32)
nmax	Allocated size of array	↓10 ↑100000000 ⊙200	Long (I32)

Output

ymin	Minimal value found	Double (F64)
ymax	Maximal value found	Double (F64)
RDY	Outputs valid (ready flag)	Bool

NSCL – Nonlinear scaling factor

Block Symbol

Licence: [STANDARD](#)

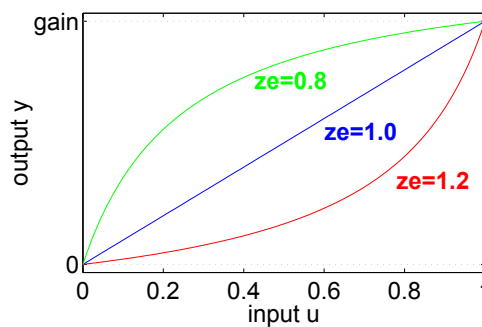


Function Description

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



This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

gain	Signal gain	⊙1.0	Double (F64)
ze	Shaping parameter	⊙1.0	Double (F64)

Output

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

OSD – One Step Delay

Block Symbol

Licence: [STANDARD](#)



Function Description

The **OSD** block implements a one step delay of the input signal *u*. The length of the step delay (in seconds) is given by the task period (see the [EXEC](#) function block description for details).

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

<i>u</i>	Analog input of the block	Any
----------	---------------------------	-----

Parameter

LB	Act as loopbreak	Bool
----	------------------	------

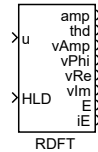
Output

<i>y</i>	Analog output of the block	Any
----------	----------------------------	-----

RDFT – Running discrete Fourier transform

Block Symbol

Licence: [ADVANCED](#)



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 $m = \text{nper}/\text{freq}/T_S$, i.e. from the time-window of the length equivalent to **nper** periods of the fundamental frequency.

If **nharm** > 0 the number of monitored higher harmonic frequencies is given solely by this parameter. On the contrary, for **nharm** = 0 the monitored frequencies are given by the user-defined vector parameter **freq2**.

For each frequency the amplitude (**vAmp** output), phase-shift (**vPhi** output), real/cosine part (**vRe** output) and imaginary/sine part (**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. The output **thd** indicates the total harmonic distortion, i.e. the part of fundamental and higher harmonic frequencies (only if **nharm** ≥ 1).

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	Analog input of the block	Double (F64)
HLD	Hold	Bool

Parameter

freq	Fundamental frequency	↓1e-09 ↑1e+09 ⊙1.0	Double (F64)
nper	Number of periods to calculate upon	↓1 ↑10000 ⊙10	Long (I32)
nharm	Number of monitored harmonic frequencies	↓0 ↑16 ⊙3	Long (I32)
ifrunit	Frequency units	⊙1	Long (I32)
	1 Hz		
	2 rad/s		

<code>iphunit</code>	Phase shift units 1 degrees 2 radians	⊙1	Long (I32)
<code>nmax</code>	Allocated size of array	↓10 ↑10000000	⊙8192 Long (I32)
<code>freq2</code>	Vector of user-defined monitored frequencies	⊙[2.0 3.0 4.0]	Double (F64)

Output

<code>amp</code>	Amplitude of the fundamental frequency	Double (F64)
<code>thd</code>	Total harmonic distortion	Double (F64)
<code>vAmp</code>	Vector of amplitudes at given frequencies	Reference
<code>vPhi</code>	Vector of phase-shifts at given frequencies	Reference
<code>vRe</code>	Vector of real parts at given frequencies	Reference
<code>vIm</code>	Vector of imaginary parts at given frequencies	Reference
<code>E</code>	Error indicator	Bool
<code>iE</code>	Error code	Error

RLIM – Rate limiter

Block Symbol

Licence: [STANDARD](#)**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 **tp** and **tn**:

$$\begin{aligned} \text{the steepest rise per second:} & \quad 1/\mathbf{tp} \\ \text{the steepest descent per second:} & \quad -1/\mathbf{tn} \end{aligned}$$

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

tp	Time constant - maximum rise	⊙2.0 Double (F64)
tn	Time constant - maximum descent	⊙2.0 Double (F64)

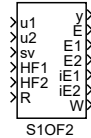
Output

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

S10F2 – One of two analog signals selector

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **S10F2** block assesses the validity of two input signals **u1** and **u2** separately. The validation method is equal to the method used in the **SAI** block. If the signal **u1** (or **u2**) is marked invalid, the output **E1** (or **E2**) is set to **on** and the error code is sent to the **iE1** (or **iE2**) output. The **S10F2** block also evaluates the difference between the two input signals. The internal flag **D** is set to **on** if the differences $|u1 - u2|$ in the last **nd** samples exceed the given limit, which is given by the following inequation:

$$|u1 - u2| > pdev \frac{vmax - vmin}{100},$$

where **vmin** and **vmax** are the minimal and maximal limits of the inputs **u1** and **u2** and **pdev** 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 **E1** and **E2**) and the internal difference flag **D** as follows:

(i) If **E1 = off** and **E2 = off** and **D = off** , then the output **y** depends on the mode parameter:

$$y = \begin{cases} \frac{u1+u2}{2}, & \text{for } codemode = 1, \\ \min(u1, u2), & \text{for } mode = 2, \\ \max(u1, u2), & \text{for } mode = 3. \end{cases}$$

and the output **E** is set to **off** unless set to **on** earlier.

(ii) If **E1 = off** and **E2 = off** and **D = on** , then **y = sv** and **E = on**.

(iii) If **E1 = on** and **E2 = off** (**E1 = off** and **E2 = on**) , then **y = u2** (**y = u1**) and the output **E** is set to **off** unless set to **on** earlier.

(iv) If **E1 = on** and **E2 = 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 **E1** (**E2**) is set to **on** for only one cycle period whenever some invalidity condition is fulfilled. On the other hand, for **R = 0**, the output **E1** (**E2**) is set to **on** and remains true until the reset (**R: off**→**on**). A

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.

The parameter **nb** specifies the number of samples after restart during which signal validity detection for **u1** and **u2** is suppressed. The parameter **nc** indicates the number of samples for testing immutability (see the **SAI** block, condition **F2**). The number of samples for testing variability (see the **SAI** block, condition **F3**) is given by the parameter **nr**. The maximum expected percentage change in input **u1** (**u2**) from the total range **vmax** – **vmin** over **nr** samples of input **u1** (**u2**) (see the **SAI** block) is determined by **prate**. The parameter **nv** represents the number of samples for testing range exceedance (see the **SAI** block, condition **F4**).

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

u1	First analog input of the block	Double (F64)
u2	Second analog input of the block	Double (F64)
sv	Substitute value for an error case	Double (F64)
HF1	Hardware error flag for signal u1	Bool
	off ... The input module of the signal works normally	
	on Hardware error of the input module occurred	
HF2	Hardware error flag for signal u2	Bool
	off ... The input module of the signal works normally	
	on Hardware error of the input module occurred	
R	Reset inner error flags	Bool

Parameter

nb	Number of samples to skip at startup	⊙10	Long (I32)
nc	Number of samples for invariability testing	⊙10	Long (I32)
nbits	Number of A/D converter bits	⊙12	Long (I32)
nr	Number of samples for variability testing	⊙10	Long (I32)
prate	Maximum allowed percentage change	⊙10.0	Double (F64)
nv	Number of samples for out-of-range testing	⊙1	Long (I32)
vmin	Lower limit for the input signal	⊙-1.0	Double (F64)
vmax	Upper limit for the input signal	⊙1.0	Double (F64)
nd	Number of samples for deviation testing	⊙5	Long (I32)
pdev	Maximum allowed percentage deviation of inputs	⊙10.0	Double (F64)
mode	Computation of output when both inputs are valid	⊙1	Long (I32)
	1 Average		
	2 Minimum		
	3 Maximum		

Output

y	Analog output of the block	Double (F64)
E	Output signal invalidity indicator off ... Signal is valid on Signal is invalid	Bool
E1	Invalidity indicator for input u1 off ... Signal is valid on Signal is invalid	Bool
E2	Invalidity indicator for input u2 off ... Signal is valid on Signal is invalid	Bool
iE1	Reason of input u1 invalidity 0 Signal valid 1 Signal out of range 2 Signal varies too much 3 Signal varies too much and signal out of range 4 Signal varies too little 5 Signal varies too little and signal out of range 6 Signal varies too much and too little 7 Signal varies too much and too little and signal out of range 8 Hardware error	Long (I32)
iE2	Reason of input u2 invalidity 0 Signal valid 1 Signal out of range 2 Signal varies too much 3 Signal varies too much and signal out of range 4 Signal varies too little 5 Signal varies too little and signal out of range 6 Signal varies too much and too little 7 Signal varies too much and too little and signal out of range 8 Hardware error	Long (I32)
W	Warning flag (invalid input signal) off ... Both input signals are valid on At least one of the input signals is invalid	Bool

SAI – Safety analog input

Block Symbol

Licence: [ADVANCED](#)



Function Description

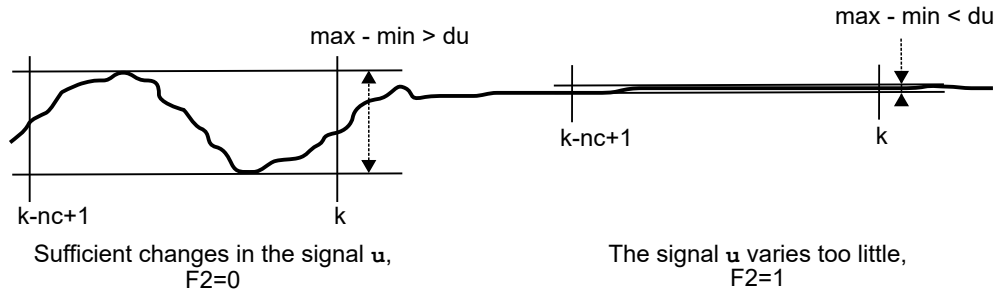
The **SAI** block tests the input signal **u** and assesses its validity. The input signal **u** is considered invalid (the output **E** = **on**) in the following cases:

F1: Hardware error. The input signal **HWF** = **on**.

F2: The input signal **u** varies too little. The last **nc** samples of the input **u** lies within the interval of width **du**,

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

where **vmin** and **vmax** are the lower and upper limits of the input **u**, respectively, and **nbits** is the number of A/D converter bits. The situation when the input signal **u** varies too little is shown in the following picture:



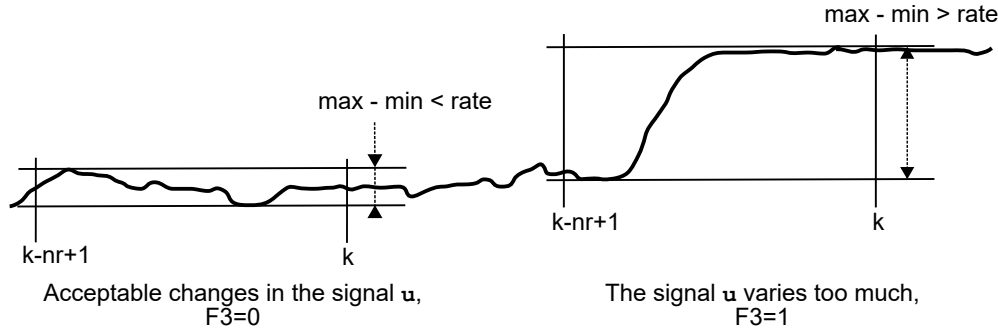
If the parameter **nc** is set to **nc** = 0, the condition **F2** is never fulfilled.

F3: The input signal **u** varies too much. The last **nr** samples of the input **u** filtered by the [SPIKE](#) filter have a span which is greater than **rate**,

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

where **prate** defines the allowed percentage change in the input signal **u** within the last **nr** samples (with respect to the overall range of the input signal $u \in \langle v_{\min}, v_{\max} \rangle$). The block includes a [SPIKE](#) filter with fixed parameters $\text{mingap} = \frac{v_{\max} - v_{\min}}{100}$ 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 $\langle v_{min}, v_{max} \rangle$. 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 **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 **on**, the invalidity indicator E is set to **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 **on** and remains true until the reset (rising edge R : **off**→**on**).

The table of error codes iE resulting from the inner error flags $F1$ – $F4$:

F1	F2	F3	F4	iE
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	*	*	*	8

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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

u	Analog input of the block	Double (F64)
sv	Substitute value for an error case	Double (F64)
HWF	Hardware error indicator	Bool
	off ... The input module of the signal works normally	
	on Hardware error of the input module occurred	
R	Reset inner error flags	Bool

Parameter

nb	Number of samples to skip at startup	⊙10	Long (I32)
nc	Number of samples for invariability testing	⊙10	Long (I32)
nbits	Number of A/D converter bits	⊙12	Long (I32)
nr	Number of samples for variability testing	⊙10	Long (I32)
prate	Maximum allowed percentage change	⊙10.0	Double (F64)
nv	Number of samples for out-of-range testing	⊙1	Long (I32)
vmin	Lower limit for the input signal	⊙-1.0	Double (F64)
vmax	Upper limit for the input signal	⊙1.0	Double (F64)

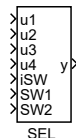
Output

y	Analog output of the block	Double (F64)
yf	Filtered output signal (SPIKE)	Double (F64)
E	Output signal invalidity indicator	Bool
	off ... Signal is valid	
	on Signal is invalid	
iE	Reason of invalidity	Long (I32)
	0 Signal valid	
	1 Signal out of range	
	2 Signal varies too much	
	3 Signal varies too much and signal out of range	
	4 Signal varies too little	
	5 Signal varies too little and signal out of range	
	6 Signal varies too much and too little	
	7 Signal varies too much and too little and signal out of range	
	8 Hardware error	

SEL – Selector switch for analog signals

Block Symbol

Licence: [STANDARD](#)



Function Description

The **SEL** block is obsolete, replace it by the [SELQUAD](#), [SELOCT](#) or [SELHEXD](#) 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:

iSW	SW1	SW2	y
0	off	off	u1
1	off	on	u2
2	on	off	u3
3	on	on	u4

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<i>u1</i> .. <i>u4</i>	Analog input of the block	Double (F64)
<i>iSW</i>	Active signal selector	Long (I32)
<i>SW1</i>	Binary signal selector	Bool
<i>SW2</i>	Binary signal selector	Bool

Parameter

<i>BINF</i>	Enable the binary selectors	Bool
	off ... Disabled (analog selector)	
	on Enabled (binary selectors)	

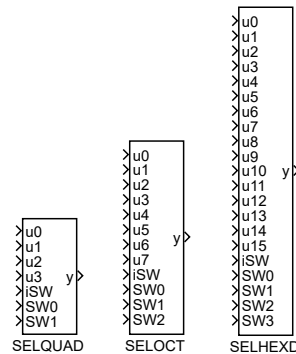
Output

<i>y</i>	The selected input signal	Double (F64)
----------	---------------------------	--------------

SELQUAD, SELOCT, SELHEXD – Selector switch for analog signals

Block Symbols

Licence: [STANDARD](#)



Function Description

The **SELQUAD**, **SELOCT** and **SELHEX** blocks select one of the input signals and copy it to the output signal **y**. Please note that the only difference among the blocks is the number of inputs. 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:

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

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u0..u15	Analog input of the block	Any
iSW	Active signal selector	Long (I32)
SW0..SW3	Binary signal selector	Bool

Parameter

BINF	Enable the binary selectors	Bool
------	-----------------------------	------

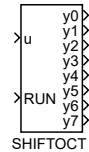
Output

y	The selected input signal	Any
---	---------------------------	-----

SHIFTOCT – Data shift register

Block Symbol

Licence: [STANDARD](#)



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:

$$\begin{aligned} y_i &= y_{i-1}, \quad i = 1..7 \\ y_0 &= u \end{aligned}$$

Thus the value on each output **y0** to **y6** is shifted to the following output and the value on input **u** is assigned to output **y0**.

The block works with any data type of signal connected to the input **u**. Data type has to be specified by the **vtype** parameter. Outputs **y0** to **y7** then have the same data type.

If you need a triggered shift register, place the [EDGE](#) block in front of the **RUN** input.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	Analog input of the block	Any
RUN	Enables outputs shift	Bool

Parameter

vtype	Output data type	⊙8 Long (I32)
1 Bool	
2 Byte (U8)	
3 Short (I16)	
4 Long (I32)	
5 Word (U16)	
6 DWord (U32)	
7 Float (F32)	
8 Double (F64)	
10 Large (I64)	

Output

y0..y7	Analog output of the block	Any
---------------	----------------------------	-----

SHLD – Sample and hold

Block Symbol

Licence: [STANDARD](#)



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 **y0**, it overpowers the **SETH** input signal.

See also the [PARR](#) block, which can be used for storing a numeric value as well.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	Analog input of the block	Double (F64)
SETH	Set and hold the output signal	Bool
R1	Block reset	Bool

Parameter

y0	Initial output value	Double (F64)
mode	Sampling mode	⊙3 Long (I32)
	1 Triggered sampling	
	2 Hold last value	
	3 Hold current value	

Output

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

SINT – Simple integrator

Block 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 $T_i = 0$, the block realize summation by following equation

$$y_k = y_{k-1} + u_k.$$

If $T_i < 0$, the block behaviour is undefined.

If y_k falls out of the saturation limits **ymin** and **ymax**, the output and state of the block are appropriately modified.

For more complex tasks, consider using the [INTE](#) block, which provides extended functionality.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

ti	Integral time constant	⊙1.0	Double (F64)
y0	Initial output value		Double (F64)
ymax	Upper limit of the output signal	⊙1.0	Double (F64)
ymin	Lower limit of the output signal	⊙-1.0	Double (F64)

Output

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

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;
if abs(delta) < gap
  then
    begin
      y := u;
      gap := gap/q;
      if gap < mingap then gap:= mingap;
    end
  else
    begin
      if delta < 0
        then y := y + gap
        else y := y - gap;
      gap := gap * q;
    end
  end

```

where **mingap** and **q** are the block parameters.

The signal passes through the filter unaffected for sufficiently large **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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

mingap	Minimum size of the tolerance window	⊙0.01	Double (F64)
q	Tolerance window adaptation speed	↓1.0 ⊙2.0	Double (F64)

Output

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

SSW – Simple switch

Block Symbol

Licence: [STANDARD](#)



Function Description

The **SSW** block selects one of two input signals **u1** and **u2** with respect to the binary input **SW**. The selected input is copied to the output **y**. If **SW = off** (**SW = on**), then the selected signal is **u1** (**u2**).

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u1	First analog input of the block	Any
u2	Second analog input of the block	Any
SW	Signal selector	Bool
	off ... The u1 signal is selected	
	on The u2 signal is selected	

Output

y	Analog output of the block	Any
----------	----------------------------	------------

SWR – Selector with ramp

Block Symbol

Licence: [STANDARD](#)



Function Description

The **SWR** block selects one of two input signals **u1** and **u2** with respect to the binary input **SW**. The selected input is copied to the output **y**. If **SW = off** (**SW = on**), then the selected signal is **u1** (**u2**). 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 **u1**, **u2** and is defined by time constants **t1** and **t2**. 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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u1	First analog input of the block	Double (F64)
u2	Second analog input of the block	Double (F64)
SW	Signal selector	Bool
	off ... The u1 signal is selected	
	on The u2 signal is selected	

Parameter

t1	Rate limiter time constant, u2 -> u1	⊙1.0 Double (F64)
t2	Rate limiter time constant, u1 -> u2	⊙1.0 Double (F64)
y0	Initial output value	Double (F64)

Output

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

VDEL – Variable time delay

Block 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 **y0** is used until n previous samples are available after the block initialization.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	Analog input of the block	Double (F64)
d	Time delay [s]	Double (F64)

Parameter

y0	Initial output value	Double (F64)
nmax	Allocated size of array	↓10 ↑100000000 ⊕1000 Long (I32)

Output

y	Delayed input signal	Double (F64)
----------	----------------------	--------------

ZV4IS – Zero vibration input shaper

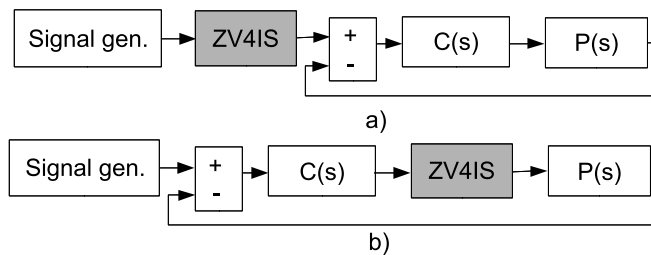
Block Symbol

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 inferior 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.

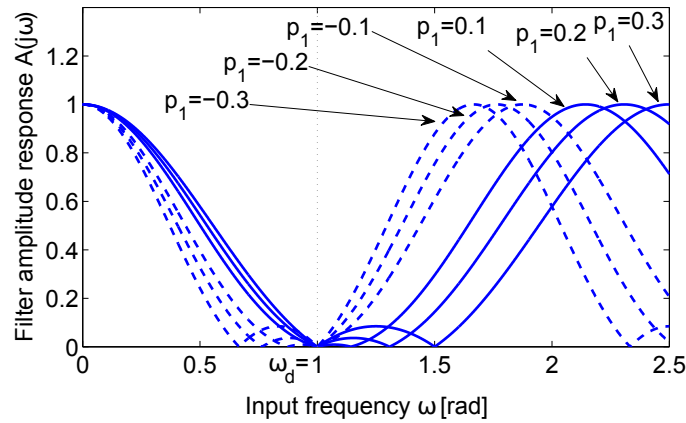
The algorithm of input shaper can be described in time domain

$$y(t) = A_1 u(t - t_1) + A_2 u(t - t_2) + A_3 u(t - t_3) + A_4 u(t - t_4)$$

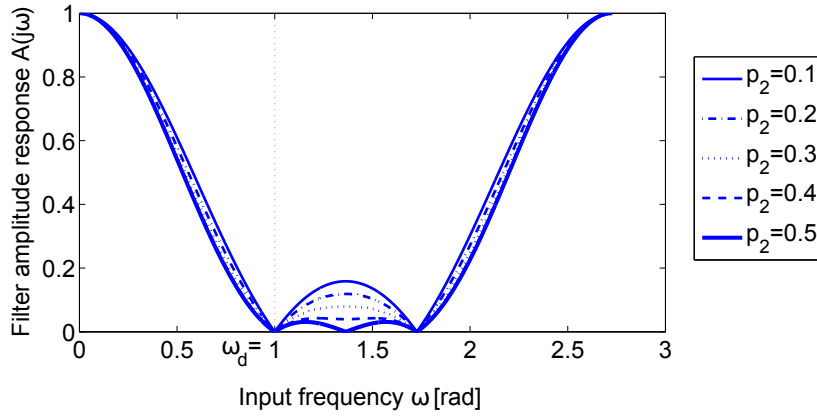
Thus, the filter has a structure of sum of weighted time delays of an input signal. The gains $A_1..A_4$ and time delay values $t_1..t_4$ depend on a choice of filter type, natural frequency and damping of controlled oscillatory mode of the system. The main advantage of this structure compared to commonly 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 ω and damping ξ of the oscillatory mode need to be set. The parameter ipar sets a filter type. For $\text{ipar} = 1$, one of ten basic filter types chosen by istype 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 ω and ξ (one, two or five percent).

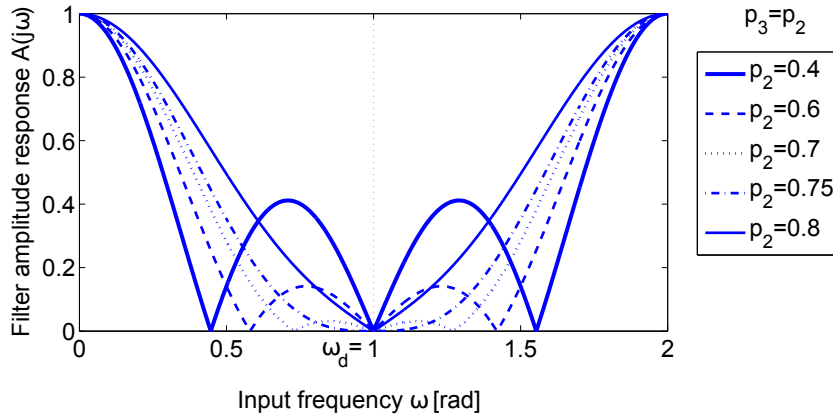
For precise tuning of the filter, complete parameterization $\text{ipar} = 2$ can be selected. For this choice, three parameters p_alpha , p_a2 and p_a3 which affect the shape of the filter frequency response can freely be assigned. These parameters can be used for finding of optimal compromise between 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 ($p_{\alpha} = 0$). 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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

omega	Natural frequency	⊙1.0	Double (F64)
xi	Relative damping coefficient		Double (F64)
ipar	Specification	⊙1	Long (I32)
	1 Basic types of IS		
	2 Complete parametrization		
istype	Type	⊙2	Long (I32)
	1 ZV		
	2 ZVD		
	3 ZVDD		
	4 MISZV		
	5 UEI1		
	6 UEI2		
	7 UEI5		
	8 UTHEI1		
	9 UTHEI2		
	10 UTHEI5		
p_alpha	Shaper duration/assymetry parameter	⊙0.2	Double (F64)
p_a2	Insensitivity parameter	⊙0.5	Double (F64)
p_a3	Additional parameter (only for p_alpha=0)	⊙0.5	Double (F64)
nmax	Allocated size of array	↓10 ↑100000000 ⊙1000	Long (I32)

Output

y	Filtered output signal	Double (F64)
E	Error indicator	Bool
	off ... No error	
	on An error occurred	

Chapter 6

GEN – Signal generators

Contents

ANLS – Controlled generator of piecewise linear function	190
BINS – Controlled binary sequence generator	192
BIS – Binary sequence generator	194
BISR – Binary sequence generator with reset	196
MP – Manual pulse generator	198
PRBS – Pseudo-random binary sequence generator	199
SG, SGI – Signal generators	201

The GEN library is specialized in signal generation. It includes blocks like [ANLS](#) for generating a piecewise linear function of time or binary sequence generators [BINS](#), [BIS](#), [BISR](#). The library also features [MP](#) for manual pulse signal generation, [PRBS](#) for pseudo-random binary sequence generation, and [SG](#) for periodic signals generation. This library provides essential tools for creating and manipulating various signal types.

ANLS – Controlled generator of piecewise linear function

Block Symbol

Licence: [STANDARD](#)



Function Description

The ANLS block generates a piecewise linear function of time given by nodes **t1,y1**; **t2,y2**; **t3,y3**; **t4,y4**. The initial value of output **y** is defined by the **y0** 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 $\langle t_i, t_{i+1} \rangle, i = 0, \dots, 3, t_0 = 0$.

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 **t4** 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 **y0**, 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 **y0**, the internal block timer is set to 0 and **is** = 0 becomes the active time interval.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

RUN	Enable execution	Bool
------------	------------------	-------------

Parameter

y0	Initial output value		Double (F64)
t1	Node 1 time	⊙1.0	Double (F64)
y1	Node 1 value		Double (F64)
t2	Node 2 time	⊙1.0	Double (F64)
y2	Node 2 value	⊙1.0	Double (F64)
t3	Node 3 time	⊙2.0	Double (F64)
y3	Node 3 value	⊙1.0	Double (F64)

t4	Node 4 time	⊙2.0	Double (F64)
y4	Node 4 value		Double (F64)
RPT	Repeating sequence		Bool
	off ... Disabled		
	on Enabled		

Output

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

BINS – Controlled binary sequence generator

Block Symbol

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, 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 ($< T_S/2$) 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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

START	Starting signal (rising edge)	Bool
--------------	-------------------------------	-------------

Parameter

Y0	Initial output value	Bool
	off ... Disabled/false	
	on Enabled/true	
t1..t8	Switching time [s]	↓0.0 ⊙1.0 Double (F64)
RPT	Repeating sequence	Bool
	off ... Disabled	
	on Enabled	

Output

Y	Logical output of the block	Bool
is	Index of the active time interval	Long (I32)

BIS – Binary sequence generator

Block Symbol

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 **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 indefinitely.
- 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.

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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Parameter

Y0	Initial output value off ... Disabled/false on Enabled/true		Bool
t1 ... t8	Switching time [s]	↓0.0 ⊙1.0	Double (F64)
RPT	Repeating sequence off ... Disabled on Enabled		Bool

Output

y	Logical output of the block
is	Index of the active time interval

Bool
Long (I32)

BISR – Binary sequence generator with reset

Block Symbol

Licence: [STANDARD](#)



Function Description

The **BISR** block generates a binary sequence at the **Y** output. The **RUN** input must be set to **on** for the whole duration of the sequence. When **RUN** is **off**, the sequence is paused and so is the internal timer.

The binary sequence is given by the block parameters. The initial value of the output is given by the **Y0** parameter. The output value **Y** is inverted (**off**→**on**, **on**→**off**) at time instants **t1**, **t2**, ..., **t8**. The **ADDT** parameter defines whether the **t_i** instants are relative to the first rising edge at the **RUN** input or relative to the last switching of the **Y** output.

If there is less than 8 edges in the desired binary sequence, set any of the **t_i** parameters to zero and the remaining ones will be ignored.

Whenever a rising edge occurs at the **R1** input, the output **Y** is set to **Y0** and the internal timer is reset. The **R1** input overpowers the **RUN** input.

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.

The **BISR** block is an extended version of the [BINS](#) block.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

RUN	Enable execution	Bool
R1	Block reset	Bool

Parameter

Y0	Initial output value off ... Disabled/false on Enabled/true	Bool
ADDT	Additive timing off ... Absolute timing (sequence as a whole) on Additive timing (segment by segment)	Bool

RPT	Repeating sequence off ... Disabled on Enabled	Bool
t1..t8	Switching time [s]	$\downarrow 0.0 \odot 1.0$ Double (F64)

Output

Y	Logical output of the block	Bool
is	Index of the active time interval	Long (I32)

MP – Manual pulse generator

Block Symbol

Licence: [STANDARD](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Parameter

<code>BSTATE</code>	Output pulse activation		<code>Bool</code>
	<code>off</code> ... No action		
	<code>on</code> Generate output pulse		
<code>pwidth</code>	Pulse width [s] (0 = one tick)	⊙1.0	<code>Double (F64)</code>
<code>RPTF</code>	Allow pulse extension		<code>Bool</code>
	<code>off</code> ... Disabled		
	<code>on</code> Enabled		

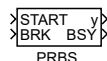
Output

<code>Y</code>	Logical output of the block	<code>Bool</code>
----------------	-----------------------------	-------------------

PRBS – Pseudo-random binary sequence generator

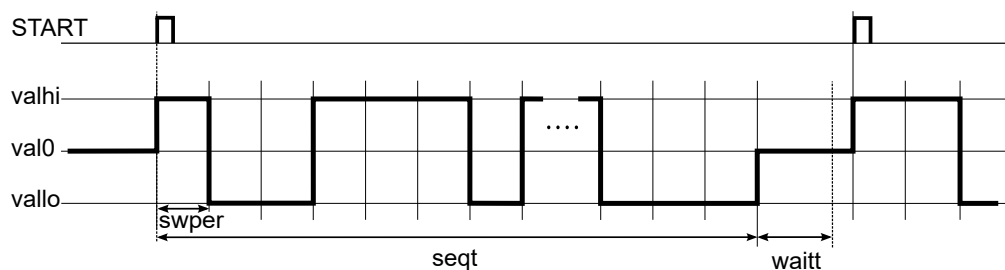
Block Symbol

Licence: [STANDARD](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>START</code>	Starting signal (rising edge)	Bool
<code>BRK</code>	Termination signal	Bool

Parameter

<code>val0</code>	Initial and final value	Double (F64)
<code>valhi</code>	Upper level of the <code>y</code> output	⊙1.0 Double (F64)
<code>vallo</code>	Lower level of the <code>y</code> output	⊙-1.0 Double (F64)
<code>swper</code>	Period of random output switching [s]	⊙1.0 Double (F64)
<code>swprob</code>	Probability of switching	↓0.0 ↑1.0 ⊙0.2 Double (F64)

<code>seqt</code>	Length of the sequence [s]	⊙10.0	Double (F64)
<code>waitt</code>	Settling period [s]	⊙2.0	Double (F64)

Output

<code>y</code>	Generated pseudo-random binary sequence	Double (F64)
<code>BSY</code>	Busy flag	Bool

SG, SGI – Signal generators

Block Symbols

Licence: [STANDARD](#)



Function Description

The **SG** and **SGI** blocks generate periodic functions according to the setting of the **isig** parameter, which determines the type of signal: sine wave, square wave (with a duty cycle of 1), sawtooth signal, random signal (white noise with uniform distribution) or triangle signal. The amplitude and frequency of the output signal **y** can be set using the **amp** and **freq** parameters. The frequency can be specified in units of **Hz** or **rad/s**, as determined by the **ifrunit** parameter. For sine, square, sawtooth and triangle signals (i.e. **isig** $\in \{1, 2, 3, 5\}$), a phase shift can be adjusted, which is set by the **phase** parameter within the range $(0, 2\pi)$. The unit of phase shift (radians or degrees) is determined by the **iphunit** parameter.

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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

RUN	Enable execution	Bool
SYN	Synchronization signal	Bool

Parameter

isig	Generated signal type	⊙1	Long (I32)
	1 SINE		
	2 SQUARE		
	3 SAWTOOTH		
	4 RANDOM		
	5 TRIANGLE		
amp	Amplitude	⊙1.0	Double (F64)
freq	Frequency	⊙1.0	Double (F64)
phase	Phase shift		Double (F64)
offset	Value added to output		Double (F64)

ifrunit	Frequency units	⊙1	Long (I32)
	1 Hz		
iphunit	2 rad/s	⊙1	Long (I32)
	Phase shift units		
	1 degrees		
	2 radians		

Output

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

Chapter 7

REG – Function blocks for control

Contents

ARLY – Advance relay	205
FIWR – Frequence Identification With Reconstructor	206
FLCU – Fuzzy logic controller unit	209
FRID – Frequency response identification	211
I3PM – Identification of a three parameter model	215
LC – Lead compensator	217
LLC – Lead-lag compensator	218
LPI – Loop performance index	219
MCU – Manual control unit	221
PIDAT – PID controller with relay autotuner	223
PIDE – PID controller with defined static error	226
PIDGS – PID controller with gain scheduling	228
PIDMA – PID controller with moment autotuner	230
PIDU – PID controller unit	237
PIDUI – PID controller unit with variable parameters	241
POUT – Pulse output	243
PRGM – Setpoint programmer	244
PSMPC – Pulse-step model predictive controller	246
PWM – Pulse width modulation	250
RLY – Relay with hysteresis	252
SAT – Saturation with variable limits	253
SC2FA – State controller for 2nd order system with frequency autotuner	255
SCU – Step controller with position feedback	262
SCUV – Step controller unit with velocity input	265
SELU – Controller selector unit	268
SMHCC – Sliding mode heating/cooling controller	270

SMHCCA – Sliding mode heating/cooling controller with autotuner	274
SWU – Switch unit	281
TSE – Three-state element	282

The control function blocks form the most extensive sub-library 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 sub-library. The involved controllers include the **PIDGS** 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.

ARLY – Advance relay

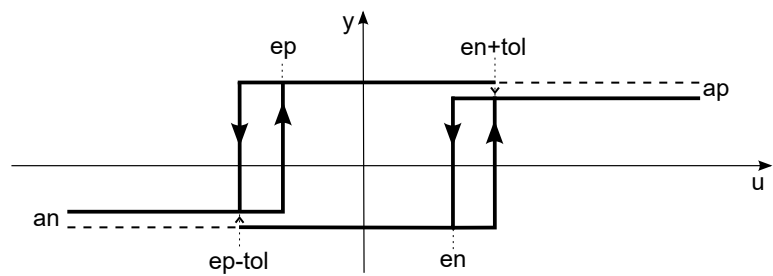
Block Symbol

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.



This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

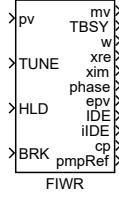
ep	Switch-on value	$\odot -1.0$	Double (F64)
en	Switch-off value	$\odot 1.0$	Double (F64)
tol	Tolerance limit	$\downarrow 0.0 \odot 0.5$	Double (F64)
ap	Output when ON	$\odot 1.0$	Double (F64)
an	Output when OFF	$\odot -1.0$	Double (F64)
$y0$	Initial output value		Double (F64)

Output

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

FIWR – Frequency Identification With Reconstructor

Block Symbol

Licence: [ADVANCED](#)

Function Description

The **FIWR** block performs the identification experiment of the process frequency characteristics. The system is excited by a harmonic signal with a static component **ubias**, amplitude **uampb** and frequency ω passing through the interval $\langle \mathbf{wb}, \mathbf{wf} \rangle$. If the adaptive amplitude change is enabled (**adaptive_amp** = **on**), then the excitation signal takes such values that the output amplitude is close to **dy_max**/2. The frequency sweep rate is determined by the **cp** state (unless it's not adaptively adjusted, in which case **cpb** exclusively determines it), outlining:

- For logarithmic mode (**mode** = 1), it signifies the proportional reduction of the initial period $T_b = \frac{2\pi}{w_b}$ of the excitation sine wave over time T_b . Hence, the equation is as follows:

$$cp = \frac{w_b}{\omega(T_b)} = \frac{w_b}{w_b e^{\gamma T_b}} = e^{-\gamma T_b}.$$

- For linear mode (**mode** = 2), it represents the frequency increase per unit of time.

The value of the **cp** state is usually in the interval $(0.7; 1)$ in logarithmic mode. For the linear mode, the parameter must be chosen with respect to the specified frequency range and the desired frequency change. If the adaptive change of the sweeping rate is enabled (**adaptive_cp** = **on**), then the **cp** state is recalculated depending on the "wrong" position of the frequency characteristic point before its convergence at each scheduled stop of the sweeping (given by the vector **PHI**). The current value of the **cp** parameter is copied to the output **cp**, its initial value **cpb** can be selected. The minimum and maximum values of **cp** **cp_min** and **cp_max** are taken into account only in **adaptive_cp** = **on** mode, otherwise the value **cpb** is copied to the output **cp**. In the combined mode (**mode** = 3), **cpb** is the sweeping rate in logarithmic segments and the linear segments including the sweeping rates are defined by the matrix **WL**. This matrix can be defined:

- **1: Explicitly:** $[w_{b1} \ w_{f1} \ cp_1, w_{b2} \ w_{f2} \ cp_2, \dots, w_{bN} \ w_{fN} \ cp_N]$, where the matrix size is $[1 \times 3N]$.

- **2: Alternatively:** $[wb_1 \ wb_2 \ \dots \ wb_N; wf_1 \ wf_2 \ \dots \ wf_N; cp_1 \ cp_2 \ \dots \ cp_N]$, where wb_i is the start of the i -th linear interval, wf_i is the end of the i -th linear interval. The matrix size is $[3 \times N]$.

The system identification is triggered by setting the input **TUNE** = **on**. After **rmi** seconds (estimated transient duration), the calculation of the current point of the frequency characteristic starts. Its real and imaginary parts are repeatedly copied to the outputs **xre** and **xim**, the phase delay at a given frequency is copied to the output **phase**. During the identification process, the sweeping stops whenever the phase delay points **PHI** are reached. The stopping time is related to the values of the parameters **q_crit** $\in (0.001; 0.1)$ and **np**. The smaller the parameter **q_crit** is, the more accurately the frequency characteristic point will be determined, and the sweeping stop time will be longer. For a shorter evaluation window given by the number of seconds **np** (usually chosen identically to **rmi**) the stopping time will be shorter. It is possible to select the maximum number of evaluation windows **cmi**, the common value is **cmi** = 10. By input **HLD** = **on** it is possible to manually stop the frequency sweeping, setting **HLD** = **off** again causes the sweeping to continue. If necessary, the identification process can be stopped by input **BRK** = **on**.

During the identification, the output **TBSY** = 1. It is set to 0 when finished. During an error-free experiment, the output **IDE** = **off**. If the identification ends in an error, then **IDE** = **on** and the output **iIDE** specifies the associated error.

The vector **PHI** can be inserted:

- **1: Explicitly:** $[\phi_1, \phi_2, \dots, \phi_M]$, where $\phi_i > \phi_{i+1}$ and the vector size is $[1 \times M]$.
- **2: Alternatively:** $[0, 0, \phi_{start}, \phi_{step}, \phi_{end}]$, where $\phi_{start} > \phi_{end}$, ϕ_{start} is the initial phase, ϕ_{step} is the step between phases, and ϕ_{end} is the final phase. The vector size is $[1 \times 5]$.

The **nmax** parameter specifies the maximum number of elements of the vector **PHI** that can be inserted. At the same time, this is the maximum width of the **pmpRef** matrix, thus the maximum number of accurately measured points that can be stored.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

pv	Process variable	Double (F64)
TUNE	Start the tuning experiment	Bool
HLD	Hold	Bool
BRK	Stop the tuning experiment	Bool

Parameter

wb	Start frequency [rad/s]	$\downarrow 0.0 \ \odot 1.0$ Double (F64)
-----------	-------------------------	---

wf	End frequency [rad/s]	↓0.0 ⊕10.0	Double (F64)
cpb	Initial sweeping rate	↓0.0 ↑1.0 ⊕0.92	Double (F64)
alpha	Relative position of reconstructor's poles	↓0.0 ⊕2.0	Double (F64)
ksi	Reconstructor damping	↓0.0 ⊕0.707	Double (F64)
mode	Frequency sweeping mode	⊕1	Long (I32)
	1 Logarithmic		
	2 Linear		
	3 Combined		
adaptive_cp	Adaptive sweep rate flag	⊕on	Bool
adaptive_amp	Adaptive amplitude flag	⊕on	Bool
np	Time for convergence detection [s]	↓0.0 ⊕3.0	Double (F64)
q_crit	Convergence threshold	↓0.0001 ↑0.2 ⊕0.05	Double (F64)
cp_min	Minimal Sweeping Rate	↓0.0 ↑1.0 ⊕0.8	Double (F64)
cp_max	Maximal Sweeping Rate	↓0.0 ↑1.0 ⊕0.99	Double (F64)
cp_ratio	Frequency and amplitude sweeping ratio	↓0.0 ↑1.0 ⊕0.8	Double (F64)
uampb	Initial amplitude of the exciting signal	↓0.0 ⊕1.0	Double (F64)
dy_max	Maximal change of amplitude of the process variable	↓0.0 ⊕1.0	Double (F64)
uamp_max	Maximal amplitude of the exciting signal	↓0.0 ⊕5.0	Double (F64)
ubias	Static component of the exciting signal		Double (F64)
hilim	Upper limit of the exciting signal	⊕10.0	Double (F64)
lolim	Lower limit of the exciting signal	⊕-10.0	Double (F64)
cmi	Maximum convergence detections per frequency	↓1 ⊕10	Long (I32)
rmi	Stabilization time at start [s]	↓0.0 ⊕10.0	Double (F64)
nmax	Allocated size of array	↓10 ↑10000000 ⊕40	Long (I32)
PHI	Phase measurement points [degrees]	⊕[-30 -90 -150]	Double (F64)
WL	Linear intervals matrix (combined mode only)	⊕[2.2	Double (F64)

Output

mv	Manipulated variable (controller output)	Double (F64)
TBSY	Tuner busy flag	Bool
	off ... Identification not running	
	on Identification in progress	
w	Actual frequency [rad/s]	Double (F64)
xre	Real part of frequency response	Double (F64)
xim	Imaginary part of frequency response	Double (F64)
phase	Phase shift of frequency response [degrees]	Double (F64)
epv	Estimated process value	Double (F64)
IDE	Error indicator	Bool
iIDE	Error code	Long (I32)
cp	Actual Sweeping Rate	Double (F64)
pmpRef	Precisely measured frequency response points (w im re)	⊕[2.2 Double (F64)

FLCU – Fuzzy logic controller unit

Block Symbol

Licence: [ADVANCED](#)



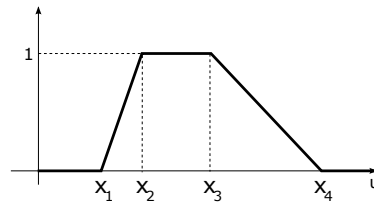
Function Description

The **FLCU** block implements a simple fuzzy logic controller with two inputs and one output. Introduction to fuzzy logic problems can be found in [4].

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

If (**u** is U_i) AND (**v** is V_j), then (**y** is Y_k),

where $U_i, i = 1, \dots, \mathbf{nu}$ are the linguistic terms of the **u** input; $V_j, j = 1, \dots, \mathbf{nv}$ are the linguistic terms of the **v** input and $Y_k, k = 1, \dots, \mathbf{ny}$ 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, \dots, 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 **mfu** and **mfv** are $(\mathbf{nu} \times 4)$ and $(\mathbf{nv} \times 4)$ 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 $Y_k, k = 1, \dots, \mathbf{ny}$ 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 **sty** has a dimension of $(\mathbf{ny} \times 3)$ and contains the rows $[y_k, a_k, b_k], k = 1, \dots, \mathbf{ny}$.

The set of inference rules is also a matrix whose rows are $[i_l, j_l, k_l, w_l], l = 1, \dots, \mathbf{nr}$, 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 $w_l \in \{0, 1, \dots, 100\}$. It is possible to suppress or emphasize a particular inference rule if necessary.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	First analog input of the block	Double (F64)
v	Second analog input of the block	Double (F64)

Parameter

umax	Upper limit of the u input	$\odot 1.0$	Double (F64)
umin	Lower limit of the u input	$\odot -1.0$	Double (F64)
vmax	Upper limit of the v input	$\odot 1.0$	Double (F64)
vmin	Lower limit of the v input	$\odot -1.0$	Double (F64)
nmax	Allocated size of array	$\downarrow 4 \uparrow 10000 \odot 10$	Long (I32)
mfu	Matrix of membership functions - input u		Double (F64)
		$\odot [-1 \ -1 \ -1 \ 0; \ -1 \ 0 \ 0 \ 1; \ 0 \ 1 \ 1 \ 1]$	
mfv	Matrix of membership functions - input v		Double (F64)
		$\odot [-1 \ -1 \ -1 \ 0; \ -1 \ 0 \ 0 \ 1; \ 0 \ 1 \ 1 \ 1]$	
sty	Matrix of membership functions - output y		Double (F64)
		$\odot [-1 \ 0 \ 0; \ 0 \ 0 \ 0; \ 1 \ 0 \ 0]$	
rls	Matrix of inference rules		Byte (U8)
		$\odot [1 \ 2 \ 3 \ 100; \ 1 \ 1 \ 1 \ 100; \ 1 \ 0 \ 3 \ 100]$	

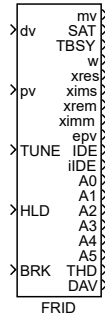
Output

y	Analog output of the block	Double (F64)
ir	Dominant rule	Long (I32)
wr	Degree of truth of the dominant rule	Double (F64)

FRID – Frequency response identification

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **FRID** block is designed for experimental identification of the frequency response of linear and weakly nonlinear systems. The basic principle of the function is to use an excitation signal generator in the form of a swept harmonic function with time-varying frequency, amplitude and DC component. This signal is injected into the input of the identified system in an open or closed loop setting. Based on the observed output response of the system, a non-parametric model in the form of frequency transfer points of the system is computed using an internally implemented state estimator.

Its primary application is for systems with oscillatory dynamics, where accurate identification of frequency characteristics, including location and shape of resonant modes, is crucial for optimal design of control algorithms.

The identification procedure is initiated upon detecting a rising edge on the **TUNE** input. The procedure alternates between two modes: sweeping and measuring. In the sweeping mode, the **FRID** block generates a swept sinusoidal signal on the **mv** output as excitation for the system under test in the form:

$$mv(t) = A(t) \sin\left\{\int_0^t \omega(\tau) d\tau\right\} + u_0(t),$$

where $A(t)$ is the current amplitude, $\omega(t)$ is the current excitation frequency, and $u_0(t)$ is the DC component. If **ADAPT_EN** is set to **on**, the amplitude and DC component are dynamically adjusted to map the system's response across a spectrum of frequencies. The amplitude adaptation is especially important for weakly damped resonant systems, for which the gain may vary considerably in the vicinity of (anti)resonances. Initial values are given by the **uamp** parameter for amplitude and **ubias** for the DC component. The rate of change in amplitude is given by **adapt_rc**.

The frequency sweeping mode is determined by the **isweep** parameter. The rate of frequency change (sweeping) is given by the **cp** parameter, which specifies the relative

reduction of the initial period $T_b = \frac{2\pi}{\omega b}$ of the excitation sinusoid over time T_b , thus

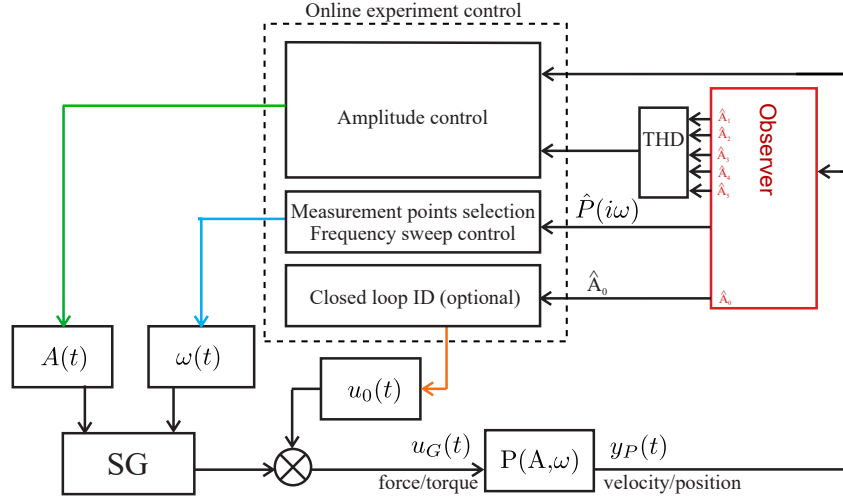
$$c_p = \frac{\omega b}{\omega(T_b)} = \frac{\omega b}{\omega b e^{\gamma T_b}} = e^{-\gamma T_b}.$$

The **cp** parameter typically ranges in the interval $\text{cp} \in \langle 0.95; 1 \rangle$.

For specific ω values, the sweeping is paused, and the block switches to the measurement phase to acquire a point of the frequency response. Depending on the **immode** parameter setting, the ω values can be determined manually, by a frequency range, or automatically (see parameter description below). This process facilitates the identification of key dynamic properties, including (anti)resonant frequencies.

The core of the **FRID** block algorithm is based on recursive Kalman filter discretization, which optimizes computational efficiency while maintaining high accuracy of frequency estimation. The block dynamically adjusts the excitation signal parameters during the identification experiment to mitigate the effects of nonlinearities and achieve a reliable representation of the system dynamics by a linear frequency response model. The amplitude adaptation also serves to keep the output within user-defined limits. The tracking speed of the observed output response can be influenced by the parameter **obw**, which affects the bandwidth of the estimator. A larger value results in faster tracking and steady state estimation of the frequency response at the cost of greater amplification of measurement noise.

Below is the schematic representation of the **FRID** block within a system identification setup. The diagram illustrates the block's interaction with the signal generator and the mechanical system being analyzed.



The **FRID** block provides outputs that contain detailed information about the frequency characteristics of the system under test. Outputs **A1** to **A5** provide estimates of the amplitudes of the first five harmonic components of the Fourier series of the plant output response the actual excitation frequency. These amplitudes are key to understanding how the system responds to different input signal frequencies and allow identification of system characteristics such as resonant frequencies. They are also used to detect nonlinearities.

The output THD, or total harmonic distortion, is an indication of the degree to which the system response deviates from the ideal linear response. A low THD value indicates that the system behaves predominantly linearly with respect to the input signal, while a high THD value may indicate the presence of nonlinearities such as saturation or backlash. In addition to nonlinearities, this also takes into account the effect of measurement noise. The THD value is calculated as the ratio of the RMS (root mean square) amplitudes of the higher harmonic components to the RMS amplitude of the first harmonic component, providing a comprehensive view of the quality and linearity of the system response.

Utilizing these outputs allows users to better understand system dynamics and tailor control strategies to achieve optimal performance. More details on the block's operation can be found in [5].

This block propagates the signal quality. More information can be found in the 1.4 section.

Input

dv	Feedforward control variable	Double (F64)
pv	Process variable	Double (F64)
TUNE	Start the tuning experiment	Bool
HLD	Hold	Bool
BRK	Stop the tuning experiment	Bool

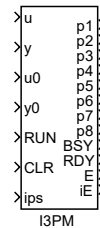
Parameter

ubias	Static component of the exciting signal	Double (F64)
uamp	Amplitude of the exciting signal	⊙1.0 Double (F64)
wb	Frequency interval lower limit [rad/s]	⊙1.0 Double (F64)
wf	Frequency interval higher limit [rad/s]	⊙10.0 Double (F64)
isweep	Frequency sweeping mode	⊙1 Long (I32)
	1 Logarithmic	
	2 Linear	
	3 Combined	
cp	Sweeping Rate	⊙0.995 Double (F64)
iavg	Number of values for averaging	⊙10 Long (I32)
obw	Observer bandwidth	⊙2 Long (I32)
	1 LOW	
	2 NORMAL	
	3 HIGH	
stime	Settling period [s]	⊙10.0 Double (F64)
umax	Maximum generator amplitude	⊙1.0 Double (F64)
thdmin	Minimum demanded THD threshold	⊙0.1 Double (F64)
adapt_rc	Maximum rate of amplitude variation	⊙0.001 Double (F64)
pv_max	Maximum desired process value	⊙1.0 Double (F64)
pv_sat	Maximum allowed process value	⊙2.0 Double (F64)

I3PM – Identification of a three parameter model

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **I3PM** block identifies a three-parameter system model using the method of generalized moments. Unlike the **PIDMA** block, it is necessary to independently control the identification experiment. The block provides estimated parameters on the outputs **p1-6** in one of the following forms, depending on the value of the **ips** input:

- **0:** First-order model with transport delay given as $F_{FOPDT}(s) = \frac{K}{\tau s + 1} \cdot e^{-Ds}$, where $p1 = K$, $p2 = D$, $p3 = \tau$,
- **1:** Input and output moments, where $p1 = mu_0$, $p2 = mu_1$, $p3 = mu_2$, $p4 = my_0$, $p5 = my_1$, $p6 = my_2$,
- **2:** Process moments, where $p1 = mp_0$, $p2 = mp_1$, $p3 = mp_2$,
- **3:** Process characteristic numbers, where $p1 = \kappa$, $p2 = \mu$, $p3 = \sigma^2$, $p4 = \sigma$. For more information on characteristic numbers, see the documentation of the **PSMPC** block.

Outputs **p7** and **p8** are reserved for later use.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

u	Input of the identified system	Double (F64)
y	Output of the identified system	Double (F64)
u0	Input steady state	Double (F64)
y0	Output steady state	Double (F64)
RUN	Execute identification	Bool
CLR	Block reset	Bool

ips	Meaning of the output signals	Long (I32)
0	FOPDT model	
1	moments of input and output	
2	process moments	
3	characteristic numbers	

Parameter

tident	Duration of identification [s]	⊙100.0 Double (F64)
irtype	Controller type (control law)	⊙6 Long (I32)
1	D	
2	I	
3	ID	
4	P	
5	PD	
6	PI	
7	PID	
ispeed	Desired closed loop speed	⊙2 Long (I32)
1	Slow closed loop	
2	Normal (middle fast) CL	
3	Fast closed loop	

Output

p1..p8	Identified parameter	Double (F64)
BSY	Busy flag	Bool
RDY	Outputs valid (ready flag)	Bool
E	Error indicator	Bool
off ...	No error	
on	An error occurred	
iE	Error code	Long (I32)
1	Premature termination	
2	$\mu_0=0$	
3	$m_p=0$	
4	$\sigma^2 < 0$	

LC – Lead compensator

Block Symbol

Licence: [STANDARD](#)



Function Description

The LC block is a discrete simulator of derivative element

$$C(s) = \frac{\mathbf{td} \cdot s}{\frac{\mathbf{td}}{\mathbf{nd}} \cdot s + 1},$$

where **td** is the derivative constant and **nd** determines the influence of parasite 1st order filter. It is recommended to use $2 \leq \mathbf{nd} \leq 10$. 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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	Analog input of the block	Double (F64)
R1	Block reset	Bool
HLD	Hold	Bool

Parameter

td	Derivative time constant	⊙1.0	Double (F64)
nd	Derivative filtering parameter	⊙10.0	Double (F64)
ISSF	Steady state at start-up		Bool
	off ... Zero initial state		
	on Initial steady state		

Output

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

LLC – Lead-lag compensator

Block Symbol

Licence: [STANDARD](#)



Function Description

The LLC block is a discrete simulator of integral-derivative element

$$C(s) = \frac{a \cdot \text{tau} \cdot s + 1}{\text{tau} \cdot s + 1},$$

where **tau** is the denominator time constant and the time constant of numerator is an **a**-multiple of **tau** (**a * tau**). 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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	Analog input of the block	Double (F64)
R1	Block reset	Bool
HLD	Hold	Bool

Parameter

tau	Time constant	⊙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	

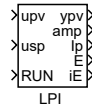
Output

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

LPI – Loop performance index

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **LPI** (*Loop Performance Index*) functional block is designed to evaluate the quality of feedback control by influencing the signal value before it is fed into the controller and measures the system's response. This block is useful for analyzing and identifying the behavior of the control loop in real-time.

The process variable **pv** of the control loop is connected to the **upv** input and from the **ypv** output to the controller. The setpoint **sp** of the control loop is fed to the **usp** input. The block is activated by the **RUN** signal only in the automatic mode of the controller when it is desired to perform identification of the control loop.

Upon activation (**RUN**=1), the **LPI** block injects a sinusoidal signal into the process variable with a defined amplitude **ad** and frequency **fd**, allowing the measurement of the system's response. The output signal is further processed by a **BandPass** filter and Fourier transform to determine the average signal amplitude. The resulting performance index **Ip** is calculated based on the ratio between the set parameters and the measured amplitude, providing a quantitative evaluation of the control system's disturbance suppression.

The output **Ip** reflects how effectively the control system suppresses disturbances in the defined frequency band **fa**. A value of **Ip**=1 indicates that the system suppresses disturbances in accordance with expectations; values higher than 1 indicate better performance; lower values indicate poorer control loop settings.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

upv	Input process variable	Double (F64)
usp	Input setpoint variable	Double (F64)
RUN	Enable execution	Bool

Parameter

ms	Sensitivity function upper limit	↓1.00001 ↑1000.0 ⊙2.0	Double (F64)
fa	Available bandwidth	↓1e-10 ↑1e+10 ⊙10.0	Double (F64)
fd	Excitation/measured frequency	↓1e-10 ↑1e+10 ⊙1.0	Double (F64)

ad	Excitation amplitude	⊙0.01	Double (F64)
nper	Window size (number of periods of fd)	↓1 ⊙4	Long (I32)
ifrunit	Frequency units	⊙1	Long (I32)
	1 Hz		
	2 rad/s		
xi	Filter damping ratio	↓0.001 ↑100.0 ⊙1.0	Double (F64)
nmax	Allocated size of array	↓10 ↑10000000 ⊙256	Long (I32)

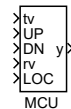
Output

ypv	Output process variable	Double (F64)
amp	Signal amplitude after filtering	Double (F64)
Ip	Control loop performance index	Double (F64)
E	Error indicator	Bool
iE	Error code	Error

MCU – Manual control unit

Block Symbol

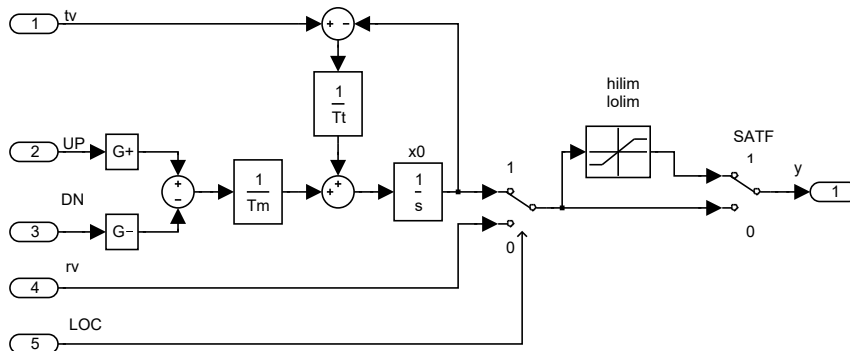
Licence: [STANDARD](#)



Function Description

The **MCU** block is suitable for manual setting of the numerical output value y , e.g. a setpoint. In the local mode ($LOC = \text{on}$) 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 elapsing 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 ($SATF = \text{on}$) by saturation limits lo_{lim} and hi_{lim} . If none of the buttons is pushed ($UP = \text{off}$ and $DN = \text{off}$), the output y tracks the input value tv . The tracking speed is controlled by the integration time constant t_t .

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



This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

tv	Tracking variable	Double (F64)
UP	The UP signal (up, more)	Bool
DN	The DOWN signal (down, less)	Bool

rv	Remote value	Double (F64)
LOC	Local or remote mode	Bool

Parameter

tt	Tracking time constant	⊙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)

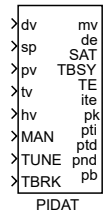
Output

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

PIDAT – PID controller with relay autotuner

Block Symbol

Licence: [AUTOTUNING](#)



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. While the tuning experiment is running, the **ite** output shows the estimated time to finish in seconds.

A successful experiment is indicated by **TE = off** and the controller parameters can be found on the outputs **pk**, **pti**, **ptd**, **pnd** and **pb**. The **c** weighting factor is assumed (and recommended) **c=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.

For the gain setting, a value of $K = 0$ disables the controller. Negative values are not allowed (use the **RACT** parameter for negative effect instead). Setting the integral time constant to $T_i = 0$ disables the integral component of the controller (same effect as disabling it with the **irtype** parameter). For $T_d = 0$, the derivative component of the controller is disabled. Static gain of the process **k0** must be provided in case of **iainf = 3, 4, 5**.

It is recommended not to change the parameters **n1**, **mm**, **ntime**, **rerrap** and **aerrph**.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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	Bool
	off ... Automatic mode	
	on Manual mode	
TUNE	Start the tuning experiment	Bool
TBRK	Stop the tuning experiment	Bool

Parameter

irtype	Controller type (control law)	⊙6	Long (I32)
	1 [D]		
	2 [I]		
	3 [ID]		
	4 [P]		
	5 [PD]		
	6 PI		
	7 PID		
RACT	Reverse action flag		Bool
	off ... Higher mv -> higher pv		
	on Higher mv -> lower pv		
k	Controller gain	↓0.0 ⊙1.0	Double (F64)
ti	Integral time constant	↓0.0 ⊙4.0	Double (F64)
td	Derivative time constant	↓0.0 ⊙1.0	Double (F64)
nd	Derivative filtering parameter	↓0.0 ⊙10.0	Double (F64)
b	Setpoint weighting - proportional part	↓0.0 ⊙1.0	Double (F64)
c	Setpoint weighting - derivative part	↓0.0	Double (F64)
tt	Tracking time constant	↓0.0 ⊙1.0	Double (F64)
hilim	Upper limit of the controller output	⊙1.0	Double (F64)
lolim	Lower limit of the controller output	⊙-1.0	Double (F64)
iainf	Type of apriori information	⊙1	Long (I32)
	1 No apriori information		
	2 Astatic process		
	3 Low order process		
	4 Static process + slow CLSR		
	5 Static process		
	6 Static process + fast CLSR		
k0	Static gain	⊙1.0	Double (F64)
n1	Maximum number of half-periods - freq. response point	⊙20	Long (I32)

<code>mm</code>	Maximum number of half-periods - averaging	$\odot 4$	Long (I32)
<code>amp</code>	Relay controller amplitude	$\odot 0.1$	Double (F64)
<code>uhys</code>	Relay controller hysteresis		Double (F64)
<code>ntime</code>	Length of noise estimation period [s]	$\odot 5.0$	Double (F64)
<code>rerrap</code>	Termination value - amplitude relative error	$\odot 0.1$	Double (F64)
<code>aerrph</code>	Termination value - phase absolute error	$\odot 10.0$	Double (F64)
<code>maxdev</code>	Maximal admissible deviation error	$\odot 1.0$	Double (F64)

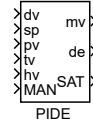
Output

<code>mv</code>	Manipulated variable (controller output)		Double (F64)
<code>de</code>	Deviation error		Double (F64)
<code>SAT</code>	Saturation flag		Bool
	<code>off</code> ... The controller implements a linear control law		
	<code>on</code> The controller output is saturated		
<code>TBSY</code>	Tuner busy flag		Bool
<code>TE</code>	Tuning error		Bool
	<code>off</code> ... Autotuning successful		
	<code>on</code> An error occurred during the experiment		
<code>ite</code>	Error code		Long (I32)
	1000 .. Signal/noise ratio too low		
	1001 .. Hysteresis too high		
	1002 .. Too tight termination rule		
	1003 .. Phase out of interval		
<code>pk</code>	Proposed controller gain		Double (F64)
<code>pti</code>	Proposed integral time constant		Double (F64)
<code>ptd</code>	Proposed derivative time constant		Double (F64)
<code>pnd</code>	Proposed derivative component filtering		Double (F64)
<code>pb</code>	Proposed weighting factor - proportional component		Double (F64)

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 ε 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

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

where

$$\beta = \frac{K\varepsilon}{1 - K\varepsilon}$$

and $U(s)$ is the Laplace transform of the manipulated variable **mv**, $W(s)$ is the Laplace transform of the setpoint **sp**, $Y(s)$ is the Laplace transform of the process variable **pv**, $E(s)$ is the Laplace transform of the deviation error, $Z(s)$ is the Laplace transform of the feedforward control variable **dv** and K , T_i , T_d , N , ε ($= b_p/100$), b and c are the controller parameters. The sign of the right hand side depends on the parameter **RACT**. The range of the manipulated variable **mv** (position controller output) is limited by parameters **hilim**, **lolim**.

By connecting the output **mv** to the input **tv** and choosing the **tt** parameter appropriately, we achieve the desired behaviour of the controller when reaching the saturation values of **mv**. This eliminates the undesirable integral windup effect while ensuring smooth switching (bumpless transfer) between automatic and manual modes.

In the manual mode (**MAN** = **on**), the input **hv** is copied to the output **mv** unless saturated. In this mode the inner controller state tracks the signal connected to the **tv** input so the successive switching to the automatic mode is bumpless. But the tracking is not precise for $\varepsilon > 0$.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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	Bool
	off ... Automatic mode	
	on Manual mode	

Parameter

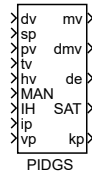
irtype	Controller type (control law)	⊙6	Long (I32)
	1 D		
	2 I		
	3 ID		
	4 P		
	5 PD		
	6 PI		
	7 PID		
RACT	Reverse action flag		Bool
	off ... Higher mv -> higher pv		
	on Higher mv -> lower pv		
k	Controller gain	↓0.0 ⊙1.0	Double (F64)
ti	Integral time constant	↓0.0 ⊙4.0	Double (F64)
td	Derivative time constant	↓0.0 ⊙1.0	Double (F64)
nd	Derivative filtering parameter	↓0.0 ⊙10.0	Double (F64)
b	Setpoint weighting - proportional part	↓0.0 ⊙1.0	Double (F64)
c	Setpoint weighting - derivative part	↓0.0	Double (F64)
tt	Tracking time constant	↓0.0 ⊙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)

Output

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	

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 $vp < thrsha(0)$, the set 1 for $thrsha(0) < vp < thrsha(1)$ etc. till the set 5 for $thrsha(4) < vp$. The index of the active parameter set is available at the **kp** output.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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	Bool
	off ... Automatic mode	
	on Manual mode	
IH	Integrator hold	Bool
	off ... Integration enabled	
	on Integration disabled	
ip	Parameter set index	↓0 ↑5 Long (I32)
vp	Switching analog signal	Double (F64)

Parameter

hilim	Upper limit of the controller output	⊙1.0 Double (F64)
lolim	Lower limit of the controller output	⊙-1.0 Double (F64)
dz	Dead zone	Double (F64)

icotype	Controller output type	⊙1	Long (I32)
	1 Analog		
	2 PWM		
	3 SCU		
	4 SCUV		
nmax	Allocated size of array	↓4 ↑10000 ⊙10	Long (I32)
GSCF	Switch parameters by analog signal vp		Bool
	off ... Index-based switching		
	on Analog signal based switching		
hys	Hysteresis for controller parameters switching		Double (F64)
irtypea	Vector of controller types (control laws)	⊙[6 6 6 6 6 6]	Byte (U8)
RACTA	Vector of reverse action flags	⊙[0 0 0 0 0 0]	Bool
ka	Vector of controller gains	⊙[1.0 1.0 1.0 1.0 1.0 1.0]	Double (F64)
tia	Vector of integral time constants	⊙[4.0 4.0 4.0 4.0 4.0 4.0]	Double (F64)
tda	Vector of derivative time constants	⊙[1.0 1.0 1.0 1.0 1.0 1.0]	Double (F64)
nda	Vector of derivative filtering parameters	⊙[10.0 10.0 10.0 10.0 10.0 10.0]	Double (F64)
ba	Setpoint weighting factors - proportional part	⊙[1.0 1.0 1.0 1.0 1.0 1.0]	Double (F64)
ca	Setpoint weighting factors - derivative part	⊙[0.0 0.0 0.0 0.0 0.0 0.0]	Double (F64)
tta	Vector of tracking time constants	⊙[1.0 1.0 1.0 1.0 1.0 1.0]	Double (F64)
thrsha	Vector of thresholds for switching the parameters	⊙[0.1 0.2 0.3 0.4 0.5 0]	Double (F64)

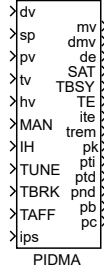
Output

mv	Manipulated variable (controller output)	Double (F64)
dmv	Controller velocity output (difference)	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	
kp	Active parameter set index	Long (I32)

PIDMA – PID controller with moment autotuner

Block Symbol

Licence: [AUTOTUNING](#)



Function Description

The PIDU block serves as a foundational component for constructing a complete PID controller (P, I, PI, PD, PID, PI+D). In its simplest form, it can operate independently as a standard PID controller with two degrees of freedom, accommodating both automatic (**MAN** = **off**) and manual modes (**MAN** = **on**).

In automatic mode (**MAN** = **off**), the **PIDMA** block executes the PID control law with two degrees of freedom as follows:

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

where $U(s)$ is Laplace transform of the manipulated variable **mv**, $W(s)$ is Laplace transform of the setpoint variable **sp**, $Y(s)$ is Laplace transform of the process variable **pv**, $Z(s)$ is Laplace transform of the feedforward control variable **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 **RACT**. The range of the manipulated variable **mv** (position controller output) is limited by parameters **hilim**, **lolim**. The parameter **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 **IH** (**IH** = **on**). For the proper function of the controller, it is necessary to connect the output **mv** of the controller to the controller input **tv** and properly set the tracking time constant **tt**.

The recommended default value for the PID controller is $tt \approx \sqrt{T_i T_d}$, and for the PI controller, it is $tt \approx T_i/2$. This ensures a bumpless transfer during switching between manual and automatic modes and correct anti-windup functionality when the output **mv** saturates. Adjusting the parameter **tt** allows for precise behaviour adjustment during saturation (e.g., bouncing off the limits due to noise) and when switching between multiple controllers (the size of the jump when switching, if there is a deviation error). A value of 0 sets recommended default values for PI and PID controllers. For controllers without an integral part, it means disabling the tracking. To enable tracking for P or PD

controllers (e.g., for control around a setpoint), set a positive value for `tt` higher than the sampling period. Disabling tracking for controllers with an integral part is not possible due to the risk of windup.

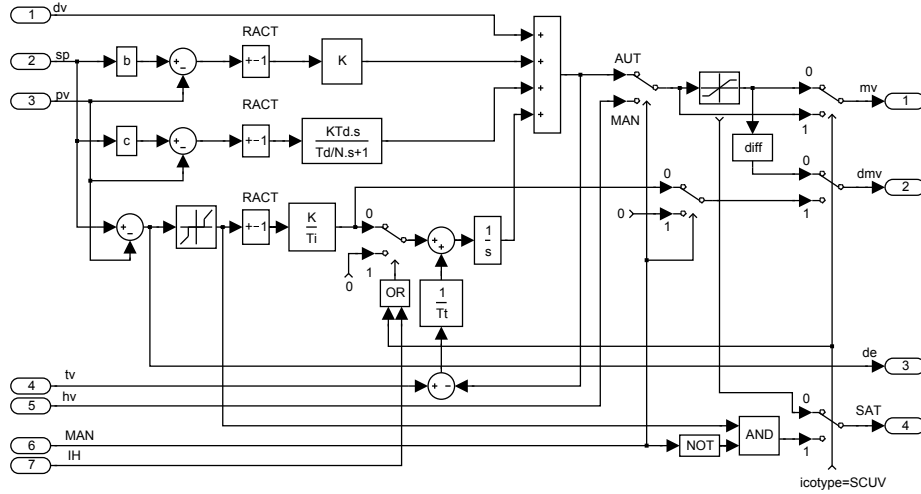
For the gain setting, a value of $K = 0$ disables the controller. Negative values are not allowed (use the `RACT` parameter for negative effect instead). Setting the integral time constant to $T_i = 0$ disables the integral component of the controller (same effect as disabling it with the `irtype` parameter). For $T_d = 0$, the derivative component of the controller is disabled. The additional outputs `dmv`, `de`, and `SAT` sequentially provide the controller's velocity output (difference of `mv`), the deviation error, and the saturation flag of the controller's output `mv`.

The `PIDMA` block can be prepared for connection with other types of control blocks by using the `icotype` parameter. The `icotype` parameter can be set to the following values with the given meanings:

- **1: Analog** - standard block mode,
- **2: PWM** - mode suitable for connecting the output `mv` to the input of pulse-width modulated regulation [PWM](#),
- **3: SCU** - mode for connection with a step controller with position feedback [SCU](#),
- **4: SCUV** - mode for connection with a step controller without positional feedback [SCUV](#).

For the last option, the meanings of the outputs `mv`, `dmv`, and `SAT` are modified in this case: the output `mv` equals the sum of the P and D components of the controller, while the output `dmv` provides the difference of its I component, and the output `SAT` carries information for the [SCUV](#) block whether the deviation error `de` in automatic mode is less than the dead zone `dz`. Additionally, for connecting the `PIDMA` and [SCUV](#) blocks, it is recommended to set the setpoint weighting factor for the derivative component `c` to zero.

In the manual mode (`MAN = 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:

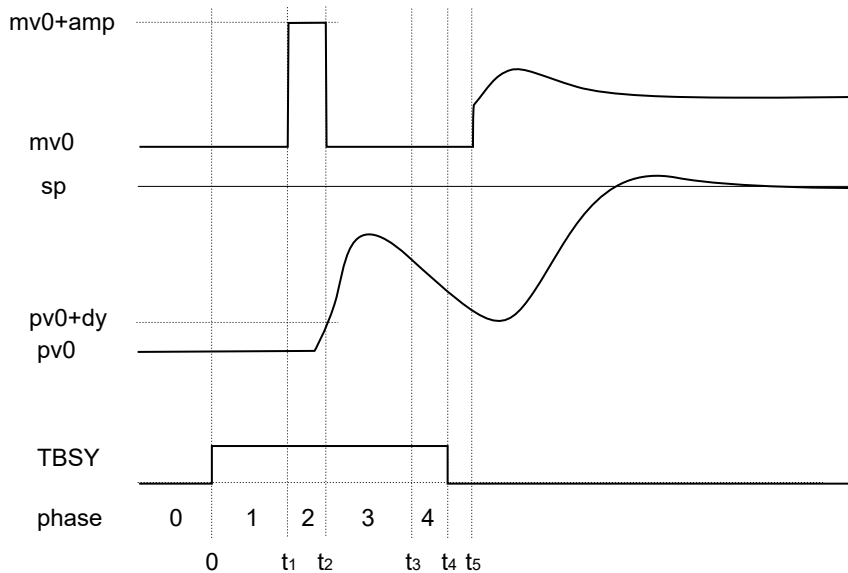


The block **PIDMA** extends the control function of the standard PID controller by the built in autotuning feature. Before start of the autotuner the operator have to reach the steady state of the process at a suitable working point (in manual or automatic mode) and specify the required type of the controller **ittype** (PI or PID) and other tuning parameters (**iainf**, **DGC**, **tdg**, **tn**, **amp**, **dy** and **ispeed**). The identification experiment is started by the rising edge **off** \rightarrow **on** on the input **TUNE** (input **TBRK** finishes the experiment). In this mode (**TBSY** = **on**), first of all the noise and possible drift gradient (**DGC** = **on**) are estimated during the user specified time (**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 **amp**. The pulse is finished when the process variable **pv** deviates from the steady value more than the **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 **trem**.

If the experiment ends successfully (**TE** = **off**), then depending on the input **ips** appears on the outputs:

- **0**: Designed parameters **pk**, **pti**, **ptd**, **pnd**, **pb**, **pc**.
- **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**). See the [FOPDT](#) block.
- **3**: Three-parameter second-order plus dead-time model with double time constant: static gain (**pk**), dead-time (**pti**), time constant (**ptd**). See the [SOPDT](#) block.
- **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**).

Other values of the `ips` input are reserved for custom specific purposes. For (`TE = on`) the output `ite` specifies the experiment error more closely. The function of the autotuner is illustrated in the following picture.



During the experiment, the output `ite` indicates the autotuner phases. In the phase of estimation of the response decay rate (`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 `ite=100`.

Remark: The rising edge `off→on` at `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).

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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

IH	Integrator hold	Bool
	off ... Integration enabled	
	on Integration disabled	
TUNE	Start the tuning experiment	Bool
TBRK	Stop the tuning experiment	Bool
TAFF	Tuning affirmation	Bool
	off ... Parameters are only computed	
	on Parameters are set into the control law	
ips	Meaning of the output signals	Long (I32)
	0 PID parameters	
	1 process moments	
	2 FOPDT model	
	3 SOPDT model	
	4 boundaries for manual tuning of controller gain	

Parameter

irtype	Controller type (control law)	⊙6	Long (I32)
	1 D		
	2 I		
	3 ID		
	4 P		
	5 PD		
	6 PI		
	7 PID		
RACT	Reverse action flag		Bool
	off ... Higher mv -> higher pv		
	on Higher mv -> lower pv		
k	Controller gain	↓0.0 ⊙1.0	Double (F64)
ti	Integral time constant	↓0.0 ⊙4.0	Double (F64)
td	Derivative time constant	↓0.0 ⊙1.0	Double (F64)
nd	Derivative filtering parameter	↓0.0 ⊙10.0	Double (F64)
b	Setpoint weighting - proportional part	↓0.0 ↑2.0 ⊙1.0	Double (F64)
c	Setpoint weighting - derivative part	↓0.0 ↑2.0	Double (F64)
tt	Tracking time constant	↓0.0 ⊙1.0	Double (F64)
hilim	Upper limit of the controller output	⊙1.0	Double (F64)
lolim	Lower limit of the controller output	⊙-1.0	Double (F64)
dz	Dead zone		Double (F64)
icotype	Controller output type	⊙1	Long (I32)
	1 Analog		
	2 PWM		
	3 SCU		
	4 SCUV		
itttype	Controller type to be designed	⊙6	Long (I32)
	6 PI		
	7 PID		

iainf	Type of apriori information	⊙1	Long (I32)
	1 Static process		
	2 Astatic process		
DGC	Drift gradient compensation	⊙on	Bool
	off ... Disabled		
	on Enabled		
tdg	Drift gradient estimation time [s]	⊙60.0	Double (F64)
tn	Length of noise estimation period [s]	⊙5.0	Double (F64)
amp	Tuning pulse amplitude	⊙0.5	Double (F64)
dy	Tuning pulse get down threshold	↓0.0 ⊙0.1	Double (F64)
ispeed	Desired closed loop speed	⊙2	Long (I32)
	1 Slow closed loop		
	2 Normal (middle fast) CL		
	3 Fast closed loop		
ipid	PID controller form	⊙1	Long (I32)
	1 Parallel form		
	2 Series form		

Output

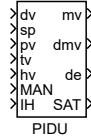
mv	Manipulated variable (controller output)	Double (F64)
dmv	Controller velocity output (difference)	Double (F64)
de	Deviation error	Double (F64)
SAT	Saturation flag	Bool
	off ... The controller implements a linear control law	
	on The controller output is saturated	
TBSY	Tuner busy flag	Bool
TE	Tuning error	Bool
	off ... Autotuning successful	
	on An error occurred during the experiment	

ite	Error code	Long (I32)
	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	
	13 Invalid format of sParams input string	
	100 ... Manual termination of tuning (warning)	
	-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	
trem	Estimated time to finish the tuning experiment [s]	Double (F64)
pk	Proposed controller gain	Double (F64)
pti	Proposed integral time constant	Double (F64)
ptd	Proposed derivative time constant	Double (F64)
pnd	Proposed derivative component filtering	Double (F64)
pb	Proposed weighting factor - proportional component	Double (F64)
pc	Proposed weighting factor - derivative component	Double (F64)

PIDU – PID controller unit

Block Symbol

Licence: [STANDARD](#)



Function Description

The **PIDU** block serves as a foundational component for constructing a complete PID controller (P, I, PI, PD, PID, PI+D). In its simplest form, it can operate independently as a standard PID controller with two degrees of freedom, accommodating both automatic (**MAN = off**) and manual modes (**MAN = on**).

In automatic mode (**MAN = off**), the **PIDU** block executes the PID control law with two degrees of freedom as follows:

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

where $U(s)$ is Laplace transform of the manipulated variable **mv**, $W(s)$ is Laplace transform of the setpoint variable **sp**, $Y(s)$ is Laplace transform of the process variable **pv**, $Z(s)$ is Laplace transform of the feedforward control variable **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 **RACT**. The range of the manipulated variable **mv** (position controller output) is limited by parameters **hilim**, **lolim**. The parameter **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 **IH** (**IH = on**). For the proper function of the controller, it is necessary to connect the output **mv** of the controller to the controller input **tv** and properly set the tracking time constant **tt**.

The recommended default value for the PID controller is $tt \approx \sqrt{T_i T_d}$, and for the PI controller, it is $tt \approx T_i/2$. This ensures a bumpless transfer during switching between manual and automatic modes and correct anti-windup functionality when the output **mv** saturates. Adjusting the parameter **tt** allows for precise behaviour adjustment during saturation (e.g., bouncing off the limits due to noise) and when switching between multiple controllers (the size of the jump when switching, if there is a deviation error). A value of 0 sets recommended default values for PI and PID controllers. For controllers without an integral part, it means disabling the tracking. To enable tracking for **P** or **PD** controllers (e.g., for control around a setpoint), set a positive value for **tt** higher than the sampling period. Disabling tracking for controllers with an integral part is not possible due to the risk of windup.

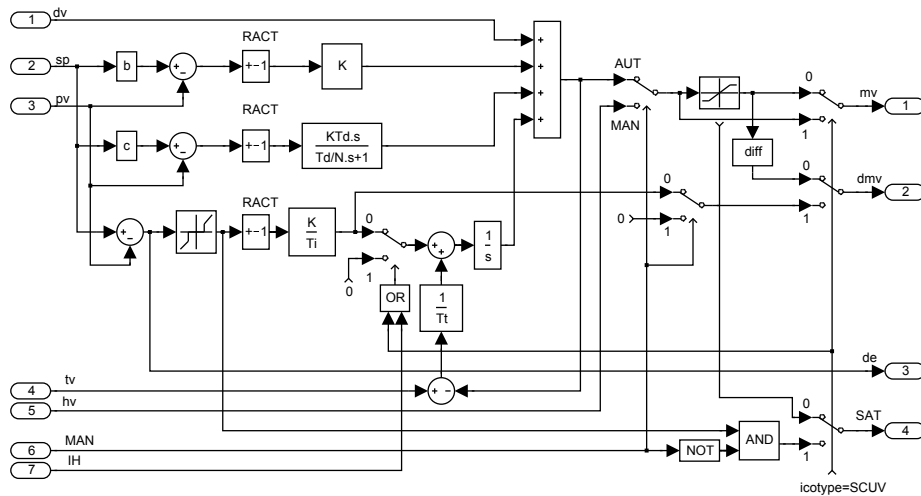
For the gain setting, a value of $K = 0$ disables the controller. Negative values are not allowed (use the **RACT** parameter for negative effect instead). Setting the integral time constant to $T_i = 0$ disables the integral component of the controller (same effect as disabling it with the **irtype** parameter). For $T_d = 0$, the derivative component of the controller is disabled. The additional outputs **dmv**, **de**, and **SAT** sequentially provide the controller's velocity output (difference of **mv**), the deviation error, and the saturation flag of the controller's output **mv**.

The PIDU block can be prepared for connection with other types of control blocks by using the **icotype** parameter. The **icotype** parameter can be set to the following values with the given meanings:

- 1 **Analog** - standard block mode,
- 2 **PWM** - mode suitable for connecting the output **mv** to the input of pulse-width modulated regulation **PWM**,
- 3 **SCU** - mode for connection with a step controller with position feedback **SCU**,
- 4 **SCUV** - mode for connection with a step controller without positional feedback **SCUV**.

For the last option, the meanings of the outputs **mv**, **dmv**, and **SAT** are modified in this case: the output **mv** equals the sum of the P and D components of the controller, while the output **dmv** provides the difference of its I component, and the output **SAT** carries information for the **SCUV** block whether the deviation error **de** in automatic mode is less than the dead zone **dz**. Additionally, for connecting the PIDU and **SCUV** blocks, it is recommended to set the setpoint weighting factor for the derivative component **c** to zero.

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



This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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	Bool
	off ... Automatic mode	
	on Manual mode	
IH	Integrator hold	Bool
	off ... Integration enabled	
	on Integration disabled	

Parameter

irtype	Controller type (control law)	⊙6	Long (I32)
	1 D		
	2 I		
	3 ID		
	4 P		
	5 PD		
	6 PI		
	7 PID		
RACT	Reverse action flag		Bool
	off ... Higher mv -> higher pv		
	on Higher mv -> lower pv		
k	Controller gain	↓0.0 ⊙1.0	Double (F64)
ti	Integral time constant	↓0.0 ⊙4.0	Double (F64)
td	Derivative time constant	↓0.0 ⊙1.0	Double (F64)
nd	Derivative filtering parameter	↓0.0 ⊙10.0	Double (F64)
b	Setpoint weighting - proportional part	↓0.0 ↑2.0 ⊙1.0	Double (F64)
c	Setpoint weighting - derivative part	↓0.0 ↑2.0	Double (F64)
tt	Tracking time constant	↓0.0 ⊙1.0	Double (F64)
hilim	Upper limit of the controller output	⊙1.0	Double (F64)
lolim	Lower limit of the controller output	⊙-1.0	Double (F64)
dz	Dead zone		Double (F64)
icotype	Controller output type	⊙1	Long (I32)
	1 Analog		
	2 PWM		
	3 SCU		
	4 SCUV		

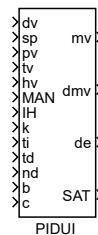
Output

<code>mv</code>	Manipulated variable (controller output)	Double (F64)
<code>dmv</code>	Controller velocity output (difference)	Double (F64)
<code>de</code>	Deviation error	Double (F64)
<code>SAT</code>	Saturation flag	Bool
	<code>off ...</code> The controller implements a linear control law	
	<code>on</code> The controller output is saturated	

PIDUI – PID controller unit with variable parameters

Block Symbol

Licence: [ADVANCED](#)



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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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	Bool
	off ... Automatic mode	
	on Manual mode	
IH	Integrator hold	Bool
	off ... Integration enabled	
	on Integration disabled	
k	Controller gain	Double (F64)
ti	Integral time constant	Double (F64)
td	Derivative time constant	Double (F64)
nd	Derivative filtering parameter	Double (F64)
b	Setpoint weighting - proportional part	Double (F64)
c	Setpoint weighting - derivative part	Double (F64)

Parameter

<code>irtype</code>	Controller type (control law)	⊙6	Long (I32)
	1 D		
	2 I		
	3 ID		
	4 P		
	5 PD		
	6 PI		
	7 PID		
<code>RACT</code>	Reverse action flag		Bool
	off ... Higher mv -> higher pv		
	on ... Higher mv -> lower pv		
<code>tt</code>	Tracking time constant	⊙1.0	Double (F64)
<code>hilim</code>	Upper limit of the controller output	⊙1.0	Double (F64)
<code>lolim</code>	Lower limit of the controller output	⊙-1.0	Double (F64)
<code>dz</code>	Dead zone		Double (F64)
<code>icotype</code>	Controller output type	⊙1	Long (I32)
	1 Analog		
	2 PWM		
	3 SCU		
	4 SCUV		

Output

<code>mv</code>	Manipulated variable (controller output)	Double (F64)
<code>dmv</code>	Controller velocity output (difference)	Double (F64)
<code>de</code>	Deviation error	Double (F64)
<code>SAT</code>	Saturation flag	Bool
	off ... The controller implements a linear control law	
	on ... The controller output is saturated	

POUT – Pulse output

Block 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**.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

U	Logical input of the block	Bool
----------	----------------------------	-------------

Parameter

dtime	Minimum width of the output pulse [s]	⊙1.0	Double (F64)
btime	Minimum delay between output pulses [s]	⊙1.0	Double (F64)

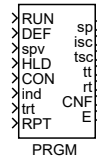
Output

Y	Logical output of the block	Bool
----------	-----------------------------	-------------

PRGM – Setpoint programmer

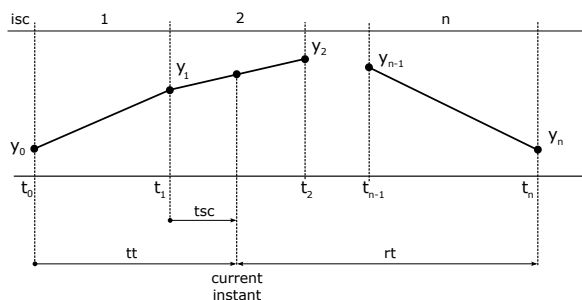
Block Symbol

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 ($\mathbf{tm} = [t_0, \dots, t_n]$) and output values ($\mathbf{y} = [y_0, \dots, y_n]$). 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.



This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

RUN	Enable execution	Bool
DEF	Initialize sp to the value of spv	Bool

spv	Initializing constant	Double (F64)
HLD	Output and timer freezing	Bool
CON	Continue from defined node	Bool
ind	Index of the node to continue from	Long (I32)
trt	Time to reach the defined node	Double (F64)
RPT	Repeating sequence	Bool

Parameter

nmax	Allocated size of array	$\downarrow 4 \uparrow 100000000$	$\odot 10$	Long (I32)
tmunits	Time units		$\odot 1$	Long (I32)
	1 seconds			
	2 minutes			
	3 hours			
tm	Vector of ascending node times	$\odot [0 \ 1 \ 2]$		Double (F64)
y	Vector of node values	$\odot [0 \ 1 \ 0]$		Double (F64)

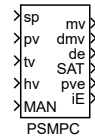
Output

sp	Setpoint variable	Double (F64)
isc	Current function sector	Long (I32)
tsc	Time elapsed since the start of current sector	Double (F64)
tt	Total elapsed time	Double (F64)
rt	Remaining time	Double (F64)
CNF	Configured curve is being followed	Bool
E	Error flag, nodes not ascending	Bool

PSMPC – Pulse-step model predictive controller

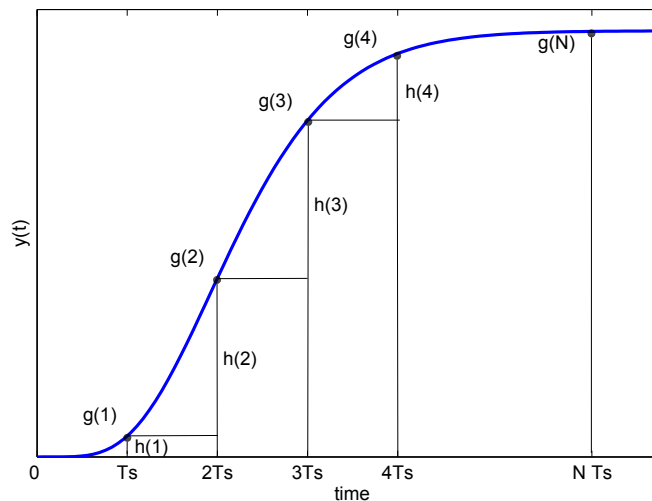
Block Symbol

Licence: [ADVANCED](#)



Function Description

The PSMPC (Pulse Step Model Predictive Control) functional block is designed for the implementation of high-quality controllers for difficult-to-control linear time-invariant systems with actuator constraints (e.g., systems with transport delay or non-minimum phase). It is especially advantageous in cases where a very rapid transition from one controlled variable value to another without overshoot is required. However, the PSMPC regulator can generally be used wherever standard PID regulators are commonly deployed and high regulation quality is demanded.



The PSMPC block is a predictive controller with explicitly defined constraints on the amplitude of manipulated variable. For prediction purposes, a model in the form of a discrete step response $g(j)$, $j = 1, \dots, N$ is used. The figure above illustrates how this sequence can be obtained from a continuous step response. Note that N must be chosen large enough so that the step response is described up to the steady state ($N \cdot T_s > t_{95}$, where T_s is the sampling period of the controller, and t_{95} is the time to settle to 95 % of the final value).

For stable, linear and t-invariant systems with a monotonous step response, it is alternatively possible to use a moment set model [6] and describe the system with only three characteristic numbers κ, μ and σ^2 , which can be determined from a simple pulse experiment. The controlled system can be approximated by first order plus dead-time system

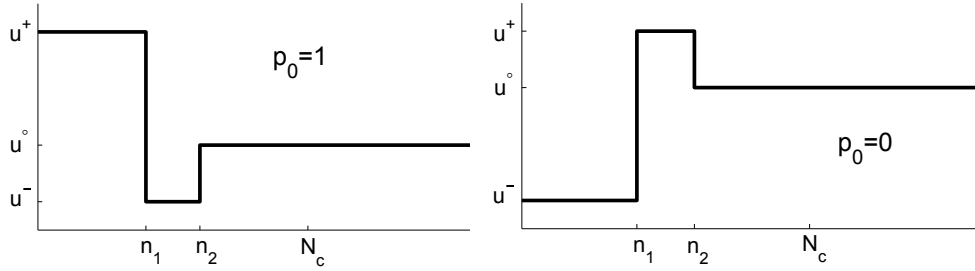
$$F_{FOPDT}(s) = \frac{K}{\tau s + 1} \cdot e^{-Ds}, \quad \kappa = K, \quad \mu = \tau + D, \quad \sigma^2 = \tau^2 \quad (7.1)$$

or second order plus dead-time system

$$F_{SOPDT}(s) = \frac{K}{(\tau s + 1)^2} \cdot e^{-Ds}, \quad \kappa = K, \quad \mu = 2\tau + D, \quad \sigma^2 = 2\tau^2 \quad (7.2)$$

with the same characteristic numbers. The type of approximation is selected by the `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:



Note that each of these sequences is uniquely defined by only four numbers $n_1, n_2 \in \{0, \dots, N_C\}$, p_0 and $u^\infty \in \langle u^-, u^+ \rangle$, where $N_C \in \{0, 1, \dots\}$ 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^∞) minimizes the criterion

$$I = \sum_{i=N_1}^{N_2} \hat{e}(k+i|k)^2 + \lambda \sum_{i=0}^{N_C} \Delta \hat{u}(k+i|k)^2 \rightarrow \min, \quad (7.3)$$

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 λ 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^∞ 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 of the predictive controller, in addition to the model of the controlled system and its input constraints, include the control horizon N_C , the prediction horizon

N_1 , N_2 , and the coefficient λ . Only the last-mentioned parameter, λ , is intended for manual fine-tuning of the control quality during routine commissioning. In the case of using the system model in the form of a transfer function (7.1) or (7.2), the parameters N_1 , N_2 are automatically selected based on the characteristic numbers μ, σ^2 . The controller can then be effectively tuned "manually" by adjusting the characteristic numbers of the process κ, μ, σ^2 .

Warning

It is necessary to set the **sr** array sufficiently large to avoid Matlab/Simulink crash when using the **PSMPC** block for simulation purposes. Especially when using FOPDT or SOPDT model, the **sr** array size must be greater than the length of the internally computed discrete step response.

This block propagates the signal quality. More information can be found in the 1.4 section.

Input

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	Bool
	off ... Automatic mode	
	on Manual mode	

Parameter

nc	Control horizon length	⊙5	Long (I32)
np1	Start of coincidence interval	⊙1	Long (I32)
np2	End of coincidence interval	⊙10	Long (I32)
lambda	Control signal penalization coefficient	⊙0.05	Double (F64)
umax	Upper limit of the controller output	⊙1.0	Double (F64)
umin	Lower limit of the controller output	⊙-1.0	Double (F64)
imtype	Controlled process model type	⊙3	Long (I32)
	1 FOPDT model		
	2 SOPDT model		
	3 Discrete step response		
kappa	Static gain	⊙1.0	Double (F64)
mu	Resident time constant	⊙20.0	Double (F64)
sigma	Measure of the system response length	⊙10.0	Double (F64)
nmax	Allocated size of array	↓10 ↑10000 ⊙32	Long (I32)
sr	Discrete step response sequence		Double (F64)
	⊙[0 0.2642 0.5940 0.8009 0.9084 0.9596 0.9826 0.9927 0.9970 0.9988 0.9995]		

Output

<code>mv</code>	Manipulated variable (controller output)	Double (F64)
<code>dmv</code>	Controller velocity output (difference)	Double (F64)
<code>de</code>	Deviation error	Double (F64)
<code>SAT</code>	Saturation flag	Bool
	<code>off</code> ... The controller implements a linear control law	
	<code>on</code> The controller output is saturated	
<code>pve</code>	Process variable estimation	Double (F64)
<code>iE</code>	Error code	Long (I32)
	0 No error	
	1 Incorrect FOPDT model	
	2 Incorrect SOPDT model	
	3 Invalid step response sequence	

PWM – Pulse width modulation

Block Symbol

Licence: [STANDARD](#)



Function Description

The **PWM** block performs pulse-width modulation of the input signal from the range from -1 to $+1$. Using this block, it is possible to implement a proportional control action even with actuators having a single (e.g., heating on/off) or dual (e.g., heating on/off and cooling on/off) binary inputs. The width L of the output pulse is determined by the relationship:

$$L = \text{pertm} * |u|,$$

where **pertm** is the modulation period. If $u > 0$ ($u < 0$), the pulse is generated at the output **UP** (**DN**). However, for practical reasons, the duration of the generated pulse is further adjusted according to the specified block parameters. The asymmetry factor **asyfac** defines the ratio between the width of the negative pulse **DN** and the width of the positive pulse **UP**. The modified widths are calculated according to the equations:

$$\begin{aligned} \text{if } u > 0 \text{ then } L(\text{UP}) &:= \begin{cases} L & \text{for } \text{asyfac} \leq 1.0 \\ L/\text{asyfac} & \text{for } \text{asyfac} > 1.0 \end{cases} \\ \text{if } u < 0 \text{ then } L(\text{DN}) &:= \begin{cases} L * \text{asyfac} & \text{for } \text{asyfac} \leq 1.0 \\ L & \text{for } \text{asyfac} > 1.0 \end{cases} \end{aligned}$$

which, for any value of **asyfac** > 0 , ensure that the maximum width of the generated pulses equals **pertm**. Moreover, if the calculated pulse width is less than **dtime**, then the resulting width is set to zero. If the calculated pulse width differs from **pertm** by less than **btime**, then the resulting width is set to **pertm**. If a positive pulse **UP** is followed by a negative pulse **DN** or vice versa, then the later pulse is, if necessary, shifted so that the distance between these two pulses is at least **offtime**. If **SYNCH** = **on**, then a change in the input u causes an immediate recalculation of the output pulse width, assuming the synchronization condition between the beginning of the modulation period and the moment of change in the input u is not met.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

<code>pertm</code>	Modulation period length [s]	⊙10.0	Double (F64)
<code>dtime</code>	Minimum width of the output pulse [s]	⊙0.1	Double (F64)
<code>btime</code>	Minimum delay between output pulses [s]	⊙0.1	Double (F64)
<code>offtime</code>	Minimum delay when altering direction [s]	⊙1.0	Double (F64)
<code>asyfac</code>	Asymmetry factor	⊙1.0	Double (F64)
<code>SYNCH</code>	Synchronization flag		Bool
	<code>off</code> ... Synchronization disabled		
	<code>on</code> Synchronization enabled		

Output

<code>UP</code>	The UP signal (up, more)	Bool
<code>DN</code>	The DOWN signal (down, less)	Bool

RLY – Relay with hysteresis

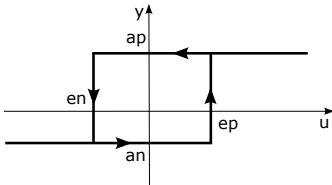
Block Symbol

Licence: [STANDARD](#)



Function Description

The RLY block transforms the input signal u to the output signal y according to the figure below.



This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

ep	Switch-on value	$\odot 1.0$	Double (F64)
en	Switch-off value	$\odot -1.0$	Double (F64)
ap	Output when ON	$\odot 1.0$	Double (F64)
an	Output when OFF	$\odot -1.0$	Double (F64)
$y0$	Initial output value		Double (F64)

Output

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

SAT – Saturation with variable limits

Block Symbol

Licence: [STANDARD](#)



Function Description

The **SAT** block copies the input u to the output y if the input signal satisfies $\text{lolim} \leq u$ and $u \leq \text{hilim}$, where lolim and hilim are state variables of the block. If $u < \text{lolim}$ ($u > \text{hilim}$), then $y = \text{lolim}$ ($y = \text{hilim}$). The upper and lower limits are either constants ($\text{HLD} = \text{on}$) defined by parameters hilim0 and lolim0 respectively or input-driven variables ($\text{HLD} = \text{off}$, 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 ($\text{HLD} = \text{on}$) using the hilim0 and lolim0 parameters. To allow immediate changes of the saturation limits, set $\text{tp} = 0$ and $\text{tn} = 0$. 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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u	Analog input of the block	Double (F64)
hi	Upper limit of the output signal	Double (F64)
lo	Lower limit of the output signal	Double (F64)

Parameter

tp	Rate limiter for positive limit changes	$\odot 1.0$	Double (F64)
tn	Rate limiter for negative limit changes	$\odot 1.0$	Double (F64)
hilim0	Upper limit of the output (valid when $\text{HLD}=1$)	$\odot 1.0$	Double (F64)
lolim0	Lower limit of the output (valid when $\text{HLD}=1$)	$\odot -1.0$	Double (F64)
HLD	Fixed saturation limits	$\odot \text{on}$	Bool
	$\text{off} \dots$ Variable saturation limits		
	$\text{on} \dots$ Fixed saturation limits		

Output

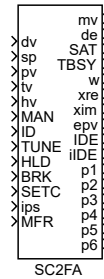
y	Analog output of the block	Double (F64)
-----	----------------------------	--------------

HL	Upper limit saturation indicator	Bool
LL	Lower limit saturation indicator	Bool

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

$$F(s) = \frac{b_1 s + b_0}{s^2 + 2\xi\Omega s + \Omega^2}, \quad (7.4)$$

where $\Omega > 0$ is the natural (undamped) frequency, ξ , $0 < \xi < 1$, 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".

Identification and design mode

The "Identification and design mode" is activated by the binary input **ID** = **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 **off**→**on** of the **TUNE** input. A harmonic signal with amplitude **uamp**, frequency ω and bias **ubias** then appears at the output **mv**. The frequency runs through the interval $\langle \mathbf{wb}, \mathbf{wf} \rangle$, it increases gradually. The current frequency is copied to the output **w**. The rate at which the frequency changes (sweeping) is determined by the **cp** parameter, which defines the relative shrinking of the initial period $T_b = \frac{2\pi}{\mathbf{wb}}$ of the exciting sine wave in time T_b , thus

$$c_p = \frac{\mathbf{wb}}{\omega(T_b)} = \frac{\mathbf{wb}}{\mathbf{wb}e^{\gamma T_b}} = e^{-\gamma T_b}.$$

The **cp** parameter usually lies within the interval $\mathbf{cp} \in (0,95;1)$. The lower the damping coefficient ξ of the controlled system is, the closer to one the **cp** parameter must be.

At the start of the identification, the exciting signal has a frequency $\omega = \mathbf{wb}$. After **stime** has elapsed, the calculation of the estimate of the current frequency characteristic point begins. Its real and imaginary parts are continuously copied in order to the outputs **xre** and **xim**. If the block parameter **MANF** is set to 0, then the frequency sweep stops twice for **stime** at the moments when points with phase delays **ph1** and **ph2** are first reached. The preset values for parameters **ph1** and **ph2** are respectively -60° and -120° , and they can be changed to any values in the interval $(-360^\circ, 0^\circ)$, where **ph1** > **ph2**. After **stime** seconds of stopping at phase **ph1**, or **ph2**, the average of the last **iavg** measured points is calculated (thus obtaining an estimate of the respective frequency characteristic point) for the subsequent calculation of the parametric model in the form (7.4). If **MANF** = **on**, it is possible to manually "sample" two points of the frequency characteristic using the input **HLD**. The input **HLD** = **on** stops the frequency sweep, and resetting **HLD** = **off** resumes it. Other functions are identical.

It is possible to terminate the identification experiment prematurely in case of necessity by the input **BRK** = **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 **IDE** = **on**.

During the actual "identification and design" process:

- the output **TBSY** is set to 1. After completion, it is reset to 0.
- If the controller design is error-free, the output **IDE** = **off** and the output **iIDE** indicates the individual phases of the identification experiment:
 - Approaching the first point is **iIDE** = -1,
 - stopping at the first point **iIDE** = 1,
 - approaching the second point is **iIDE** = -2,
 - stopping at the second point **iIDE** = 2, and
 - the final phase after stopping at the second point is **iIDE** = -3.
- If the identification ends with an error, then **IDE** = **on** and the number on the output **iIDE** specifies the corresponding error. See the description of the **iIDE** parameter below.

The computed state controller parameters are taken over by the control algorithm as soon as the **SETC** input is set to 1 (i.e. immediately if **SETC** is constantly set to **on**). The identified model and controller parameters can be obtained from the **p1**, **p2**, ..., **p6** outputs after setting the **ips** input to the appropriate value. For individual **ips** values, the parameters have the following meanings:

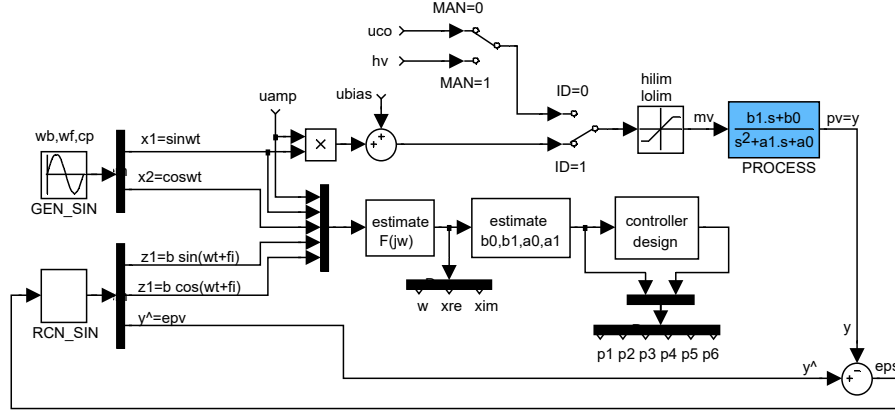
- **0**: Two points of frequency response
 - **p1** ... frequency of the 1st measured point in rad/s
 - **p2** ... real part of the 1st point

- p3 ... imaginary part of the 1st point
- p4 ... frequency of the 2nd measured point in rad/s
- p5 ... real part of the 2nd point
- p6 ... imaginary part of the 2nd point
- **1:** Second order model in the form (7.5)
 - p1 ... b_1 parameter
 - p2 ... b_0 parameter
 - p3 ... a_1 parameter
 - p4 ... a_0 parameter
- **2:** Second order model in the form (7.6)
 - p1 ... K_0 parameter
 - p2 ... τ parameter
 - p3 ... Ω parameter in rad/s
 - p4 ... ξ parameter
 - p5 ... Ω parameter in Hz
 - p6 ... resonance frequency in Hz
- **3:** State feedback parameters
 - p1 ... f_1 parameter
 - p2 ... f_2 parameter
 - p3 ... f_3 parameter
 - p4 ... f_4 parameter
 - p5 ... f_5 parameter

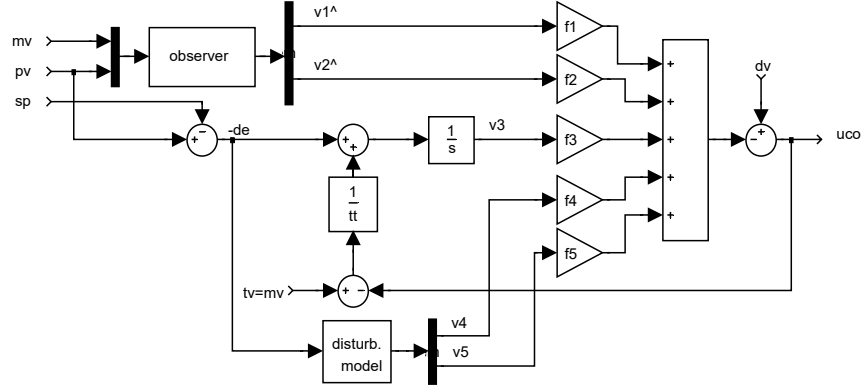
After a successful identification it is possible to generate the frequency response of the controlled system model, which is initiated by a rising edge at the **MFR** input. The frequency response can be read from the **w**, **xre** and **xim** outputs, which allows easy confrontation of the model and the measured data.

Controller

The "Controller mode" (binary input **ID** = **off**) has manual (**MAN** = **on**) and automatic (**MAN** = **off**) submodes. After a cold start of the block with the input **ID** = **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 **SETC** = **on**, then the control law uses the parameters from the very beginning. 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 $f1, f2, \dots, f5$ after identification experiment (and $SETC = on$).



The controlled system is assumed in the form of (7.4). Another forms of this transfer function are

$$F(s) = \frac{(b_1 s + b_0)}{s^2 + a_1 s + a_0} \quad (7.5)$$

and

$$F(s) = \frac{K_0 \Omega^2 (\tau s + 1)}{s^2 + 2\xi \Omega s + \Omega^2}. \quad (7.6)$$

The coefficients of these transfer functions can be found at the outputs $p1, \dots, p6$ after the identification experiment ($TBSY = off$). The output signals meaning is switched when a change occurs at the ips input.

This block propagates the signal quality. More information can be found in the 1.4 section.

Input

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	Bool
ID	Identification or controller operating mode off ... Controller mode on Identification and design mode	Bool
TUNE	Start the tuning experiment	Bool
HLD	Stop frequency sweeping	Bool
BRK	Termination signal	Bool
SETC	Accept and set the controller parameters off ... Parameters are only computed on Parameters are accepted as soon as computed off->on One-shot confirmation of the computed parameters	Bool
ips	Meaning of the output signals 0 Two points of frequency response 1 Second order model (general) 2 Second order model (frequency) 3 State feedback parameters	Long (I32)
MFR	Model frequency response generation	Bool

Parameter

ubias	Static component of the exciting signal	Double (F64)
uamp	Amplitude of the exciting signal	⊙1.0 Double (F64)
wb	Frequency interval lower limit [rad/s]	⊙1.0 Double (F64)
wf	Frequency interval upper limit [rad/s]	⊙10.0 Double (F64)
isweep	Frequency sweeping mode 1 Logarithmic 2 Linear 3 Combined	⊙1 Long (I32)
cp	Sweeping rate	↓0.5 ↑1.0 ⊙0.995 Double (F64)
iavg	Number of values for averaging	⊙10 Long (I32)
alpha	Relative positioning of the observer poles (ident.)	⊙2.0 Double (F64)
xi	Observer damping coefficient (ident.)	⊙0.707 Double (F64)
MANF	Manual frequency response points selection off ... Disabled on Enabled	Bool
ph1	Phase delay of the 1st point [degrees]	⊙-60.0 Double (F64)

ph2	Phase delay of the 2nd point [degrees]	⊖-120.0	Double (F64)
stime	Settling period [s]	⊖10.0	Double (F64)
ralpha	Relative positioning of the observer poles	⊖4.0	Double (F64)
rxl	Observer damping coefficient	⊖0.707	Double (F64)
acl1	Relative positioning of the 1st CL poles couple	⊖1.0	Double (F64)
xicl1	Damping of the 1st closed-loop poles couple	⊖0.707	Double (F64)
INTGF	Integrator flag	⊖on	Bool
	off ... State-space model without integrator		
	on Integrator included in the state-space model		
apcl	Relative position of the real pole	⊖1.0	Double (F64)
DISF	Disturbance flag		Bool
	off ... State space model without disturbance model		
	on Disturbance model is included in the state space model		
dom	Disturbance model natural frequency [rad/s]	⊖1.0	Double (F64)
dxi	Disturbance model damping coefficient		Double (F64)
acl2	Relative positioning of the 2nd CL poles couple	⊖2.0	Double (F64)
xicl2	Damping of the 2nd closed-loop poles couple	⊖0.707	Double (F64)
tt	Tracking time constant	⊖1.0	Double (F64)
hilim	Upper limit of the controller output	⊖1.0	Double (F64)
lolim	Lower limit of the controller output	⊖-1.0	Double (F64)
mb1p	Controlled system transfer function coefficient b1		Double (F64)
mb0p	Controlled system transfer function coefficient b0	⊖1.0	Double (F64)
ma1p	Controlled system transfer function coefficient a1	⊖0.2	Double (F64)
ma0p	Controlled system transfer function coefficient a0	⊖1.0	Double (F64)

Output

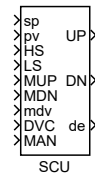
mv	Manipulated variable (controller output)	Double (F64)
de	Deviation error	Double (F64)
SAT	Saturation flag	Bool
	off ... The controller implements a linear control law	
	on The controller output is saturated	
TBSY	Tuner busy flag	Bool
	off ... Identification not running	
	on Identification in progress	
w	Frequency response point estimate - frequency [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)
IDE	Identification error indicator	Bool
	off ... Identification successful	
	on Identification error occurred	

iIDE	Error code	Long (I32)
	101 ... Sampling period too low	
	102 ... Error identifying frequency response point(s)	
	103 ... Output saturation occurred during experiment	
	104 ... Invalid process model	
p1..p6	Results of identification and design phase	Double (F64)

SCU – Step controller with position feedback

Block Symbol

Licence: [STANDARD](#)



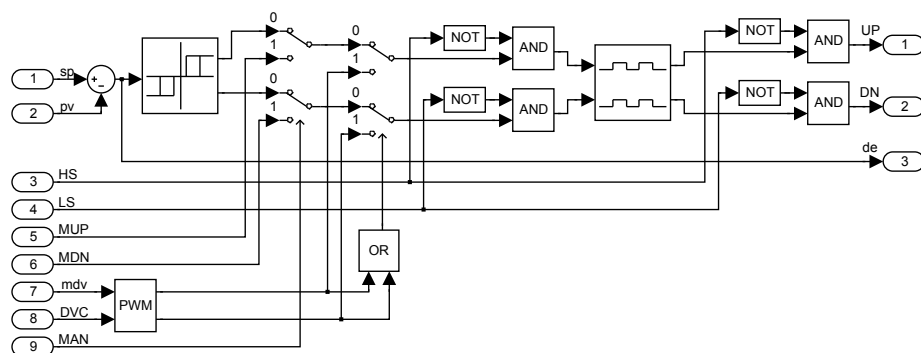
Function Description

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.

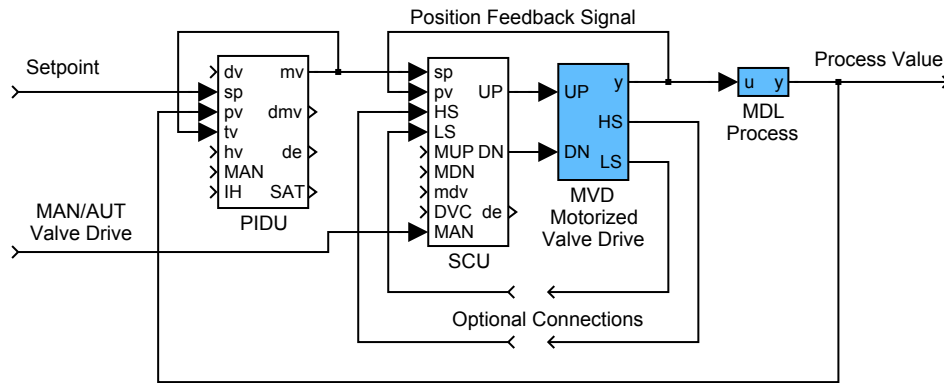
The **SCU** block processes the control deviation $sp - pv$ by a three state element with parameters (thresholds) **thron** and **throff** (see the **TSE** block, use parameters **ep** = **thron**, **epoff** = **throff**, **en** = **-thron** and **enoff** = **-throff**). The **trun** parameter specifies the time it takes for the motor position to change by one unit. 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 **dtime** and minimum pulse break time **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.



The complete structure of the three-state step controller is depicted in the following figure.



This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

sp	Setpoint (output of the primary controller)	Double (F64)
pv	Controlled variable (valve position)	Double (F64)
HS	Upper end switch	Bool
LS	Lower end switch	Bool
MUP	Manual UP signal	Bool
MDN	Manual DN signal	Bool
mdv	Manual differential value	Double (F64)
DVC	Differential value change command	Bool
MAN	Manual or automatic mode	Bool
	off ... Automatic mode	
	on Manual mode	

Parameter

thron	Switch-on value	↓0.0 ⊕0.02	Double (F64)
throff	Switch-off value	↓0.0 ⊕0.01	Double (F64)
dtime	Minimum width of the output pulse [s]	↓0.0 ⊕0.1	Double (F64)
btime	Minimum delay between output pulses [s]	↓0.0 ⊕0.1	Double (F64)
RACT	Reverse action flag		Bool
	off ... Higher mv -> higher pv		
	on Higher mv -> lower pv		
trun	Motor time constant	↓0.0 ⊕10.0	Double (F64)

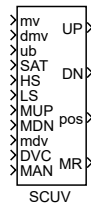
Output

UP	The UP signal (up, more)	Bool
DN	The DOWN signal (down, less)	Bool
de	Deviation error	Double (F64)

SCUV – Step controller unit with velocity input

Block Symbol

Licence: [STANDARD](#)



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 **mv**, **dmv** and **SAT**.

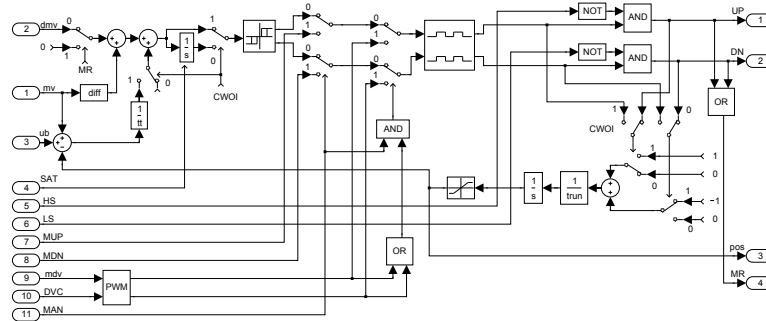
If the primary controller uses PI or PID control law (**CW0I** = **off**), then all three inputs **mv**, **dmv** and **SAT** of the block **SCUV** are sequentially processed by the special integration algorithm and by the three state element with parameters **thron** and **throff** (see the **TSE** block, 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 **mv** 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**.

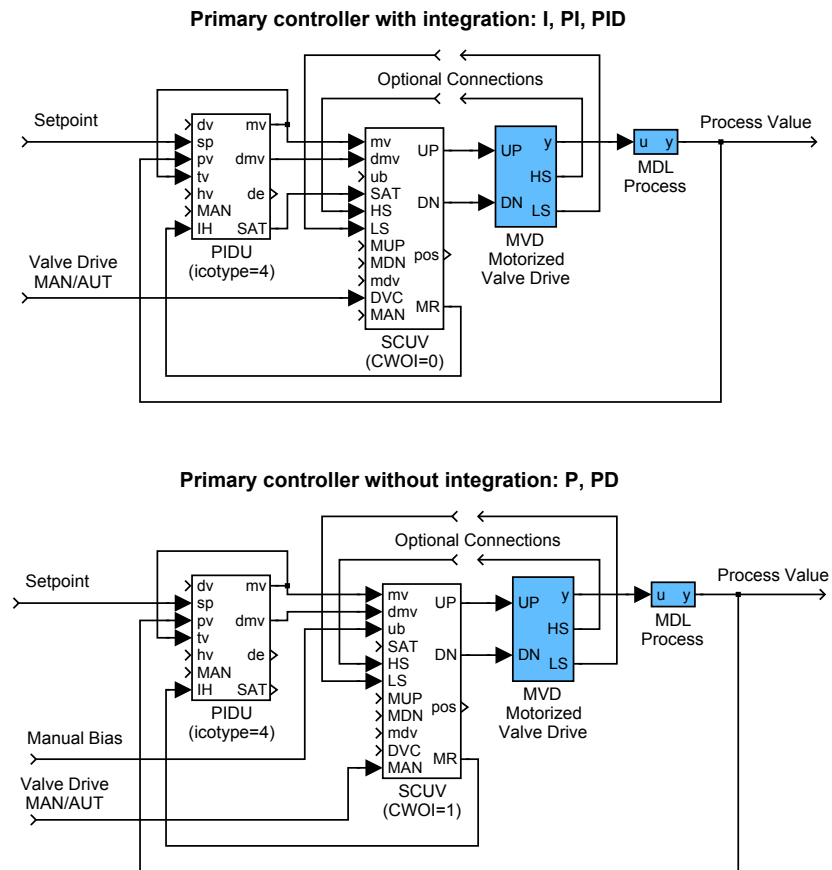
If the primary controller uses P or PD control law (**CW0I** = **on**), then the deviation error of the primary controller can be eliminated by the bias **ub** manually. In this case, the control algorithm is slightly modified, the position of the motor **pos** is used and the proper settings of **thron**, **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 **mdv** 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:



This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

mv	Manipulated variable (controller output)	Double (F64)
dmv	Controller velocity output (difference)	Double (F64)
ub	Bias (only for P or PD primary controller)	Double (F64)
SAT	Internal integrator reset	Bool
HS	Upper end switch	Bool
LS	Lower end switch	Bool
MUP	Manual UP signal	Bool
MDN	Manual DN signal	Bool
mdv	Manual differential value	Double (F64)
DVC	Differential value change command	Bool
MAN	Manual or automatic mode	Bool
	off ... Automatic mode	
	on Manual mode	

Parameter

thron	Switch-on value	$\downarrow 0.0 \odot 0.02$	Double (F64)
throff	Switch-off value	$\downarrow 0.0 \odot 0.01$	Double (F64)
dtime	Minimum width of the output pulse [s]	$\downarrow 0.0 \odot 0.1$	Double (F64)
btime	Minimum delay between output pulses [s]	$\downarrow 0.0 \odot 0.1$	Double (F64)
RACT	Reverse action flag		Bool
	off ... Higher mv -> higher pv		
	on Higher mv -> lower pv		
trun	Motor time constant	$\downarrow 0.0 \odot 10.0$	Double (F64)
CW0I	Controller without integration flag		Bool
	off ... Controller with integrator (I, PI, PID)		
	on Controller without integrator (P, PD)		
tt	Tracking time constant	$\downarrow 0.0 \odot 1.0$	Double (F64)

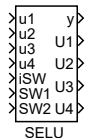
Output

UP	The UP signal (up, more)	Bool
DN	The DOWN signal (down, less)	Bool
pos	Position output of motor simulator	Double (F64)
MR	Request to move the motor	Bool
	off ... Motor idle (UP=off and DN=off)	
	on Request to move (UP=on or DN=on)	

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:

iSW	SW1	SW2	y	U1	U2	U3	U4
0	off	off	u1	off	on	on	on
1	off	on	u2	on	off	on	on
2	on	off	u3	on	on	off	on
3	on	on	u4	on	on	on	off

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).

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u1..u4	Analog input of the block	Double (F64)
iSW	Active signal selector	Long (I32)
SW1	Binary signal selector	Bool
SW2	Binary signal selector	Bool

Parameter

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

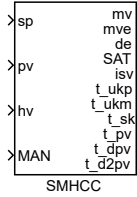
Output

y	The selected input signal	Double (F64)
U1..U4	Binary output signal for selector control	Bool

SMHCC – Sliding mode heating/cooling controller

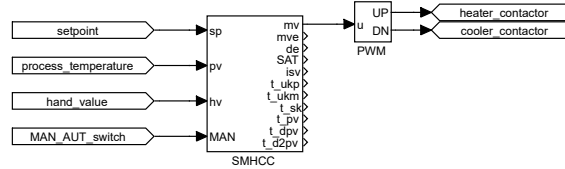
Block Symbol

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 several 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 $T_C = i_{pwm} T_S$ 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 **mv** 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_{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:

$$T_S = 0.1, i_{pwm} = 100, T_C = 10s, T_R = 0.01s.$$

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

$$s_k \triangleq \ddot{e}_k + 2\xi\Omega\dot{e}_k + \Omega^2 e_k$$

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 ($\mathbf{pv} - \mathbf{sp}$) and its first and second derivatives in the control period k , respectively, and ξ, Ω 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

$$s \triangleq \ddot{e} + 2\xi\Omega\dot{e} + \Omega^2 e$$

is approximately equal to zero at any time. Therefore the control deviation behaves according to the second order differential equation

$$s \triangleq \ddot{e} + 2\xi\Omega\dot{e} + \Omega^2 e = 0$$

describing so called *zero sliding dynamics*. From it follows that the evolution of e can be prescribed by the parameters ξ and Ω . For stable behavior, it must hold $\xi > 0$ and $\Omega > 0$. A typical optimal value of ξ ranges in the interval $[0.1, 8]$ and ξ about 6 is often a satisfactory value. The optimal value of Ω strongly depends on the controlled process. The slower processes the lower optimal Ω . The recommended value of Ω for start of tuning is $\pi/(5T_C)$.

The manipulated variable \mathbf{mv} usually ranges in the interval $[-1, 1]$. The positive (negative) value corresponds to heating (cooling). For example, $\mathbf{mv} = 1$ means the full heating. The limits of \mathbf{mv} can be reduced when needed by the controller parameters `hilim_p` and `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 `hilim_p` = 1 and `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 `u0_p` (`u0_m`) to the suitable value less than `hilim_p` (`hilim_m`). Otherwise set `u0_p` = `hilim_p` and `u0_m` = `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 `isv` alternates the values 1 and -1 . The rate of adaptation of the heating (cooling) amplitude is given by the time constant `taup` (`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 \mathbf{mv} is determined from the action amplitudes `uk_p`, `uk_m` by the following expression

$$\text{if } (s_k < 0.0) \text{ then } \mathbf{mv} = \mathbf{t_ukp} \text{ else } \mathbf{mv} = -\mathbf{t_ukm} .$$

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 (**MAN = on**) the controller input **hv** is (after limitation to the range $[-\text{hilim_m}, \text{hilim_p}]$) copied to the manipulated variable **mv**. The controller output **mve** provides the equivalent amplitude-modulated value of the manipulated variable **mv** for informative purposes. The output **mve** is obtained by the first order filter with the time constant **tauf** applied to **mv**.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

sp	Setpoint variable	Double (F64)
pv	Process variable	Double (F64)
hv	Manual value	Double (F64)
MAN	Manual or automatic mode	Bool
	off ... Automatic mode	
	on Manual mode	

Parameter

ipwmc	PWM cycle (in sampling periods of the block)	⊙100	Long (I32)
xi	Relative damping of sliding zero dynamics	⊙1.0	Double (F64)
om	Natural frequency of sliding zero dynamics	⊙0.01	Double (F64)
taup	Time constant for adaptation - heating [s]	⊙700.0	Double (F64)
taum	Time constant for adaptation - cooling [s]	⊙400.0	Double (F64)
beta	Bandwidth parameter of the derivative filter	⊙0.01	Double (F64)
hilim_p	Upper limit of the heating action amplitude	⊙1.0	Double (F64)
hilim_m	Upper limit of the cooling action amplitude	⊙1.0	Double (F64)
u0_p	Initial amplitude - heating action	⊙1.0	Double (F64)
u0_m	Initial amplitude - cooling action	⊙1.0	Double (F64)
sp_dif	Setpoint difference threshold	⊙10.0	Double (F64)
tauf	Equivalent manipulated variable filter time constant	⊙400.0	Double (F64)

Output

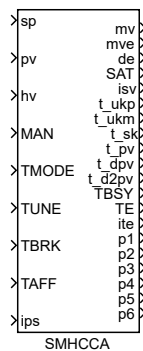
mv	Manipulated variable (controller output)	Double (F64)
mve	Equivalent manipulated variable	Double (F64)
de	Deviation error	Double (F64)
SAT	Saturation flag	Bool
	off ... The controller implements a linear control law	
	on The controller output is saturated	

isv	Number of 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	Double (F64)
t_dpv	Filtered first derivative of process variable	Double (F64)
t_d2pv	Filtered second derivative of process variable	Double (F64)

SMHCCA – Sliding mode heating/cooling controller with auto-tuner

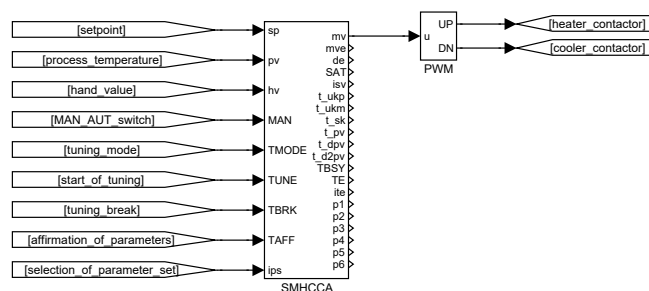
Block Symbol

Licence: [AUTOTUNING](#)



Function Description

The functional block **SMHCCA** (Sliding Mode Heating/Cooling Controller with Autotuner) is a high-quality control algorithm with a built-in autotuner for automatic tuning of the controller parameters. The controller is an easily adjustable controller for quality control of thermal systems with two-state (ON-OFF) heating and two-state (ON-OFF) cooling. A classic example of such systems is the plastic extruder. However, it can of course also be deployed on other systems where conventional thermostats are commonly used so far. To ensure proper function, the **SMHCCA** block must be supplemented by the **PWM** block (Pulse Width Modulation), as is evident from the following figure.



Operating Principles

It's important to realize that the **SMHCCA** block operates with several time periods. The *first* period T_S is the sampling period of the measured temperature and is also equal to the period with which the **SMHCCA** controller block is executed. The *second* period $T_C = i_{pwm} T_S$ is the control period with which the **SMHCCA** block generates the manipulated variable. This period T_C is identical to the cycle period of the **PWM** block. At every instant

when the manipulated variable mv of the **SMHCCA** block changes, the **PWM** block algorithm recalculates the pulse width and starts a new **PWM** cycle. The *third* period that needs to be set is the triggering period T_R of the **PWM** block. Generally, T_R may be different from T_S . To achieve the best control quality, it is recommended to set the period T_S to the minimum possible value (i_{pwmc} to the maximum possible value), the ratio T_C/T_S maximum, but T_C should be sufficiently small with respect to the process dynamics. For applications in the plastics industry, the following values are recommended:

$$T_S = 0.1, \quad i_{pwmc} = 50, \quad T_C = 5s, \quad T_R = 0.1s.$$

Note, however, that for a faster controlled system, the sampling periods T_S , T_C , and T_R must be shortened! More precisely, the three minimum time constants of the process are important for selecting these time periods (all real thermal processes have 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 current 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 , T_C , T_R and the parameter **beta** is critical for the 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**, **om**, **taup**, **taum**, **tauf**) can be set automatically by the built-in autotuner.

Automatic Tuning Mode

The autotuner uses two methods for this purpose:

- The first one is intended for situations where the process asymmetry is not too large (approximately, this means that the gain ratio of heating/cooling or cooling/heating is less than 5).
- The second method provides tuning support for strongly asymmetric processes and is not yet implemented (So far, this method has been developed and tested in Simulink only).

Despite the fact that the first method of tuning is based only on the heating mode, the resulting parameters are usually satisfactory for both heating and cooling modes due to the strong robustness of sliding mode control. The tuning procedure is very quick and can be completed during the normal rise time of the process temperature from a cold state to the setpoint usually without any delay or degradation of control performance. Thus, the tuning procedure can be included in every start-up from a cold state to a working point specified by a sufficiently high temperature.

Now, the implemented procedure will be described in detail:

- The tuning procedure begins in tuning mode or in manual mode. If the tuning mode (**TMODE** = **on**) is selected, the manipulated variable mv is automatically set to

zero, and the output **TBSY** is set to **on** to indicate the tuning phase of the controller. The cold state of the process is preserved until a rising edge **off**→**on** is indicated at the **TUNE** input.

- After some time (dependent on **beta**), when the noise amplitude is estimated, heating is turned on with the amplitude given by the **ut_p** parameter. The process temperature **pv** and its two derivatives (outputs **t_pv**, **t_dpv**, **t_d2pv**) are observed to obtain the optimal controller parameters.
- If the tuning procedure ends without errors, then **TBSY** is set to **off**, and the controller begins to operate in manual or automatic mode according to the **MAN** input. If **MAN** = **off** and the confirmation input **TAFF** is set to **on**, then the controller begins to operate in automatic mode with the new set of parameters provided by tuning (if **TAFF** = **off**, then the new parameters are only displayed on the outputs **p1**...**p6**).
- If an error occurs during tuning, then the tuning procedure stops immediately or stops after the condition **pv>sp** is met, the output **TE** is set to **on**, and **ite** indicates the type of error. Also in this case, the controller begins to operate in the mode determined by the **MAN** input. If **MAN** = **off**, then it operates in automatic mode with the original parameters before tuning!
- Tuning errors are usually caused by either inappropriate setting of the **beta** parameter or too low a value of **sp**. The suitable value of **beta** ranges in the interval (0.001,0.1). If drift and noise in **pv** are large, a small **beta** value 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** can sometimes be slightly increased.
- The tuning procedure can also be started from manual mode (**MAN** = **off**) with any constant value of the **hv** input. However, a steady state must be ensured in this case. Again, tuning is initiated by an upward edge at the **TUNE** input, and after tuning stops, the controller continues in manual mode. In both cases, the resulting parameters appear on the outputs **p1**, ..., **p6**.

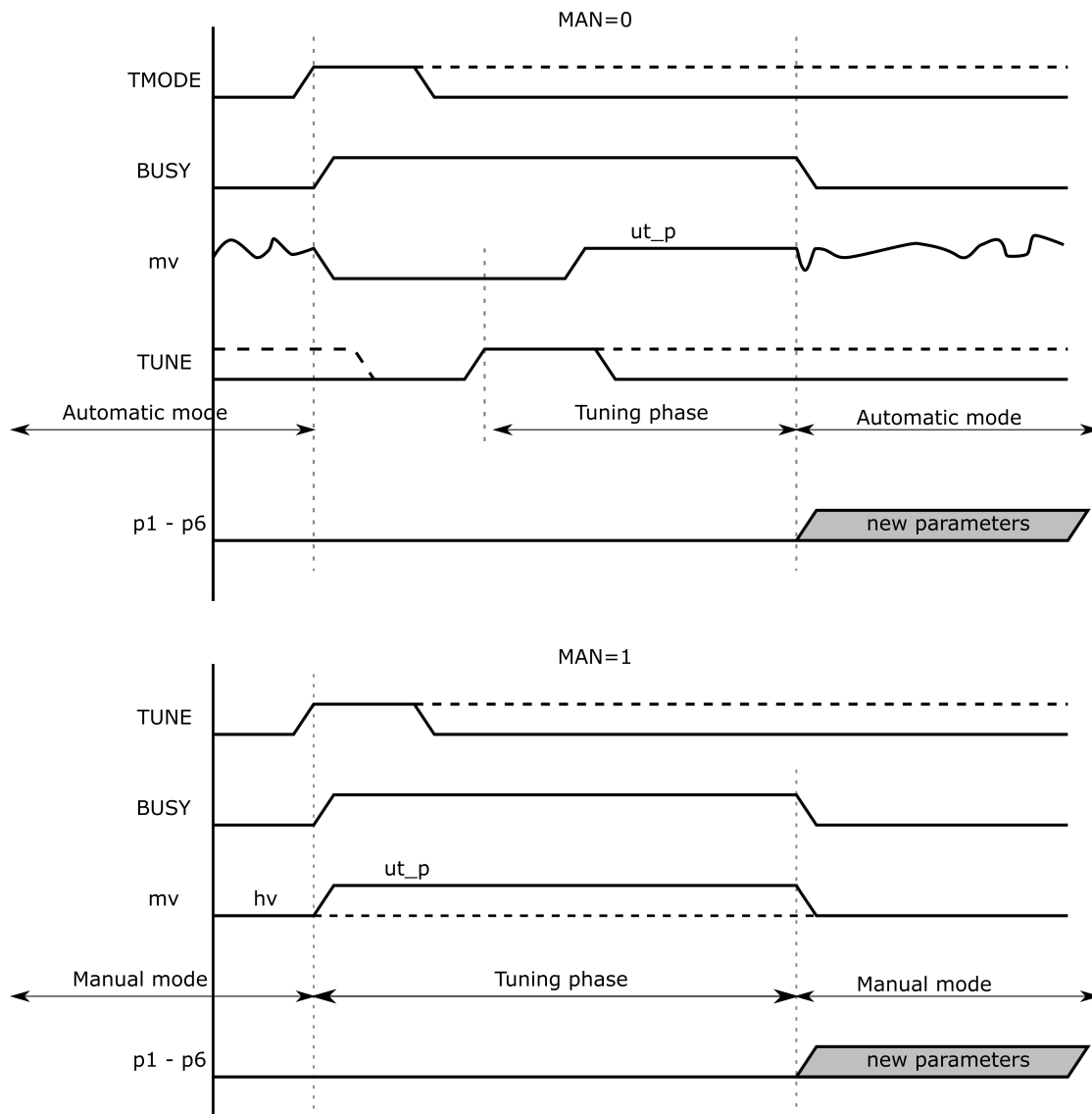
For individual **ips** values, the parameters **p1**, ..., **p6** have the following meanings:

- **0**: Controller parameters
 - **p1** ... recommended control period T_C
 - **p2** ... **xi**
 - **p3** ... **om**
 - **p4** ... **taup**
 - **p5** ... **taum**

– p6 ... `tauf`

• **1:** Auxiliary parameters

- p1 ... `htp2` – 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 ... `tgain`



Automatic mode

The control law of the **SMHCCA** block in automatic mode (**MAN=off**) is based on the discrete dynamic sliding mode control technique and employs a special third-order filter for estimating the first and second derivatives of the control error.

After a setpoint change or upset, the controller enters the first phase, the *reaching phase*, where the discrete sliding variable

$$s_k \triangleq \ddot{e}_k + 2\xi\Omega\dot{e}_k + \Omega^2 e_k$$

is forced to zero. In this definition, e_k , \dot{e}_k , \ddot{e}_k denote the filtered deviation error (**pv – sp**), its first and second derivatives at time k , respectively. The parameters ξ and Ω are described below. In the second phase, the quasi *sliding mode*, the variable s_k is kept near zero value through appropriate control actions, alternating between heating and cooling modes. The amplitudes for heating and cooling are adapted to approximately achieve $s_k = 0$. Consequently, the hypothetical continuous sliding variable

$$s \triangleq \ddot{e} + 2\xi\Omega\dot{e} + \Omega^2 e$$

remains approximately zero at all times. In other words, the control deviation e is described by a second-order differential equation

$$s \triangleq \ddot{e} + 2\xi\Omega\dot{e} + \Omega^2 e = 0.$$

This implies that the evolution of e can be influenced by choosing the parameters ξ and Ω . Note that for stable behavior, it is required that $\xi > 0$ and $\Omega > 0$. The typical optimal value of ξ lies in the range $[0.1, 8]$. The optimal value of Ω is strongly dependent on the controlled process; slower processes have a lower optimal Ω , and faster ones have a higher. The recommended value of Ω for the start of tuning parameters is $\pi/(5T_C)$.

The manipulated variable **mv** typically ranges from $[-1, 1]$. A positive value corresponds to heating, a negative to cooling, e.g., **mv** = 1 means full heating. The limits on **mv** can be set by the parameters **hilim_p** and **hilim_m**. This limitation may be necessary when there is a significant asymmetry between heating and cooling. For example, if cooling is much more aggressive than heating in the working zone, it is appropriate to set **hilim_p** = 1 and **hilim_m** < 1. If such limitation is only to be applied in some time interval after a change of setpoint (during the transient response), the initial values of the heating (cooling) action amplitude **u0_p** and **u0_m** should be set such that **u0_p** ≤ **hilim_p** and **u0_m** ≤ **hilim_m**.

The amplitudes of heating and cooling variables **t_ukp** and **t_ukm**, respectively, are automatically adapted by a special algorithm to achieve a quasi-sliding mode, where the signs of **sk** alternate at each step. In this case, the controller output **isv** switches between 1 and –1. The rate of adaptation of heating and cooling amplitudes is given by the time constants **taup** and **taum**. Both of these time constants must be sufficiently large to ensure the proper functioning of adaptation, but fine-tuning is not essential for the

final quality of regulation. For completeness, **mv** is determined based on the amplitudes **t_ukp** and **t_ukm** according to the following expression:

$$\text{if } (sk < 0.0) \text{ then } mv = t_ukp \text{ else } mv = -t_ukm.$$

It is also worth mentioning that achieving quasi-sliding mode occurs very rarely because controlled processes contain transport delays and are subject to disturbances. A suitable indicator of the quality of sliding is again the output **isv**. For fine-tuning, it may be possible in exceptional cases to use the **beta** parameter defining the bandwidth of the derivative filter. In most cases, however, the preset value **beta** = 0.1 suffices. In manual mode (**MAN** = **on**), the controller input **hv** is copied (after possible limitation by saturation limits **[-hilim_m, hilim_p]**) to the output **mv**.

Manual mode

In the manual mode (**MAN** = **on**) the controller input **hv** is (after limitation to the range **[-hilim_m, hilim_p]**) copied to the manipulated variable **mv**. The controller output **mve** provides the equivalent amplitude-modulated value of the manipulated variable **mv** for informative purposes. The output **mve** is obtained by the first order filter with the time constant **tauf** applied to **mv**.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

sp	Setpoint variable	Double (F64)
pv	Process variable	Double (F64)
hv	Manual value	Double (F64)
MAN	Manual or automatic mode	Bool
	off ... Automatic mode	
	on Manual mode	
TMODE	Tuning mode	Bool
TUNE	Start the tuning experiment	Bool
TBRK	Stop the tuning experiment	Bool
TAFF	Tuning affirmation	Bool
	off ... Parameters are only computed	
	on Parameters are set into the control law	
ips	Meaning of the output signals	Long (I32)
	0 Controller parameters	
	1 Auxiliary parameters	

Parameter

ipwmc	PWM cycle (in sampling periods of the block)	⊙100	Long (I32)
xi	Relative damping of sliding zero dynamics	↓0.5 ↑8.0 ⊙1.0	Double (F64)
om	Natural frequency of sliding zero dynamics	↓0.0 ⊙0.01	Double (F64)

taup	Time constant for adaptation - heating [s]	⊙700.0	Double (F64)
taum	Time constant for adaptation - cooling [s]	⊙400.0	Double (F64)
beta	Bandwidth parameter of the derivative filter	⊙0.01	Double (F64)
hilim_p	Upper limit of the heating action amplitude	↓0.0 ↑1.0 ⊙1.0	Double (F64)
hilim_m	Upper limit of the cooling action amplitude	↓0.0 ↑1.0 ⊙1.0	Double (F64)
u0_p	Initial amplitude - heating action	⊙1.0	Double (F64)
u0_m	Initial amplitude - cooling action	⊙1.0	Double (F64)
sp_dif	Setpoint difference threshold	⊙10.0	Double (F64)
tauf	Equivalent manipulated variable filter time constant	⊙400.0	Double (F64)
itm	Tuning method	⊙1	Long (I32)
	1 Restricted to symmetrical processes		
	2 Asymmetrical processes (not implemented yet)		
ut_p	Amplitude of heating for tuning experiment	↓0.0 ↑1.0 ⊙1.0	Double (F64)
ut_m	Amplitude of cooling for tuning experiment	↓0.0 ↑1.0 ⊙1.0	Double (F64)

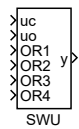
Output

mv	Manipulated variable (controller output)	Double (F64)
mve	Equivalent manipulated variable	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	
isv	Number of 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	Double (F64)
t_dpv	Filtered first derivative of process variable	Double (F64)
t_d2pv	Filtered second derivative of process variable	Double (F64)
TBSY	Tuner busy flag	Bool
TE	Tuning error	Bool
	off ... Autotuning successful	
	on ... An error occurred during the experiment	
ite	Error code	Long (I32)
	0 No error	
	1 Too noisy pv, check the temperature input	
	2 Incorrect parameter ut_p	
	3 Setpoint too low	
	4 Sampling frequency too low or 2nd derivative of pv too noisy	
	5 Premature termination of the tuning procedure	
p1..p6	Results of identification and design phase	Double (F64)

SWU – Switch unit

Block Symbol

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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uc	Signal valid for all ORi =0	Double (F64)
uo	Signal valid for any ORi =1	Double (F64)
OR1..OR4	Logical output of the block	Bool

Output

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

TSE – Three-state element

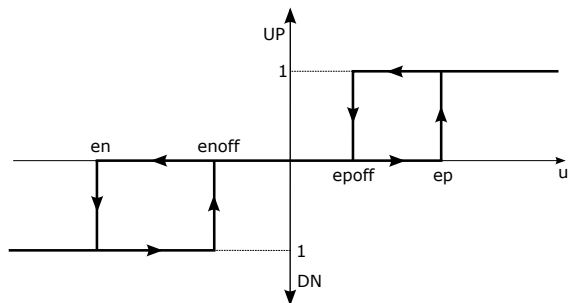
Block Symbol

Licence: [STANDARD](#)



Function Description

The **TSE** block transforms the analog input **u** to a three-state signal ("up", "idle" and "down") according to the diagram below.



This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

ep	UP switch on value	⊙1.0	Double (F64)
en	DN switch on value	⊙-1.0	Double (F64)
epoff	UP switch off value	⊙0.5	Double (F64)
enoff	DN switch off value	⊙-0.5	Double (F64)

Output

UP	The UP signal (up, more)	Bool
DN	The DOWN signal (down, less)	Bool

Chapter 8

LOGIC – Logic control

Contents

AND – Logical product of two signals	284
ANDQUAD, ANDOCT, ANDHEXD – Multi-input logical product	285
ATMT – Finite-state automaton	286
BDOCT, BDHEXD – Bitwise demultiplexers	288
BITOP – Bitwise operation	289
BMOCT, BMHEXD – Bitwise multiplexers	290
COUNT – Controlled counter	291
EATMT – Extended finite-state automaton	293
EDGE – Falling/rising edge detection in a binary signal	296
EQ – Equivalence two signals	297
INTSM – Integer number bit shift and mask	298
ISSW – Simple switch for integer signals	299
ITOI – Transformation of integer and binary numbers	300
NOT – Boolean complementation	301
OR – Logical sum of two signals	302
ORQUAD, OROCT, ORHEXD – Multi-input logical sum	303
RS – Reset-set flip-flop circuit	304
SR – Set-reset flip-flop circuit	305
TIMER – Multipurpose timer	306

The LOGIC library encompasses a range of blocks for executing logical and sequential operations. It includes basic Boolean blocks like [AND](#), [OR](#), [NOT](#) for fundamental logical operations, and advanced blocks like [ATMT](#) for finite state machines. Blocks like [COUNT](#) and [TIMER](#) extend functionality to bidirectional pulse counting and time-based operations. Additional elements like [BITOP](#), [BMOCT](#), and [BDOCT](#) offer bitwise operations and multiplexing/demultiplexing capabilities, enhancing the library's versatility in handling combinational and sequential logic control.

AND – Logical product of two signals

Block Symbol

Licence: [STANDARD](#)



Function Description

The **AND** block computes the logical product of two input signals $Y = U1 \wedge U2$. If you need to work with more input signals, use the [ANDQUAD](#), [ANDDOCT](#) or [ANDHEXD](#) block.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

U1	First logical input of the block	Bool
U2	Second logical input of the block	Bool

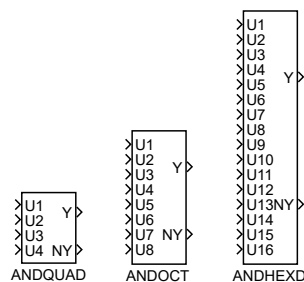
Output

Y	Output signal, logical product	Bool
NY	Boolean complementation of Y	Bool

ANDQUAD, ANDOCT, ANDHEXD – Multi-input logical product

Block Symbols

Licence: [STANDARD](#)



Function Description

The **ANDQUAD**, **ANDOCT** and **ANDHEXD** blocks compute the logical product of up to sixteen input signals $U1, U2, \dots, U16$. The signals listed in the **n1** parameter are negated prior to computing the logical product.

For an empty **n1** parameter a simple logical product $Y = U1 \wedge U2 \wedge U3 \wedge U4 \wedge U5 \wedge U6 \wedge U7 \wedge U8$ is computed. For e.g. **n1=1,3..5**, the logical function is $Y = \neg U1 \wedge U2 \wedge \neg U3 \wedge \neg U4 \wedge \neg U5 \wedge U6 \wedge \dots U16$.

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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

U1..U16	Logical input of the block	Bool
----------------	----------------------------	-------------

Parameter

n1	List of signals to negate	Long (I32)
-----------	---------------------------	-------------------

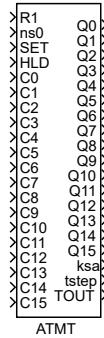
Output

Y	Output signal, logical product	Bool
NY	Boolean complementation of Y	Bool

ATMT – Finite-state automaton

Block Symbol

Licence: [STANDARD](#)



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 , $i = 0, 1, \dots, 15$ is indicated by the binary outputs $Q0, Q1, \dots, Q15$. If the state i is active, the corresponding output is set to $Qi=on$. The current state is also indicated by the **ksa** output ($ksa \in \{0, 1, \dots, 15\}$).

The transition conditions C_k , $k = 0, 1, \dots, 15$ are activated by the binary inputs $C0, C1, \dots, C15$. If $Ck = on$ the k -th transition condition is fulfilled. The transition cannot happen when $Ck = off$.

The automat function is defined by the following table of transitions:

$S1$	$C1$	$FS1$
$S2$	$C2$	$FS2$
		\dots
Sn	Cn	FSn

Each row of this table represents one transition rule. For example the first row

$$S1 \quad C1 \quad FS1$$

has the meaning

If ($S1$ is the current state AND transition condition $C1$ is fulfilled),
 then proceed to the following state $FS1$.

The above mentioned table can be easily constructed from the automat state diagram or SFC description (Sequential Function Charts, formerly Grafcet).

The $R1 = on$ input resets the automat to the initial state $S0$. 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 automat activity, the automat 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 automat changes its state.

It is possible to allow more state transitions in one cycle by the moresteps 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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

R1	Block reset	Bool
ns0	Target state forced by the SET input	Long (I32)
SET	Forced transition to state ns0	Bool
HLD	Hold	Bool
C0..C15	Transition condition	Bool

Parameter

moresteps	Allow multiple transitions in one cycle off ... Disabled on Enabled	Bool
sfcname	Name of special editor data file	String
STT	State transition table ⊙[0 0 1; 1 1 2; 2 2 3; 3 3 0]	Byte (U8)
touts	Vector of timeouts ⊙[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16]	Double (F64)

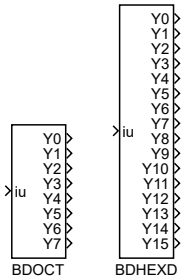
Output

Q0..Q15	Active state indicator	Bool
ksa	Integer code of the active state	Long (I32)
tstep	Time elapsed since the last state transition	Double (F64)
TOUT	Timeout flag	Bool

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** shifter by **shift** bits to the right. The **Y0** output is the least significant bit.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

iu	Input signal to be decomposed	Long (I32)
-----------	-------------------------------	------------

Parameter

shift	Bit shift of the input signal	↓0 ↑31	Long (I32)
vtype	Numeric type	⊙4	Long (I32)
	2 Byte (U8)		
	3 Short (I16)		
	4 Long (I32)		
	5 Word (U16)		
	6 DWord (U32)		
	10 Large (I64)		

Output

Y0..Y15	Individual bit of the input signal	Bool
----------------	------------------------------------	------

BITOP – Bitwise operation

Block Symbol

Licence: [STANDARD](#)



Function Description

The **BITOP** block performs bitwise operation $i1 \circ i2$ on the signals **i1** and **i2**, 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 **i2** is ignored (i.e. the operation is unary).

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

i1	First integer input of the block	Long (I32)
i2	Second integer input of the block	Long (I32)

Parameter

iop	Bitwise operation	⊙1 Long (I32)
	1 Bit NOT	
	2 Bit OR	
	3 Bit AND	
	4 Bit XOR	
	5 Shift Left	
	6 Shift Right	
	7 2's Complement - Byte	
	8 2's Complement - Word	
	9 2's Complement - Long	
vtype	Numeric type	⊙4 Long (I32)
	2 Byte (U8)	
	3 Short (I16)	
	4 Long (I32)	
	5 Word (U16)	
	6 DWord (U32)	
	10 Large (I64)	

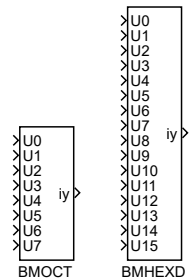
Output

n	Result of the bitwise operation	Long (I32)
----------	---------------------------------	------------

BMOCT, BMHEXD – Bitwise multiplexers

Block Symbols

Licence: [STANDARD](#)



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. If the parameter **shift** = 0, the individual bits of the output signal **iy** are directly formed by the input signals **Ui**. The **U0** output is the least significant bit.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

U0..U15	Individual bit of the output signal	Bool
---------	-------------------------------------	------

Parameter

shift	Bit shift of the output signal	↓0 ↑31	Long (I32)
vtype	Numeric type	⊙4	Long (I32)
	2		Byte (U8)
	3		Short (I16)
	4		Long (I32)
	5		Word (U16)
	6		DWord (U32)
	10		Large (I64)

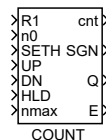
Output

iy	Composed output signal	Long (I32)
----	------------------------	------------

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 **cnt** output is incremented (decremented) by 1. Simultaneous occurrence of rising edges at both inputs is indicated by the error output **E** set to **on**. The **R1** input resets the counter to 0 and no addition or subtraction is performed unless the **R1** input returns to **off** again. It is also possible to set the output **cnt** to the value **n0** by the **SETH** input. Again, no addition or subtraction is performed unless the **SETH** input returns to **off** again. The **R1** input has higher priority than the **SETH** input. The input **HLD** = **on** prevents both incrementing and decrementing. When the counter reaches the value $\text{cnt} \geq \text{nmax}$, the **Q** output is set to **on**.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

R1	Block reset	Bool
n0	Value to set the counter to	Long (I32)
SETH	Set the counter value	Bool
UP	Incrementing input signal	Bool
DN	Decrementing input signal	Bool
HLD	Counter freeze	Bool
	off ... Counter is running	
	on Counter is locked	
nmax	Counter target value	Long (I32)

Output

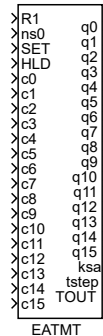
cnt	Total number of pulses	Long (I32)
SGN	Sign of the cnt output	Bool
	off ... Less or equal to zero	
	on Positive value	

Q	Counter state	Bool
	off ... Target value not reached	
	on Target value reached	
E	Error indicator	Bool
	off ... No error	
	on An error occurred	

EATMT – Extended finite-state automaton

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **EATMT** block implements a finite automaton 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 , $i = 0, 1, \dots, 255$ is indicated by individual bits of the integer outputs $q0, q1, \dots, q15$. Only a single bit with index $i \bmod 16$ of the $q(i \text{ DIV } 16)$ 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 $ksa \in \{0, 1, \dots, 255\}$ output.

The transition conditions C_k , $k = 0, 1, \dots, 255$ are activated by individual bits of the inputs $c0, c1, \dots, c15$. The k -th transition condition is fulfilled when the $(k \bmod 16)$ -th bit of the input $c(k \text{ DIV } 16)$ is equal to 1. The transition cannot happen otherwise.

The [BMHEXD](#) or [BMOCT](#) bitwise multiplexers can be used for composition of the input signals $c0, c1, \dots, c15$ from individual Boolean signals. Similarly the output signals $q0, q1, \dots, q15$ can be decomposed using the [BDHEXD](#) or [BDOCT](#) bitwise demultiplexers.

The automaton function is defined by the following table of transitions:

$S1$	$C1$	$FS1$
$S2$	$C2$	$FS2$
		\dots
Sn	Cn	FSn

Each row of this table represents one transition rule. For example the first row

$$S1 \quad C1 \quad FS1$$

has the meaning

If ($S1$ is the current state AND transition condition $C1$ is fulfilled),
 then proceed to the following state $FS1$.

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 = on** input resets the automat to the initial state $S0$. 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 automat activity, the automat 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 automat changes its state.

It is possible to allow more state transitions in one cycle by the **moresteps** 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 REXYGEN 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 **EATMT** block.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

R1	Block reset	Bool
ns0	Target state forced by the SET input	Long (I32)
SET	Forced transition to state ns0	Bool
HLD	Hold	Bool
c0..c15	Transition condition	Long (I32)

Parameter

moresteps	Allow multiple transitions in one cycle off ... Disabled on Enabled	Bool
sfcname	Name of special editor data file	String
STT	State transition table $\odot[0\ 0\ 1; 1\ 1\ 2; 2\ 2\ 3; 3\ 3\ 0]$	Short (I16)
touts	Vector of timeouts $\odot[1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16]$	Double (F64)

Output

<code>q0..q15</code>	Active state indicator	Long (I32)
<code>ksa</code>	Integer code of the active state	Long (I32)
<code>tstep</code>	Time elapsed since the last state transition	Double (F64)
<code>TOUT</code>	Timeout flag	Bool

EDGE – Falling/rising edge detection in a binary signal

Block Symbol

Licence: [STANDARD](#)



Function Description

The **EDGE** block detects rising (**off**→**on**), falling (**on**→**off**), or both edges on the input signal **U**, depending on the value of the **iedge** parameter. In case the desired edge (change in input signal) is found, the output **Y** is set to **on** for one step. As long as the value of the input signal remains unchanged, the output **Y** equals **off**. The output **Y** will also remain zero if the **iedge** parameter is set to detect a rising (falling) edge and a falling (rising) edge occurs in the signal.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

U	Logical input of the block	Bool
---	----------------------------	------

Parameter

iedge	Type of edges to detect	⊙1 Long (I32)
1 Rising edge	
2 Falling edge	
3 Both edges	

Output

Y	Logical output of the block	Bool
---	-----------------------------	------

EQ – Equivalence two signals

Block Symbol

Licence: [STANDARD](#)



Function Description

The block compares two input signals and **Y=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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u1	Block input signal	Any
u2	Block input signal	Any

Output

Y	Logical output of the block	Bool
NY	Boolean complementation of Y	Bool

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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

`i` Integer value to shift and mask ↓-9.22337E+18 ↑9.22337E+18 Large (I64)

Parameter

<code>shift</code>	Bit shift (negative=left, positive=right)	↓-63 ↑63	Long (I32)
<code>mask</code>	Bit mask (applied after bit shift)		Large (I64)
		↓0 ↑4294970000 ⊕4294967295	
<code>vtype</code>	Numeric type	⊕4	Long (I32)
	2 Byte (U8)		
	3 Short (I16)		
	4 Long (I32)		
	5 Word (U16)		
	6 DWord (U32)		
	10 Large (I64)		

Output

`n` Resulting integer value 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 `i1` and `i2` whose decision variable is the binary input `SW`. If `SW` is `off`, then the output `n` is equal to the `i1` signal. If `SW` is `on`, then the output `n` is equal to the `i2` signal.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

<code>i1</code>	First integer input of the block	Long (I32)
<code>i2</code>	Second integer input of the block	Long (I32)
<code>SW</code>	Signal selector	Bool
	<code>off</code> ... The <code>i1</code> signal is selected	
	<code>on</code> The <code>i2</code> signal is selected	

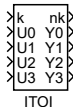
Output

<code>n</code>	Integer output of the block	Long (I32)
----------------	-----------------------------	------------

ITOI – Transformation of integer and binary numbers

Block Symbol

Licence: STANDARD



Function Description

The ITOI block transforms the input number k , or the binary number $(U3\ U2\ U1\ U0)_2$, from the set $\{0, 1, 2, \dots, 15\}$ 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

k	0	1	2	...	15
nk	n0	n1	n2	...	n15

where $n0, \dots, n15$ are given by the mapping table target vector $fktab$. If $BINF = off$, then the integer input k is active, while for $BINF = on$ the input is defined by the binary inputs $(U3\ U2\ U1\ U0)_2$.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

k	Integer input of the block	Long (I32)
U0..U3	Binary input (mask)	Bool

Parameter

BINF	Enable the binary selectors off ... Disabled (analog selector) on Enabled (binary selectors)	⊙on Bool
fktab	Mapping table ⊙[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]	Byte (U8)

Output

nk	Integer output of the block	Long (I32)
Y0..Y3	Binary output (mask)	Bool

NOT – Boolean complementation

Block Symbol

Licence: [STANDARD](#)



Function Description

The NOT block negates the input signal $Y = \neg U$.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

U	Logical input of the block	Bool
---	----------------------------	------

Output

Y	Logical output of the block	Bool
---	-----------------------------	------

OR – Logical sum of two signals

Block Symbol

Licence: [STANDARD](#)



Function Description

The **OR** block computes the logical sum of two input signals $Y = U1 \vee U2$. If you need to work with more input signals, use the [ORQUAD](#), [OROC](#) or [ORHEX](#) block.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

U1	First logical input of the block	Bool
U2	Second logical input of the block	Bool

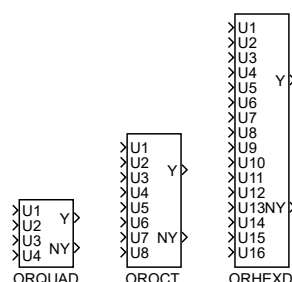
Output

Y	Output signal, logical sum	Bool
NY	Boolean complementation of Y	Bool

ORQUAD, OROCT, ORHEXD – Multi-input logical sum

Block Symbols

Licence: [STANDARD](#)



Function Description

The **ORQUAD**, **OROCT** and **ORHEXD** blocks compute the logical sum of up to sixteen input signals $U1, U2, \dots, U16$. The signals listed in the **n1** parameter are negated prior to computing the logical sum.

For an empty **n1** parameter a simple logical sum $Y = U1 \vee U2 \vee U3 \vee U4 \vee U5 \vee U6 \vee U7 \vee \dots \vee U16$ is computed. For e.g. **n1**=1,3...5, the logical function is $Y = \neg U1 \vee U2 \vee \neg U3 \vee \neg U4 \vee \neg U5 \vee U6 \vee \dots \vee U16$.

If you have only two input signals, consider using the [OR](#) block.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

$U1..U16$	Logical input of the block	Bool
-----------	----------------------------	------

Parameter

n1	List of signals to negate	Long (I32)
-----------	---------------------------	------------

Output

Y	Output signal, logical sum	Bool
NY	Boolean complementation of Y	Bool

RS – Reset-set flip-flop circuit

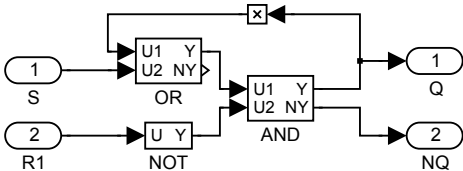
Block Symbol

Licence: [STANDARD](#)



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.



This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

S	Flip-flop set	Bool
R1	Priority flip-flop reset	Bool

Output

Q	Flip-flop circuit state	Bool
NQ	Boolean complementation of Q	Bool

SR – Set-reset flip-flop circuit

Block Symbol

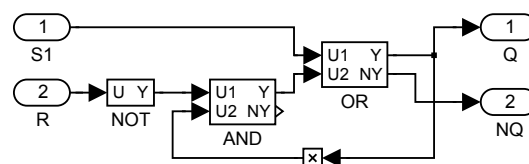
Licence: [STANDARD](#)



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.



This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

S1	Priority flip-flop set	Bool
R	Flip-flop reset	Bool

Output

Q	Flip-flop circuit state	Bool
NQ	Boolean complementation of Q	Bool

TIMER – Multipurpose timer

Block Symbol

Licence: [STANDARD](#)

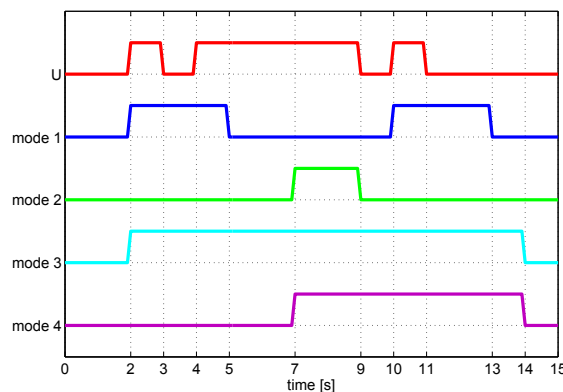


Function Description

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. Supported modes are:

- **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.

The graph illustrates the behaviour of the block in individual modes for **pt** = 3:



The timer can be paused by the **HLD** input. The **R1** input resets the timer. The reset signal overpowers the **U** input, similarly to the [RS](#) block.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

U	Trigger of the timer	Bool
HLD	Timer hold	Bool
R1	Timer reset	Bool

Parameter

mode	Timer mode	⊙1 Long (I32)
	1 Pulse generator	
	2 Delayed ON	
	3 Delayed OFF	
	4 Delayed change	
pt	Timer interval [s]	⊙1.0 Double (F64)

Output

Q	Timer output	Bool
et	Elapsed time [s]	Double (F64)
rt	Remaining time [s]	Double (F64)

Chapter 9

TIME – Blocks for handling time

Contents

DATE – Current date	310
DATETIME – Get, set and convert time	311
TC – Timer control and status	314
TIME – Current time	316
TS – Current timestamp	317
TS2NS – Timestamp difference in nanoseconds	319
WSCH – Week scheduler	320

The TIME library is specialized for time-based operations and scheduling in REXY-GEN system. It includes blocks like [DATE](#), [TIME](#) and [DATETIME](#) for handling date and datetime, providing essential tools for working with temporal data. The library features [TC](#) for internal timer control. Additionally, [WSCH](#) is used for scheduling, enabling efficient management of time-dependent tasks. This library is particularly valuable for systems requiring precise time management and scheduling capabilities.

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. The first day of the week is Sunday (numbered as 1).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Parameter

tz	Timezone	⊙1	Long (I32)
	1 Local time		
	2 UTC		

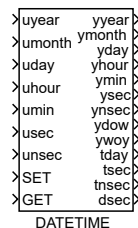
Output

year	Year	Long (I32)
month	Month	Long (I32)
day	Day	Long (I32)
dow	Day of week	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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uyear	Input for setting year	Long (I32)
umonth	Input for setting month	Long (I32)
uday	Input for setting day	Long (I32)

uhour	Input for setting hours		Long (I32)
umin	Input for setting minutes		Long (I32)
usec	Input for setting seconds		Long (I32)
unsec	Input for setting nanoseconds	↓-9.22E+18 ↑9.22E+18	Large (I64)
SET	Trigger for setting time		Bool
GET	Trigger for getting time		Bool

Parameter

isetmode	Source for setting time	⊙1	Long (I32)
	1 OS clock		
	2 Block inputs		
	3 The unsec input		
	4 The usec input		
	5 The unsec input - relative		
	6 The usec input - set HW clock only		
igetmode	Source for getting time	⊙6	Long (I32)
	1 OS clock		
	2 Block inputs		
	3 The unsec input		
	4 The usec input		
	5 The uday input		
	6 REXYGEN clock		
	7 HW clock		
settol	Tolerance for setting the REXYGEN clock [s]	⊙1.0	Double (F64)
setper	Period for setting time [s] (0=not periodic)		Double (F64)
getper	Period for getting time [s] (0=not periodic)	⊙0.001	Double (F64)
FDOW	First day of week is Sunday		Bool
	off ... Week starts on Monday		
	on Week starts on Sunday		
tz	Timezone	⊙1	Long (I32)
	1 Local time		
	2 UTC		

Output

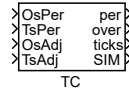
yyear	Year	Long (I32)
ymonth	Month	Long (I32)
yday	Day	Long (I32)
yhour	Hours	Long (I32)
ymin	Minutes	Long (I32)
ysec	Seconds	Long (I32)
ynsec	Nanoseconds	Long (I32)
ydw	Day of week	Long (I32)
ywoy	Week of year	Long (I32)
tday	Total number of days	Long (I32)

<code>tsec</code>	Total number of seconds	Long (I32)
<code>tnsec</code>	Total number of nanoseconds	Large (I64)
<code>dsec</code>	Number of seconds since midnight	Long (I32)

TC – Timer control and status

Block Symbol

Licence: [STANDARD](#)



Function Description

The TC function block controls the internal timer of REXYGEN. It is possible to modify the actual basic tick period (e.g. the value set in the `tick` parameter of the [EXEC](#) block) or logical tick period (e.g. the time added to the timestamp of each tick if `timer = CORETIMER` is selected). By default, the logical and physical period is the same and is the `EXEC:tick` parameter. The discretization period of the blocks in the control algorithm is not affected by the TC block.

The actual period can be changed in two ways: set the desired value to the `OsPer` input or set `OsAdj` for one tick. `OsAdj` will temporarily increase or decrease the actual period until the total shift set on the `OsAdj` input is realized. How much the period increases is controlled by the `OsMax` parameter.

Example: Let's expect the tick period to be 0.1s and `OsMax=0.2`, so let's set `OsAdj=1.0` to temporarily increase the real period to 0.12s (e.g. 20% defined in the `OsMax` parameter) until a total shift of 1s is realized, e.g. for 50 ticks.

Logical period control is the same using inputs/parameter `TsPer`, `TsAdj`, `TsMax`.

Note 1: The unconnected input or the input with a value of 0 is ignored.

Note 2: The actual period adjustment is not supported on Windows targets.

Note 3: The primary reason for this block is to synchronize with another controller in time-critical application, so the period should only be changed by a few percent. For simulation and debugging purposes, it is possible to change the period significantly to speed up a slow process (or slow down a fast process). This should be done with caution, as the synchronization with other controllers will not work and all calculations must be done in the shortened period. Also, in this case, warnings about missing ticks, incorrect period, etc. will appear in the log. For these purposes, it is better to use the simulation mode.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>OsPer</code>	Physical tick period [s]	Double (F64)
<code>TsPer</code>	Logical (timestamp) tick period [s]	Double (F64)
<code>OsAdj</code>	Physical tick shift [s]	Double (F64)
<code>TsAdj</code>	Logical (timestamp) tick shift [s]	Double (F64)

Parameter

<code>OsMax</code>	Maximal relative quantum for physical adjustment $\downarrow 0.0 \uparrow 1.0 \odot 0.1$	Double (F64)
<code>TsMax</code>	Maximal relative quantum for logical adjustment $\downarrow 0.0 \uparrow 1.0 \odot 0.1$	Double (F64)

Output

<code>per</code>	Last real physical tick period [s]	Double (F64)
<code>over</code>	Number of lose periods in last tick	Long (I32)
<code>ticks</code>	Number of ticks since start	Large (I64)
<code>SIM</code>	Timer in simulation mode	Double (F64)

TIME – Current time

Block Symbol

Licence: [STANDARD](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Parameter

tz	Timezone	⊕1	Long (I32)
	1 Local time		
	2 UTC		

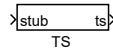
Output

hour	Hours	Long (I32)
min	Minutes	Long (I32)
sec	Seconds	Long (I32)

TS – Current timestamp

Block Symbol

Licence: [STANDARD](#)



Function Description

The **TS** block generates a time stamp on the output **ts** from a source specified by the **source** parameter. The **stub** input is used only to ensure the correct execution order of the block in the program.

The **source** parameter can be used to switch between several different time sources:

- 1: **CORETIMER** - is the primary time stamp used in the REXYGEN system, expressing the number of nanoseconds since January 1, 2000.
- 2: **CORETIMER (precise)** - operates similarly to **CORETIMER**, but the time stamp is updated at the moment the block is executed, using a different source, usually **RTC** or **PFC**.
- 3: **RTC (UTC)** - returns the time stamp in Coordinated Universal Time format.
- 4: **RTC (localtime)** - returns the time stamp in local time format (considering time zones).
- 5: **PFC** - is based on the **QueryPerformanceCounter** function on Windows systems and returns the system time stamp with a resolution of 1 ns.
- 6: **TSC** - is the fastest time stamp source, using the **RDTSC** instruction on x86 processors and the **CNTVCT_ELO** instruction on ARM64 processors. On other platforms, the **TSC** time stamp is identical to the **PFC** time stamp.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

stub	Dummy input for block ordering	Any
-------------	--------------------------------	------------

Parameter

source	Source of the time stamp	⊙1 Word (U16)
1	CORETIMER	
2	CORETIMER (precise)	
3	RTC (UTC)	
4	RTC (localtime)	
5	PFC	
6	TSC	

Output

ts	Time stamp	Large (I64)
----	------------	-------------

TS2NS – Timestamp difference in nanoseconds

Block Symbol

Licence: [STANDARD](#)



Function Description

The **TS2NS** block generates on the output **ns** the time difference between the time stamps on the **start** and **end** inputs (e.g., from the **TS** blocks). When calculating the difference, all **TS** and **TS2NS** blocks must have the same source in the **source** parameter, otherwise the result is nonsensical. The block provides the frequency of the time stamp on the **freq** output. The difference between the time stamps is in nanoseconds (ns). For more information on time stamp sources, see the **TS** block.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

start	Start time stamp	↓-9.22E+18 ↑9.22E+18	Large (I64)
end	End time stamp	↓-9.22E+18 ↑9.22E+18	Large (I64)

Parameter

source	Source of the time stamp	⊙1 Word (U16)
1 CORETIMER	
2 CORETIMER (precise)	
3 RTC (UTC)	
4 RTC (localtime)	
5 PFC	
6 TSC	

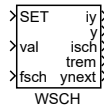
Output

ns	Interval in nanoseconds	Large (I64)
freq	Frequency of the time stamp	Large (I64)

WSCH – Week scheduler

Block Symbol

Licence: [STANDARD](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

SET	Trigger for setting output	Bool
val	Value to set the output to	Double (F64)
fsch	Forced schedule (0=normal operation)	Long (I32)
	0 Standard weekly schedule	
	1 Monday	
	2 Tuesday	
	3 Wednesday	
	4 Thursday	
	5 Friday	
	6 Saturday	
	7 Sunday	
	8 and above Additional daily program as defined by the wst table	

Parameter

tz	Timezone	⊙1	Long (I32)
	1 Local time		
	2 UTC		
nmax	Allocated size of arrays	↓10 ↑1000000 ⊙100	Long (I32)
imode	Reserved for internal use		Long (I32)
user	Reserved for special editor		String
wst	Weekly schedule table (day-hour-value)		Double (F64)
	⊙[1 0.01 18.0; 2 6.0 22.0; 2 18.0 18.0; 3 6.0 22.0; 3 18.0 18.0; 4 6.0 22.0; 4 18.0 18.0		
specdays	List of special days (date-daily program)		Long (I32)
	⊙[20150406 1; 20151224 1; 20151225 1; 20151226 1; 20160101 1; 20160328 1; 20170417 1; 20		

Output

iy	Integer output value	Long (I32)
y	Output value	Double (F64)
isch	Daily schedule identifier	Long (I32)
trem	Time remaining in the current section [s]	Double (F64)
ynext	Output in the next section	Double (F64)

Chapter 10

ARC – Data archiving

Contents

ACD – Archive compression using Delta criterion	325
ACLEAR – Forced archive purge	327
AFLUSH – Forced archive flushing	328
ALB, ALBI – Alarms for Boolean value	329
ALM, ALMI – Alarm store value	331
ALN, ALNI – Alarms for numerical value	332
ARS – Archive store value	335
TRND – Real-time trend recording	337
TRNDV – Real-time trend recording (for vector signals)	340

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 function blocks can be divided into groups by their use:

- Blocks for generating alarms and events – [ALB](#), [ALBI](#), [ALM](#), [ALMI](#), [ALN](#), [ALNI](#), [ARS](#).
- Blocks for recording trends – [ACD](#), [TRND](#), [TRNDV](#).
- Blocks for handling archives – [AFLUSH](#), [ACLEAR](#).

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 of 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 **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.

General archive properties

The archiving and trending blocks have several common properties which are listed below.

Archive list

The list of archives is specified in the blocks by the **arc** parameter in the form e.g. 1,3..5,8. Details about the archive numbering are in the [ARC](#) block. Third-party programs (Simulink, OPC clients, etc.) work with the whole number which is a bit mask – for the example above it is 157, in binary form 10011101.

Event identification code

The event identification code in the archive **id** must be unique in the entire target device with the **REXYGEN** control system (i.e. in all archive blocks). If **id** = 0, no alarm is generated. For **id** = -1, the alarm is identified by its name (i.e. the block name must be the same as the **Name** column in the alarm definition table).

Types of events and alarms

Events and alarms are differentiated in the **REXYGEN** system by the **lvl** parameter. If $1 \leq \text{lvl} \leq 127$, it is an alarm where the start, end and acknowledgment are stored in the archive. The range $128 \leq \text{lvl} \leq 255$ is reserved for events where only the time instant of the event is stored in the archive.

ACD – Archive compression using Delta criterion

Block Symbol

Licence: [STANDARD](#)



Function Description

The **ACD** (Archive Compression using Delta criterion) block is meant for storing compressed analog signals to archives using archive events.

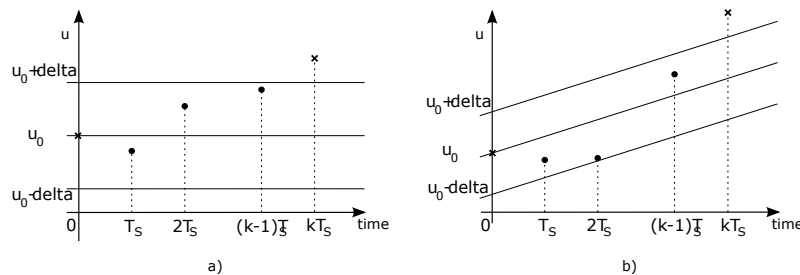
The main idea is to store the input signal **u** only when it changes significantly. The interval between two samples is in the range $\langle \mathbf{tmin}, \mathbf{tmax} \rangle$ seconds (rounded to the nearest multiple of the sampling period). A constant input signal is stored every **tmax** seconds while rapidly changing signal is stored every **tmin** seconds. When the execution of the block is started, the first input value is stored. This value will be referred to as **u0** in the latter. The rules for storing the following samples are given by the **delta** and **TR** input signals.

The list of archives for storing is specified by the **arc** parameter, e.g. 1,3..5,8. The event will be stored in all specified archives. Each archiving block must have a unique event identification code in the archive given by the **id** parameter. For more information about these parameters see the introduction of chapter 10.

For **TR = off** the condition $|u - u_0| > \mathbf{delta}$ is checked. If it holds and the last stored sample occurred more than **tmin** seconds ago, the value of input **u** is stored and **u0**=**u** is set. If the condition is fulfilled sooner than **tmin** seconds after the last stored value, the error output **E** is set to 1 and the first value following the **tmin** interval is stored. At that time the output **E** is set back to 0 and the whole procedure is repeated.

For **TR=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) **TR=off**, b) **TR=on**. The stored samples are marked by the symbol \times .



This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

u	Signal to compress and store		Double (F64)
delta	Threshold for storing the signal	↓0.0 ↑1e+10	Double (F64)

Parameter

acIs	Archive class (data type)	⊙8	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)		
	10 Large (I64)		
arc	List of archives to write the events to		Word (U16)
id	Unique archive item ID	⊙1	Word (U16)
tmin	The shortest interval between two samples [s]	↓0.001 ↑1000000.0 ⊙1.0	Double (F64)
tmax	The longest interval between two samples [s]	↓1.0 ↑1000000.0 ⊙1000.0	Double (F64)
TR	Trend evaluation	⊙on	Bool
	off ... Disabled		
	on Enabled		
Desc	Event description string	⊙Value Description	String

Output

y	The last value stored in the archive	Double (F64)
E	Error indicator	Bool
	off ... No error	
	on An error occurred	

ACLEAR – Forced archive purge

Block Symbol

Licence: [STANDARD](#)



Function Description

The **ACLEAR** block is meant for clearing the content of an archive when a rising edge **off**→**on** appears on the **CLEAR** input. The list of archives to be cleared is specified by the **arc** parameter, e.g. **1,3..5,8**. For more information see Chapter [10](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

CLEAR	Archive purge on rising edge	Bool
--------------	------------------------------	-------------

Parameter

arc	List of archives to write the events to	Word (U16)
------------	---	-------------------

AFLUSH – Forced archive flushing

Block Symbol

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. The list of archives is specified in the **arc**, e.g. **1,3..5,8**. For more information see Chapter [10](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

FLUSH	Force archive flushing	Bool
--------------	------------------------	-------------

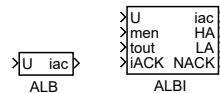
Parameter

arc	List of archives to write the events to	Word (U16)
------------	---	-------------------

ALB, ALBI – Alarms for Boolean value

Block Symbols

Licence: [STANDARD](#)



Function Description

The blocks ALB and ALBI generate alarms or events upon changes in the logical input signal **U**. The output **iac** indicates the current alarm (event) code. The parameter (or input) **men** selects whether to indicate a rising edge (**off**→**on**), which corresponds to an upper alarm (HA), a falling edge (**on**→**off**), which corresponds to a lower alarm (LA), or both edges of the input signal.

The **ALBI** block is an extension of the **ALB** block. The blocks differ only in that the inputs of the **ALBI** block: **men**, **tout**, **iACK** are parameters of the **ALB** block. The **ALB** does have **HA**, **LA** and **NACK** outputs.

Events and alarms are distinguished in the **REXYGEN** system using the **lvl** parameter. The list of archives for writing is specified by the **arc** parameter in the form e.g. 1,3..5,8. Each archiving block must have a unique event identification code in the archive given by the **id** parameter. For more information about these parameters see the introduction of Chapter 10. Positive values of the **iac** output codes can be added, e.g. the value 514 means that the upper alarm is unacknowledged. However, not all combinations make sense.

Note 1: The input (parameter) **iACK** is automatically reset to 0 after processing by the block. The alarm is assumed to be acknowledged by the operator from the visualization so that it is not necessary to write 0 with another query. It is a similar principle to the **BSTATE** parameter in the [MP](#) block.

Note 2: Formatting commands (values attached to the alarm, multilingual text) can be inserted into the **Desc** parameter. Their detailed description is in the [ALARMS](#) block.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

U	Logical input of the block	Bool
men	Enable alarms (mask)	Long (I32)
	0 All alarms disabled	
	1 Low-alarm enabled	
	2 High-alarm enabled	
	3 All alarms enabled	
tout	Alarm activation delay time [s]	↓0.0 Double (F64)

iACK	Alarm acknowledge (mask)	Byte (U8)
1	Low-alarm (LA) acknowledge	
2	High-alarm (HA) acknowledge	
3	Both alarms acknowledge	

Parameter

arc	List of archives to write the events to	Word (U16)
id	Unique archive item ID	⊙1 Word (U16)
lvl	Alarm level	↓1 ⊙1 Byte (U8)
group	Group to which the alarm belongs	↓0 ↑9.22337E+18 Large (I64)
Desc	Alarm description string	⊙Alarm Description String

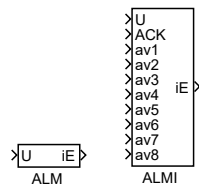
Output

iac	Current alarm code	Long (I32)
0	Signal within limits	
1	Low-alarm active (LA)	
2	High-alarm active (HA)	
256	Low-alarm (LA) not acknowledged (NACK)	
512	High-alarm (HA) not acknowledged (NACK)	
HA	High-alarm indicator	Bool
LA	Low-alarm indicator	Bool
NACK	Not-acknowledged-alarm indicator	Bool

ALM, ALMI – Alarm store value

Block Symbols

Licence: [STANDARD](#)



Function Description

The ALM and ALMI blocks are used for generating alarms. An alarm is activated when the input U changes to **on**. Each alarm must be defined using the [ALARMS](#) block and is uniquely identified by the **id** parameter (see [10](#)). Active alarms (i.e., with U=**on**) can be displayed in the **AlarmsTable** visualization component. Records of the alarm state change and its acknowledgment are stored in the archive if allowed in the [ALARMS](#) block configuration. An alarm can be acknowledged by activating the **ACK=on** parameter.

Note: The system allows displaying the acknowledgment status, acknowledging an active alarm, and can display this status in both the HMI and the archive. However, the REXYGEN system does not operate further on the acknowledgment and no functionality depends on it. Decisions about acknowledging alarms depend on the filter settings in the visualization and on the specific system design.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

U	Logical input of the block	Bool
ACK	Alarm acknowledge (mask)	Bool
av1..av8	Alarm associated value	$\downarrow 1.79769\text{e}+308 \odot -1.79769\text{e}+308$ Double (F64)

Parameter

id	Unique archive item ID	$\downarrow -1 \uparrow 65535 \odot -1$ Long (I32)
----	------------------------	--

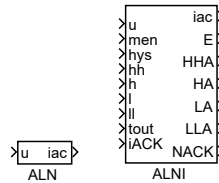
Output

iE	Error code	Error
----	------------	-------

ALN, ALNI – Alarms for numerical value

Block Symbols

Licence: [STANDARD](#)



Function Description

Blocks ALN and ALNI are intended for generating two-level alarms or events when the numerical value of input **u** exceeds (undershoots) one of the alarm limits. **iac** output indicates alarm (event) status. The **men** parameter (or input) specifies which of the boundaries are monitored. You can choose between the following limits:

- **LA** - low-alarm
- **HA** - high-alarm
- **LLA** - second low-alarm
- **HHA** - second high-alarm

and their combinations.

The ALNI block is an extension of the ALN block. The blocks differ only in that most of the inputs of the ALNI block are parameters of the ALN block. The ALN block does not have the HHA, HA, LA, LLA and NACK outputs.

Individual limit values can be set by the parameters (inputs) **l**, **h**, **ll** and **hh**. The **hys** value determines the alarm hysteresis. The outputs **HHA**, **HA**, **LA**, **LLA** and **NACK** indicate whether the alarm is active/unacknowledged.

Events and alarms are distinguished in the REXYGEN system using the **lvl** parameter. The list of archives for writing is specified by the **arc** parameter, e.g. **1,3..5,8**. The event identification code in the archive **id** must be unique in the entire target device with the REXYGEN control system. For more information about these parameters see the introduction of Chapter 10. Positive values of the **iac** output codes can be added, e.g. the value **514** means that the upper alarm is unacknowledged. However, not all combinations make sense.

Note 1: The input (parameter) **iACK** is set back to 0 immediately by the block algorithm. The functionality is similar to the parameter **BSTATE** of the block [MP](#).

Note2: The parameter **Desc** can include formatting characters (multilingual texts, associated variables). Formatting rules are described in the [ALARMS](#) block.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

u	Analog input of the block		Double (F64)
men	Enable alarms (mask)	⊙15	Long (I32)
	0 All alarms disabled		
	1 Low-alarm (LA) enabled		
	2 High-alarm (HA) enabled		
	3 LA and HA enabled		
	4 Second low-alarm (LLA) enabled		
	5 LA and LLA enabled		
	6 HA and LLA enabled		
	7 LA, HA and LLA enabled		
	8 Second high-alarm (HHA) enabled		
	9 LA and HHA enabled		
	10 HA and HHA enabled		
	11 LA, HA and HHA enabled		
	12 LLA and HHA enabled		
	13 LA, LLA and HHA enabled		
	14 HA, LLA and HHA enabled		
	15 All alarms enabled		
hys	Alarm hysteresis	↓1e-10 ↑1e+10	Double (F64)
hh	The second high-alarm limit		Double (F64)
h	High-alarm limit		Double (F64)
l	Low-alarm limit		Double (F64)
ll	The second low-alarm limit		Double (F64)
tout	Alarm activation delay time [s]	↓0.0	Double (F64)
iACK	Alarm acknowledge (mask)		Byte (U8)
	1 Low-alarm (LA) acknowledge		
	2 High-alarm (HA) acknowledge		
	4 Second low-alarm acknowledge (LLA)		
	8 Second high-alarm acknowledge (HHA)		

Parameter

acls	Alarm class (data type)	⊙8	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)		
	10 Large (I64)		
arc	List of archives to write the events to		Word (U16)

id	Unique archive item ID	⊙1	Word (U16)
lv11	Level of low and high alarms	↓1 ⊙1	Byte (U8)
lv12	Level of the second low and high alarms	↓1 ⊙10	Byte (U8)
group	Group to which the alarm belongs	↓0 ↑9.22337E+18	Large (I64)
Desc	Alarm description string	⊙Alarm Description	String

Output

iac	Current alarm code	Long (I32)
	0 Signal within limits	
	1 Low-alarm active (LA)	
	2 High-alarm active (HA)	
	4 Second low-alarm active (LLA)	
	8 Second high-alarm (HHA) active	
	256 . . . Low-alarm (LA) not acknowledged (NACK)	
	512 . . . High-alarm (HA) not acknowledged (NACK)	
	1024 . . Second low-alarm (LLA) not acknowledged (NACK)	
	2048 . . Second high-alarm (HHA) not acknowledged (NACK)	
	-1 Invalid alarm limits	
E	Error indicator	Bool
	off . . . No error	
	on An error occurred	
HHA	The second high-alarm indicator	Bool
HA	High-alarm indicator	Bool
LA	Low-alarm indicator	Bool
LLA	The second low-alarm indicator	Bool
NACK	Not-acknowledged-alarm indicator	Bool

ARS — Archive store value

Block Symbol

Licence: [STANDARD](#)



Function Description

If **RUN** = **on**, the **ARS** block writes the value at the **u** input to the archive. The type of the value at the input is determined by the **type** parameter, and the type of the archive item is the same. The **subtype** parameter allows you to specify the type of alarm that alarm blocks write (e.g. **L->H** for a logical alarm, or **HiHi** for a numeric alarm). The value of the parameter can be 0 to 7 and is not used for arrays. This parameter is usually not used. The meaning of the other parameters is the same as for other blocks for writing to the archive.

If **type** = **Reference**, an array (column vector or matrix) is expected. If it is a matrix, each of its columns is saved as a separate file in the archive (i.e., in one tick task with this block, as many entries as the matrix of columns will stand out in the archive).

Note 1: In the case of arrays, the archive subsystem is limited to 255 values in one item. At the same time, there is a limit of 512 bytes of data in one item, so for the **Short** type, at most 128 values are saved, for the **Long** type at most 64 values, and for the **Double** type at most 32 values. If the input array is longer, the block saves the specified number of values from the beginning of the array and does not report any errors.

Note 2: In the case of a **string**, the archive subsystem is limited to 65535 bytes (characters in UTF8 encoding may be less). If the input text is longer, the block saves the first 65535 bytes from the beginning of the array and does not report any errors. Some reading functions may have a small buffer, and such a long text cannot be read, so it is recommended not to exceed 4080 bytes (characters if only characters from the English keyboard are used).

Note 3: The **id** parameter usually serves to link the item in the archive to the source block/signal (and alarm in some cases). Therefore, its uniqueness is checked across the entire configuration. The **ARS** block is considered a low-level block that writes an event to the archive without further context and checks. Therefore, the uniqueness of the **id** parameter is not checked here. For example, if numeric or text items start appearing in the archive for a binary alarm, they are almost certainly generated by some **ARS** block (or an analogous function in the script of the [REXLANG](#) block).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

u	Signal to store into archive	Any
---	------------------------------	-----

RUN	Enable execution	Bool
-----	------------------	------

Parameter

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)
id	Unique archive item ID	⊙1	Word (U16)
lvl	Alarm level	⊙1	Word (U16)
Desc	Event description string	⊙Value Description	String

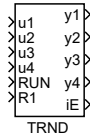
Output

iE	Error code	Error
----	------------	-------

TRND – Real-time trend recording

Block Symbol

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. For better clarity, the individual trends can be assigned names by the **Title** parameter and the individual signals by the **SigNames** parameter.

The actual number of stored signals is determined by the parameter **n**. In case the trend buffers of length 1 samples get full, the oldest samples are overwritten. Data can be stored once in **pfac** executions of the block (decimation), i.e. with the period $pfac \cdot T_s$, where T_s is the period of the block execution [s]. The stored data can be further processed according to the values of the parameters **ptype1** to **ptype4**. The other decimation factor **afac** can be used for storing data in archives. The period of storing is then given by $afac \cdot pfac \cdot T_s$. Each value is stored with a time stamp. The source of the time stamp can be set by the **timesrc** parameter (see the **TS** block for more information).

The list of archives for storing is specified by the **arc** parameter, e.g. 1,3..5,8. The event will be stored in all specified archives. Each archiving block must have a unique event identification code in the archive given by the **id** parameter. For more information about these parameters see the introduction of Chapter 10.

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 $s \cdot n \cdot l$, 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 $l = 1000$, $8 \cdot 4 \cdot 1000 = 32000$ 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 24 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 **iE** output according to the following table (the unused bits are omitted):

Error	Range underflow				Range overflow			
Input	u4	u3	u2	u1	u4	u3	u2	u1
Bit number	11	10	9	8	3	2	1	0
Bit weight	2048	1024	512	256	8	4	2	1

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.

It is possible to read, display and export the stored data by the REXYGEN Studio in the **Watch** mode. After double-clicking on the corresponding TRND block, a new card with the prefix Trend will open.

WARNING: set any of the parameters **arc**, **afac**, **id** to 0/empty disable writing data into archive. The data are available in diagnostic tools only in this case.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

u1..u4	Analog input of the block	Double (F64)
RUN	Enable execution	Bool
R1	Block reset	Bool

Parameter

n	Number of signals (trend buffers)	↓1 ↑4 ⊙4	Long (I32)
l	Number of samples per trend buffer	↓0 ↑268435000 ⊙1000	Long (I32)
btype	Type of all trend buffers	⊙8	Long (I32)
	1 Bool		
	2 Byte (U8)		
	3 Short (I16)		
	4 Long (I32)		
	5 Word (U16)		
	6 DWord (U32)		
	7 Float (F32)		
	8 Double (F64)		
	10 Large (I64)		
ptype1..ptype4	Data processing	⊙1	Long (I32)
	1 Store		
	2 Minimum		
	3 Maximum		
	4 Sum		
	5 Average		
	6 RMS		
	7 Variance		
pfac	Processing factor	↓1 ↑1000000 ⊙1	Long (I32)
afac	Archiving factor	↓0 ↑1000000	Long (I32)

arc	List of archives to write the events to		Word (U16)
id	Unique archive item ID	⊙1	Word (U16)
Title	Trend title string	⊙Trend Title	String
timesrc	Source of timestamps	⊙1	Long (I32)
	1 CORETIMER		
	2 CORETIMER (precise)		
	3 RTC (UTC)		
	4 RTC (localtime)		
	5 PFC		
	6 TSC		
SigNames	Name of signals (each line one signal respectively)		String
ViewConfig	User string for description and formatting		String

Output

y1..y4	Analog output of the block	Double (F64)
iE	Error code (bitwise multiplexed)	Long (I32)

TRNDV – Real-time trend recording (for vector signals)

Block Symbol

Licence: [STANDARD](#)



Function Description

The **TRNDV** block is very similar to the [TRND](#) block. However, it allows storing more than 4 signals. The signals are passed to the **uVec** input in the form of a vector. The number of processed signals is then determined by the **n** parameter. In contrast to the [TRND](#) block, it is necessary to set the **HLD** input to **onto** stop the execution of the block.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uVec	Vector signal to record	Reference
HLD	Hold	Bool
R1	Block reset	Bool

Parameter

n	Number of signals (trend buffers)	↓1 ↑64 ⊙8	Long (I32)
l	Number of samples per trend buffer	↓2 ↑268435000 ⊙1000	Long (I32)
btype	Type of all trend buffers	⊙8	Long (I32)
	1 Bool		
	2 Byte (U8)		
	3 Short (I16)		
	4 Long (I32)		
	5 Word (U16)		
	6 DWord (U32)		
	7 Float (F32)		
	8 Double (F64)		
	10 Large (I64)		
pfac	Processing factor	↓1 ↑1000000 ⊙1	Long (I32)
afac	Archiving factor	↓0 ↑1000000	Long (I32)
arc	List of archives to write the events to		Word (U16)
id	Unique archive item ID	⊙1	Word (U16)
Title	Trend title string	⊙Trend Title	String

<code>timesrc</code>	Source of timestamps	⊙1	Long (I32)
	1 CORETIMER		
	2 CORETIMER (precise)		
	3 RTC (UTC)		
	4 RTC (localtime)		
	5 PFC		
	6 TSC		
<code>SigNames</code>	Name of signals (each line one signal respectively)		String
<code>ViewConfig</code>	User string for description and formatting		String

Output

<code>iE</code>	Error code	Error
	i REXYGEN error code	

Chapter 11

STRING – Blocks for string operations

Contents

CNS – String constant	344
CONCAT – Concat string by pattern	345
FIND – Find substring	346
ITOS – Integer number to string conversion	347
LEN – String length	348
MID – Substring extraction	349
PJROCT – Parse JSON string (real output)	350
PJSEXOCT – Parse JSON string (string output)	352
PJSOCT – Parse JSON string (string output)	354
REGEXP – Regular expression parser	356
REPLACE – Replace substring	358
RTOS – Real number to string conversion	359
SELSOCT – Selector switch for string signals	360
STOR – String to real number conversion	362
TRIM – Remove leading and trailing whitechar	363

The STRING library is dedicated to string manipulation and analysis in REXY-GEN system. It includes blocks like **CONCAT** for concatenating strings, **FIND** for searching within strings, and **REPLACE** for replacing string segments. The library offers **LEN** and **MID** for determining string length and extracting substrings, respectively. Advanced pattern matching is provided by **REGEXP**. Conversion blocks such as **ITOS**, **STOR** and **RTOS** convert integers and real numbers to strings, while a simple **CNS** block defines a string constant. Additionally, the library features blocks like **PJROCT** for JSON parsing. This collection of blocks is essential for handling and processing string data in various applications.

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**. If the string parameter **filename** is not empty, the initialization data are loaded from the file **filename** on the host computer.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Parameter

scv	String (constant) value		String
nmax	Allocated size of string	↓0 ↑65520	Long (I32)
filename	Data file (content loaded into scv if set)		String

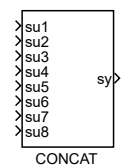
Output

sy	String output value	String
-----------	---------------------	--------

CONCAT – Concat string by pattern

Block Symbol

Licence: [STANDARD](#)



Function Description

The **CONCAT** block concatenates up to 8 input strings **su1** to **su8** by pattern specified in **ptrn** parameter.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

su1..su8	String input value	String
-----------------	--------------------	--------

Parameter

ptrn	Concatenation pattern	⊕%1%2%3%4	String
nmax	Allocated size of string	↓0 ↑65520	Long (I32)

Output

sy	String output value	String
-----------	---------------------	--------

FIND – Find substring

Block Symbol

Licence: [STANDARD](#)



Function Description

The **FIND** block searches for the string **su2** in the string **su1**. If **su2** is found, the index of the first occurrence of **su2** in **su1** (counted from one) is returned in the output **pos**. If **su2** is not found, zero is returned in the output **pos** and an error is indicated in the output **iE**. Both input strings are truncated to the length **nmax**.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

su1	String input value	String
su2	String input value	String

Parameter

nmax	Allocated size of string	↓0 ↑65520	Long (I32)
------	--------------------------	-----------	------------

Output

pos	Position of substring	Long (I32)
iE	Error code	Error

ITOS – Integer number to string conversion

Block Symbol

Licence: [STANDARD](#)



Function Description

The **ITOS** block is used for converting an integer into text. The **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 **mode** parameter. The **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).

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

n	Integer input of the block	Long (I32)
----------	----------------------------	------------

Parameter

len	Minimum length of output string	↓0 ↑30	Long (I32)
mode	Output string format	⊙1	Long (I32)
	1 Align right, fill with spaces		
	2 Align right, fill with zeroes		
	3 Align left, fill with spaces		
radix	Radix	⊙10	Long (I32)
	2 Binary		
	8 Octal		
	10 Decimal		
	16 Hexadecimal		

Output

sy	String output value	String
-----------	---------------------	--------

LEN – String length

Block Symbol

Licence: [STANDARD](#)



Function Description

The LEN block returns the actual length of the string in **su** in UTF-8 characters.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

su	String input value	String
-----------	--------------------	--------

Parameter

nmax	Allocated size of string	↓0 ↑65520	Long (I32)
-------------	--------------------------	-----------	------------

Output

len	Length of input string	Long (I32)
------------	------------------------	------------

MID – Substring extraction

Block Symbol

Licence: [STANDARD](#)



Function Description

The MID block extracts a substring **sy** from the string **su**. The inputs **l** and **p** specify the position and length of the string to be extracted in UTF-8 characters. The value of the input **p** is counted from one.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

su	String input value	String
l	Length of output string	Long (I32)
p	Position of output string	Long (I32)

Parameter

nmax	Allocated size of string	↓0 ↑65520	Long (I32)
-------------	--------------------------	-----------	------------

Output

sy	String output value	String
iE	Error code	Error

PJROCT – Parse JSON string (real output)

Block Symbol

Licence: [STANDARD](#)



Function Description

The **PJROCT** block parses input JSON string **jtxt** according to specified **name*** parameters when the input **RUN** is **on**. Output signals are **real** type. In case of an error, the **y*** outputs are set to the value of the **yerr** parameter (e.g. the specified object does not exist or the value is not a number).

This block expects text in JSON format on the **jtxt** input. The outputs of **y1** to **y7** then have the values (string) of the objects identified by the parameters **name1** to **name7**. If one of the parameters **name1** to **name7** is empty, the corresponding output will be empty and this is not considered as an error. The input string evaluates only if **RUN** = **on**. An error is indicated on the output **iE**. The following cases may occur:

- **0** - no error
- **-1** - one of the parameters **name1** to **name7** refers to an object that does not appear in the input text (at the input **jtxt**)
- **-103** - the text on the input **jtxt** does not correspond to the JSON format
- **-106** - all of the parameters **name1** to **name7** refer to an object that does not appear in the input text (on the input **jtxt**)

Examples

Let

```
jtxt = "{\"id\": 12345, \"params\": {\"temperature\": 23, \"pressure\": 2.34 },
\"description\": \"reactor1\", \"values\" :[12, 34.5 , 45.0, 30.2]}\"
```

```
name1 = \"params.temperature\",
```

```
name2 = \"values[0]\",
```

```
name3 = \"pressure\",
```

```
name4 = \"description\",
```

then the output **y1** will be the **"23"** string, the output **y2** will be the **"12"** string, output **y3** will remain empty and an error will be signaled, the output **y4** will remain empty and an error will be signaled.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>jtxt</code>	JSON formatted string	String
<code>RUN</code>	Enable execution	Bool

Parameter

<code>name1..name8</code>	Name of JSON object	String
<code>nmax</code>	Allocated size of string	↓0 ↑65520 Long (I32)
<code>yerr</code>	Substitute value for an error case	Double (F64)

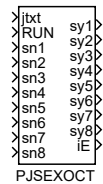
Output

<code>y1..y8</code>	Block output signal	Double (F64)
<code>iE</code>	Error code	Error

PJSEXOCT – Parse JSON string (string output)

Block Symbol

Licence: STANDARD



Function Description

The PJSEXOCT block is almost identical to the [PJSOCT](#) block. It expects text in JSON format on the `jtxt` input. The outputs of `sy1` to `sy7` then have the values of the objects identified by the parameters `name1` to `name7`. Unlike the [PJSOCT](#) block, the parameters `name1` to `name7` can contain the placeholder `% + number` instead of which the text from the input `sn + number` is substituted.

Examples

Let

```
sn1 = "2",
sn2 = "rpm",
name1 = "motor[%1].temp",
name2 = "motor[%1].%2",
```

then the output `sy1` will be the value of the object `motor[2].temp`, and the output `sy2` will be the value of the object `motor[2].rpm`.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>jtxt</code>	JSON formatted string	String
<code>RUN</code>	Enable execution	Bool
<code>sn1..sn8</code>	Name of JSON object	String

Parameter

<code>name1..name8</code>	Name of JSON object	String
<code>nmax</code>	Allocated size of string	$\downarrow 0 \uparrow 65520$ Long (I32)

Output

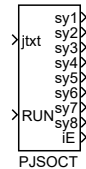
<code>sy1..sy8</code>	String output value
<code>iE</code>	Error code

String
Error

PJSOCT – Parse JSON string (string output)

Block Symbol

Licence: [STANDARD](#)



Function Description

The **PJSOCT** block parses input JSON string **jtxt** according to specified **name*** parameters when the input **RUN** is **on**. Output signals are **string** type.

This block expects text in JSON format on the **jtxt** input. The outputs of **sy1** to **sy7** then have the values of the objects identified by the parameters **name1** to **name7**. If one of the parameters **name1** to **name7** is empty, the corresponding output will be empty and this is not considered as an error. The input string evaluates only if **RUN** = **on**. An error is indicated on the output **iE**. The following cases may occur:

- **0** - no error
- **-1** - one of the parameters **name1** to **name7** refers to an object that does not appear in the input text (at the input **jtxt**)
- **-103** - the text on the input **jtxt** does not correspond to the JSON format
- **-106** - all of the parameters **name1** to **name7** refer to an object that does not appear in the input text (on the input **jtxt**)

Examples

Let

```
jtxt = "{\"id\": 12345, \"params\": {\"temperature\": 23, \"pressure\": 2.34 },
\"description\": \"reactor1\", \"values\" :[12, 34.5 , 45.0, 30.2]}\"
```

```
name1 = \"params.temperature\",
```

```
name2 = \"values[0]\",
```

```
name3 = \"pressure\",
```

```
name4 = \"description\",
```

then the output **sy1** will be the **"23"** string, the output **sy2** will be the **"12"** string, output **sy3** will remain empty and an error will be signaled, the output **sy4** will be the **"reactor1"** string.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

jtxt	JSON formatted string	String
RUN	Enable execution	Bool

Parameter

name1..name8	Name of JSON object	String
nmax	Allocated size of string	↓0 ↑65520 Long (I32)

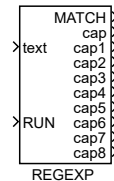
Output

sy1..sy8	String output value	String
iE	Error code	Error

REGEXP – Regular expression parser

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **REGEXP** block implements the most common subset of regular expressions as known, for example, from the **regex** command in Perl or **C#**, or the **grep** command known from the command line of Unix operating systems.

Supported syntax is as follows:

- **(?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]+` ... Finds first integer in input string (and puts it into `cap` output).
- `[-+]?[0-9]*\.[0-9]+([eE] [-+]?[0-9]+)?` ... Find first real number in input string (and puts it into `cap` output).
- `^\s*(.*?)\s*$` ... Puts trimmed input string into `cap1` output.
- `num\s*:\s*([0-9]*\.[0-9]*)` ... Expects input string in JSON format; find integer parameter `num`, and puts its value into `cap1`.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>text</code>	String to parse	String
<code>RUN</code>	Enable execution	Bool

Parameter

<code>expr</code>	Regular expression pattern	String
<code>nmax</code>	Allocated size of string	↓0 ↑65534 Long (I32)
<code>bufmax</code>	Parser internal buffer size (0 = autodetect)	↓0 ↑10000000 Long (I32)

Output

<code>MATCH</code>	Pattern match flag	Bool
<code>cap</code>	Whole matching string	String
<code>cap1..cap8</code>	Captured string	String

REPLACE – Replace substring

Block Symbol

Licence: [STANDARD](#)



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 numbered from one.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

<code>su1</code>	String input value	String
<code>su2</code>	String input value	String
<code>l</code>	Length of origin text	Long (I32)
<code>p</code>	Position of origin text	Long (I32)

Parameter

<code>nmax</code>	Allocated size of string	$\downarrow 0 \uparrow 65520$	Long (I32)
-------------------	--------------------------	-------------------------------	------------

Output

<code>sy</code>	String output value	String
<code>iE</code>	Error code	Error

RTOS – Real number to string conversion

Block Symbol

Licence: [STANDARD](#)



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. Possible values of the **mode** parameter are:

- **1: Best fit** – fixed point, but for extremely small or big numbers exponential format; parameter **prec** is total maximum number of characters in output (mantisa for exponential format)
- **2: Normal** – fixed point format; parameter **prec** is number of places after the decimal point
- **3: Exponential** – scientific format; parameter **prec** is number of places after the decimal point

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

prec	Precision (number of digits)	↓0 ↑20	Long (I32)
mode	Output string format	⊙1	Long (I32)
	1 Best fit		
	2 Normal		
	3 Exponential		

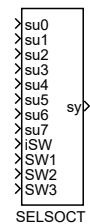
Output

sy	String output value	String
-----------	---------------------	--------

SELSOCT – Selector switch for string signals

Block Symbol

Licence: STANDARD



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:

iSW	SW1	SW2	SW3	y
0	off	off	off	u0
1	on	off	off	u1
2	off	on	off	u2
3	on	on	off	u3
4	off	off	on	u4
5	on	off	on	u5
6	off	on	on	u6
7	on	on	on	u7

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

su0...su7	String input value	String
iSW	Active signal selector	Long (I32)
SW1	Binary signal selector	Bool
SW2	Binary signal selector	Bool
SW3	Binary signal selector	Bool

Parameter

BINF	Enable the binary selectors	Bool
nmax	Allocated size of string	↓0 ↑65520 Long (I32)

Output

sy

The selected input signal

String

STOR – String to real number conversion

Block Symbol

Licence: [STANDARD](#)



Function Description

The **STOR** block converts the string on the input **su** to a real number on the output **y**. If the conversion fails, an error is indicated in **E**.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

su	String input value	String
----	--------------------	--------

Parameter

yerr	Substitute value for an error case	Double (F64)
------	------------------------------------	--------------

Output

y	Analog output of the block	Double (F64)
E	Error indicator	Bool

TRIM – Remove leading and trailing whitechar

Block Symbol

Licence: [STANDARD](#)



Function Description

The **TRIM** block removes leading and trailing white spaces from the input string **su** and puts the result in the output string **sy**.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

su	String input value	String
-----------	--------------------	--------

Parameter

nmax	Allocated size of string	↓0 ↑65520	Long (I32)
-------------	--------------------------	-----------	------------

Output

sy	String output value	String
-----------	---------------------	--------

Chapter 12

PARAM – Blocks for parameter handling

Contents

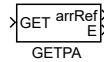
GETPA – Block for remote array parameter acquirement	366
GETPB, GETPI, GETPR – Blocks for remote parameter acquirement .	368
GETPS – Block for remote string parameter acquirement	370
GETPX – Block for remote parameter acquirement	371
PARA – Block with input-defined array parameter	373
PARE – Block with input-defined enumeration parameter	375
PARB, PARI, PARR – Blocks with input-defined parameter	376
PARS – Block with input-defined string parameter	378
PARX – Block with input-defined parameter	379
SETPA – Block for remote array parameter setting	381
SETPB, SETPI, SETPR – Blocks for remote parameter setting	383
SETPS – Block for remote string parameter setting	385
SETPX – Block for remote parameter setting	386
SGSLP – Set, get, save and load parameters	388
SILO – Save input value, load output value	392
SILOS – Save input string, load output string	394

The PARAM library is designed for parameter management and signal processing in the REXYGEN system. It includes blocks like [PARR](#) and its variants for defining and modifying various types of parameters. Blocks for getting parameters of other blocks like [GETPA](#) and [GETPS](#). Conversely, [SETPA](#), [SETPR](#) and [SETPS](#) are used to dynamically set parameter values of other blocks. Additionally, the library contains [SILO](#) and [SILOS](#) for exporting and importing values from a file. This library is crucial for systems requiring dynamic parameter manipulation and the ability to read/save values to a file.

GETPA – Block for remote array parameter acquirement

Block Symbol

Licence: [STANDARD](#)



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 with a **'.'** character, indicating the level where the **GETPA** block is placed. Examples of paths: **".CNDR:yp"**, **".Lights.ATMT:touts"**.
- Starts with **'..'** characters, indicating a level above the **GETPA** block. Examples of paths: **"..CNDR:yp"**, **"..Lights.ATMT:touts"**.

- **Relative to Task:** Starts at the root level of the task where the **GETPA** block is located. The string has to be prefixed with **'%'** in this case. Examples of paths: **"%CNDR:yp"**, **"%Lights.ATMT:touts"**.

- **Absolute:** A 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 presented in a tree-like structure within the **Diagnostics** section of the **REXYGEN Studio** program.

Warning: If the remote parameter is in a task other than the **GETPA** block, block execution is delayed until the remote task is completed. It is necessary to avoid the so-called race conditions and guarantee the correct value reading. Therefore, it is recommended to include the **GETPA** block in a slower task (longer period/execution time) and read parameter in a faster task (shorter period/execution time). In the opposite situation (e.g. the **GETPA** block in a faster task), the [SETPA](#) block should be used in a slower task.

Note 1: If parameter **GETF** = **off** and source array is in same task as the **GETPA** block, data are not copy into intermediate array, but output is direct reference to original array. It save resources (cpu time and memory). The **nmax**, **etype** parameters are ignored in this case.

Note 2: If multiple **GETPA** blocks are used to read arrays in another task, it is not guaranteed that all arrays will be read in one period of the second task. It is only guaranteed that the **GETPA** block executed earlier will read the array from the same or earlier period of the second task than the **GETPA** block executed later. The execution order can be seen in the **REXYGEN Studio** program diagnostics.

Note 3: The remote parameter must be a primary array (for example **CNA:acn**, **RTOV:xVec**, **MX_MAT:ay**). The array reference (like **CNA:vec**, **RTOV:yVec**, **SUBSYSTEM:Outputport**) is not supported.

Note 4: The **INCONN** block can also be used to read the value remotely.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

GET	Input for initiating one-shot parameter read	Bool
------------	--	-------------

Parameter

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	
nmax	Maximum size of array	↓10 ⊙256 Long (I32)
etype	Type of elements	⊙8 Long (I32)
	1 Bool	
	2 Byte (U8)	
	3 Short (I16)	
	4 Long (I32)	
	5 Word (U16)	
	6 DWord (U32)	
	7 Float (F32)	
	8 Double (F64)	
	10 Large (I64)	

Output

arrRef	Array reference	Reference
E	Error indicator	Bool
	off ... No error	
	on An error occurred	

GETPB, GETPI, GETPR – Blocks for remote parameter acquisition

Block Symbols

Licence: [STANDARD](#)



Function Description

The **GETPR**, **GETPI**, and **GETPB** blocks are used for acquiring the parameters of other blocks in the model remotely. Blocks have identical functionality, differing only in the type of parameter they acquire. The **GETPR** block is for a real number, **GETPI** for an integer, and **GETPB** for a Boolean value. To comply with the naming convention for variables [1.3](#), the outputs of individual blocks are named according to the type of the acquired parameter:

- **y** – real output of the **GETPR** block,
- **k** – integer output of the **GETPI** block,
- **Y** – Boolean output for the **GETPB** block.

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 (**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>**. 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 with a **'.'** character, indicating the level where the **GETPR** block (or **GETPI**, **GETPB**) is placed. Examples: **".GAIN:k"**, **".Motor1.Position:ycn"**.
 - Starts with **'..'** characters, indicating a level above the **GETPR** block (or **GETPI**, **GETPB**). Examples: **"..GAIN:k"**, **"..Motor1.Position:ycn"**.

- **Relative to Task:** Starts at the root level of the task where the `GETPR` block (or `GETPI`, `GETPB`) is located. The string has to be prefixed with `'%'` in this case. Examples: `"%GAIN:k"`, `"%Motor1.Position:ycn"`.
- **Absolute:** A 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: `"task1.inputs.lin1:u2"`, `"&EfaDrv.measurements.DER1:n"`.

The order and names of individual hierarchic levels are presented in a tree-like structure within the **Diagnostics** section of the **REXYGEN Studio** program.

Warning: If the remote parameter is in a task other than the `GETPx` block, block execution is delayed until the remote task is completed. It is necessary to avoid the so-called race conditions and guarantee the correct value reading. Therefore, it is recommended to include the `GETPx` block in a slower task (longer period/execution time) and read parameter in a faster task (shorter period/execution time). In the opposite situation (e.g. the `GETPx` block in a faster task), the `SETPx` block should be used in a slower task.

Note 1: When using multiple `GETPx` blocks, it is not guaranteed to read all data from a remote task in the same tick. It is only guaranteed that the previous block will receive a value in the same or previous period as the next block. The execution order can be seen in the **REXYGEN Studio** program diagnostics. To obtain multiple values in the same period, it is needed to use the `Inport` and `Outport` blocks or the `GETPA` block.

Note 2: The `GETPX` and `INCONN` blocks can also be used to read the value remotely.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

GET	Input for initiating one-shot parameter read	Bool
-----	--	------

Parameter

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	

Output

y	Parameter value	Double (F64)
E	Error indicator	Bool
	off ... No error	
	on An error occurred	

GETPS – Block for remote string parameter acquirement

Block Symbol

Licence: [STANDARD](#)



Function Description

The **GETPS** block has the same function as the [GETPR](#), [GETPI](#), and [GETPB](#) blocks, differing only in that it acquires a string parameter value.

Note: The [GETPX](#) and [INCONN](#) blocks can also be used to read the value remotely.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

GET	Input for initiating one-shot parameter read	Bool
-----	--	------

Parameter

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	
nmax	Allocated size of string	Long (I32)

Output

sy	Parameter value	String
E	Error indicator	Bool
	off ... No error	
	on An error occurred	

GETPX – Block for remote parameter acquirement

Block Symbol

Licence: [STANDARD](#)



Function Description

The **GETPX** block works on the same principle as the [GETPB](#), [GETPI](#), [GETPR](#) and [GETPS](#) blocks. However, unlike these blocks, it is universal and can read parameters of all types except array. The name of the remote parameter is entered in the **sc** parameter in the same way as with the other blocks. The value type is set by the **type** parameter, and the parameter reading mode is set by the **GETF** parameter. If the **GETF** parameter is set to **on**, the block reads the parameter value only when requested at the **GET** input. If the **GETF** parameter is set to **off**, the block reads the parameter value continuously.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

GET	Input for initiating one-shot parameter read	Bool
------------	--	-------------

Parameter

sc	String connection to the parameter	String
GETF	Get parameter only when forced to off ... Remote parameter is continuously read on ... One-shot mode	Bool
type	Data type of item 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	⊙8 Byte (U8)

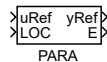
Output

y	Parameter value	Any
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 logical input **LOC** (LOCal) determines whether the value of the internal parameter **apar** is read from the input **uRef**. In this case **LOC** = **off**. If the block is in the local mode (**LOC** = **on**), the internal parameter **apar** stores the last value that was on the input **uRef** just before the local mode was activated (**LOC** = **off** → **on**).

The output value is equivalent to the value of the parameter (**yRef** = **apar**).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uRef	Array reference	Reference
LOC	Activation of local mode	Bool
	off ... The parameter follows the input	
	on Local mode active	

Parameter

SETS	Set array size flag	Bool
nmax	Allocated size of array	↓10 ⊙100 Long (I32)
etype	Type of elements	⊙8 Long (I32)
	1 Bool	
	2 Byte (U8)	
	3 Short (I16)	
	4 Long (I32)	
	5 Word (U16)	
	6 DWord (U32)	
	7 Float (F32)	
	8 Double (F64)	
	10 Large (I64)	
apar	Internal value of parameter	⊙[0.0 1.0 2.0 3.0 4.0 5.0] Double (F64)

Output

yRef	Array reference	Reference
------	-----------------	-----------

PARE – Block with input-defined enumeration parameter

Block Symbol

Licence: [STANDARD](#)



Function Description

The block is similar to 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 **CNE** block: `<number1>: <description1>|<number2>: <description2>|<number3>: <description3> ...`

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

ip	Parameter value	Long (I32)
LOC	Activation of local mode	Bool
	off ... The parameter follows the input	
	on Local mode active	

Parameter

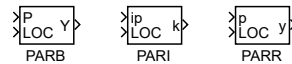
ipar	Internal value of parameter	⊙1 Long (I32)
pupstr	Popup list definition	String
	⊙1: option A 2: option B 3: option C	
NUM	Number in string output	Bool
LIST	Bitfield mode	Bool
SATF	Saturation flag	Bool

Output

iy	Integer output of the block	Long (I32)
sy	String output value	String

PARB, PARI, PARR – Blocks with input-defined parameter

Block Symbols

 Licence: [STANDARD](#)


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,
- **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**.

Note: The [PARX](#) block works on the same principle, but it can set parameters of all types. Consider also using the [SHLD](#) block, which can be used for storing numerical values, similarly to the **PARR** block.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

p	Parameter value	Double (F64)
LOC	Activation of local mode	Bool
	off ... The parameter follows the input	
	on Local mode active	

Parameter

par	Internal value of parameter	©1.0 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)

Output

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

PARS – Block with input-defined string parameter

Block Symbol

Licence: [STANDARD](#)



Function Description

The **PARS** block has the same function as the **PARR**, **PARI**, and **PARB** blocks, differing only in that the set parameter **spar** is a string and is set by changing the input **sp**.

Note: The **PARX** and **INCONN** blocks can also be used to change the string value of a remote parameter.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

sp	Parameter value	String
LOC	Activation of local mode	Bool

Parameter

spar	Internal value of parameter	String
nmax	Allocated size of string	Long (I32)

Output

sy	String output of the block	String
----	----------------------------	--------

PARX – Block with input-defined parameter

Block Symbol

Licence: [STANDARD](#)



Function Description

The **PARX** block, like the [PARB](#), [PARI](#), [PARR](#) and [PARS](#) blocks, allows changing one of its parameters by changing the input. Unlike the blocks mentioned above, the **PARX** block parameter can be of any type. The parameter type is set with the **type** parameter. It is possible to set parameter saturation limits for relevant types.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

p	Parameter value	Any
LOC	Activation of local mode	Bool
	off ... The parameter follows the input	
	on Local mode active	

Parameter

par	Internal value of parameter	⊙1.0	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)

type	Data type of item	⊙8 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	

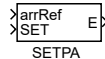
Output

y	Parameter value	Any
---	-----------------	-----

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 with a **'.'** character, indicating the level where the **SETPA** block is placed. Examples of paths: **".CNDR:yp"**, **".Lights.ATMT:touts"**.
 - Starts with **'..'** characters, indicating a level above the **SETPA** block. Examples of paths: **"...CNDR:yp"**, **"...Lights.ATMT:touts"**.
- **Relative to Task:** Starts at the root level of the task where the **SETPA** block is located. The string has to be prefixed with **'%'** in this case. Examples of paths: **"%CNDR:yp"**, **"%Lights.ATMT:touts"**.
- **Absolute:** A 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 presented in a tree-like structure within the **Diagnostics** section of the **REXYGEN Studio** program.

Warning: If the remote parameter is in a task other than the **SETPA** block, block execution is delayed until the remote task is completed. It is necessary to avoid the so-called race conditions and guarantee the correct value setting. Therefore, it is recommended to include the **SETPA** block in a slower task (longer period/execution time) and

set parameter in a faster task (shorter period/execution time). In the opposite situation (e.g. the **SETPA** block in a faster task), the **GETPA** block should be used in a slower task.

Note 1: When using multiple **SETPA** blocks, it is not guaranteed that all data will be written to the remote task in the same tick. It is only guaranteed that the previous block will set a value in the same or previous period as the next block. The execution order can be seen in the **REXYGEN Studio** program diagnostics.

Note 2: The remote parameter must be a primary array (for example **CNA:acn**, **RTOV:xVec**, **MX_MAT:ay**). The array reference (like **CNA:vec**, **RTOV:yVec**, **SUBSYSTEM:Outport**) is not supported.

Note 3: The **OUTCONN** block can also be used for writing the value remotely.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

arrRef	Array reference	Reference
SET	Input for initiating one-shot parameter write	Bool

Parameter

sc	String connection to the parameter	String
SETF	Set parameter only when forced to off ... Remote parameter is continuously updated on One-shot mode	Bool
SETS	Set array size flag	Bool

Output

E	Error indicator off ... No error on An error occurred	Bool
----------	--	-------------

SETPB, SETPI, SETPR – Blocks for remote parameter setting

Block Symbols

Licence: [STANDARD](#)



Function Description

The **SETPR**, **SETPI**, **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. To comply with the naming convention for variables [1.3](#), the inputs and outputs of individual blocks are named according to the type of the set parameter:

- **p, y** – real input and output of the **SETPR** block,
- **ip, k** – integer input and output of the **SETPI** block,
- **P, Y** – Boolean input and output for the **SETPB** block.

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 with a **'.'** character, indicating the level where the **SETPR** block (or **SETPI, SETPB**) is placed. Examples: **".GAIN:k"**, **".Motor1.Position:ycn"**.
 - Starts with **'..'** characters, indicating a level above the **SETPR** block (or **SETPI, SETPB**). Examples: **"..GAIN:k"**, **"..Motor1.Position:ycn"**.

- **Relative to Task:** Starts at the root level of the task where the **SETPR** block (or **SETPI**, **SETPB**) is located. The string has to be prefixed with **'%'** in this case. Examples: **"%GAIN:k"**, **"%Motor1.Position:ycn"**.
- **Absolute:** A 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: **"task1.inputs.lin1:u2"**, **"&EfaDrv.measurements.DER1:n"**.

The order and names of individual hierarchic levels are presented in a tree-like structure within the **Diagnostics** section of the **REXYGEN Studio** program.

Warning: If the remote parameter is in a task other than the **SETPx** block, block execution is delayed until the remote task is completed. It is necessary to avoid the so-called race conditions and guarantee the correct value setting. Therefore, it is recommended to include the **SETPx** block in a slower task (longer period/execution time) and set parameter in a faster task (shorter period/execution time). In the opposite situation (e.g. the **SETPx** block in a faster task), the **GETPx** block should be used in a slower task.

Note 1: When using multiple **SETPx** blocks, it is not guaranteed that all data will be written to the remote task in the same tick. It is only guaranteed that the previous block will set a value in the same or previous period as the next block. The execution order can be seen in the **REXYGEN Studio** program diagnostics. To send multiple values in the same period, it is needed to use the **Inport** and **Outport** blocks or the **SETPA** block.

Note 2: The **SETPX** and **OUTCONN** blocks can also be used for writing the value remotely.

This block propagates the signal quality. More information can be found in the 1.4 section.

Input

p	Desired parameter value	Double (F64)
SET	Input for initiating one-shot parameter write	Bool

Parameter

sc	String connection to the parameter	String
SETF	Set parameter only when forced to	Bool
	off ... Remote parameter is continuously updated	
	on One-shot mode	

Output

y	Parameter value	Double (F64)
E	Error indicator	Bool
	off ... No error	
	on An error occurred	

SETPS – Block for remote string parameter setting

Block Symbol

Licence: [STANDARD](#)



Function Description

The **SETPS** block has the same function as the [SETPR](#), [SETPI](#), and [SETPB](#) blocks, differing only in that it sets a string parameter value.

Note: The [SETPX](#) and [OUTCONN](#) blocks can also be used to set the value of a remote parameter.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

<code>sp</code>	Desired parameter value	String
<code>SET</code>	Input for initiating one-shot parameter write	Bool

Parameter

<code>sc</code>	String connection to the parameter	String
<code>SETF</code>	Set parameter only when forced to	Bool
<code>nmax</code>	Allocated size of string	Long (I32)

Output

<code>sy</code>	Parameter value	String
<code>E</code>	Error indicator	Bool

SETPX – Block for remote parameter setting

Block Symbol

Licence: [STANDARD](#)



Function Description

The **SETPX** block works on the same principle as the [SETPB](#), [SETPI](#), [SETPR](#) and [SETPS](#) blocks. However, unlike these blocks, it is universal and can set parameters of all types except array. The name of the remote parameter is entered in the **sc** parameter in the same way as with the other blocks. The value type is set by the **type** parameter, and the parameter setting mode is set by the **SETF** parameter. If the **SETF** parameter is set to **on**, the block sets the parameter value only when requested at the **SET** input. If the **SETF** parameter is set to **off**, the block sets the parameter value continuously.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

p	Desired parameter value	Any
SET	Input for initiating one-shot parameter write	Bool

Parameter

sc	String connection to the parameter	String
SETF	Set parameter only when forced to off ... Remote parameter is continuously updated on ... One-shot mode	Bool
type	Data type of item 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	⊙8 Byte (U8)

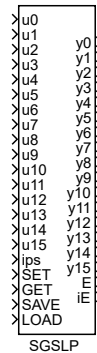
Output

y	Parameter value	Any
E	Error indicator	Bool
	off ... No error	
	on An error occurred	

SGSLP – Set, get, save and load parameters

Block Symbol

Licence: [ADVANCED](#)



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. **<block>:<param>** – Specifies one function block named **block** and its parameter **param**. The same block and parameter are used for all **nps** parameter sets in this case.
2. **<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 **<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 [ATMT](#) block from the **LOGIC** library).

The **broot** parameter is suitable when all blocks whose parameters are to be controlled 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 $0 \leq \text{ips0} \leq \text{nps} - 1$. 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**, $i = 0, \dots, m$, 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 **SIL0** block if working with only a few values.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

u0..u15	Analog input of the block	Double (F64)
ips	Parameter set index	Long (I32)
SET	Set parameters	Bool
GET	Get parameters	Bool
SAVE	Save parameters	Bool
LOAD	Load parameters	Bool

Parameter

nps	Number of parameter sets	↓1 ↑16 ⊙1	Long (I32)
ips0	Initial parameter set index	↓-1 ↑15	Long (I32)
iprec	Precision of parameters (number of digits)	↓2 ↑15 ⊙12	Long (I32)
icolw	Column width in status file	↓0 ↑22	Long (I32)
fname	Name of persistent storage file	⊙status	String
broot	Root block in hierarchy	⊙.	String
sc0..sc15	List of connected parameters		String

Output

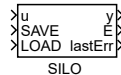
y0..y15	Analog output of the block	Double (F64)
E	Error indicator	Bool
	off ... No error	
	on An error occurred	

iE	Error or warning code	Long (I32)
0 Operation successful	
1 Fatal error of the Matlab system	
2 LOAD operation error	
3 SAVE operation error	
4 Incorrect file format	
5 The ips parameter set not found	
6 Parameter not found, name mismatch	
7 Unexpected end of file	
8 Error writing to file (disk full?)	
9 Parameter syntax error	
10 Only whitespace in the parameter name	
11 Error creating the backup file	
12 GET operation error	
13 SET operation error	
14 Timeout	
15 The specified parameter is read-only	
16 The ips parameter is out of range	

SILO – Save input value, load output value

Block Symbol

Licence: [STANDARD](#)



Function Description

The **SILO** 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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

u	Input signal	Double (F64)
SAVE	Save value to file	Bool
LOAD	Load value from file	Bool

Parameter

fname	Name of persistent storage file	String
CSF	Continuous saving	Bool
CLF	Continuous loading	Bool
yerr	Substitute value for an error case	Double (F64)

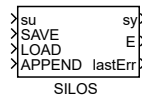
Output

y	Output signal	Double (F64)
E	Error indicator	Bool
	off ... No error	
	on An error occurred	
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 the **APPEND** input is set to **on**, the string from the input is appended to the end of the file. This mode is suitable for logging events to text files. This entry has no effect on loading from a file.

The **LLO** 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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

su	String input of the block	⊙0	String
SAVE	Save string to file		Bool
LOAD	Load string from file		Bool
APPEND	Append saved string to file		Bool

Parameter

fname	Name of persistent storage file	String
--------------	---------------------------------	--------

CSF	Continuous saving	Bool
CLF	Continuous loading	Bool
LL0	Last line only loading	Bool
nmax	Allocated size of string	↓0 ↑65520 Long (I32)

Output

sy	String output of the block	String
E	Error indicator	Bool
	off ... No error	
	on An error occurred	
lastErr	Result of last operation	Long (I32)

Chapter 13

MODEL – Dynamic systems simulation

Contents

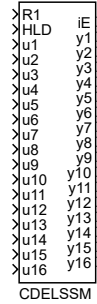
CDELSSM – Continuous state space model with time delay	399
CSSM – Continuous state space model	402
DDELSSM – Discrete state space model with time delay	405
DFIR – Discrete finite input response filter	407
DSSM – Discrete state space model	408
EKF – Extended (nonlinear) Kalman filter	411
FOPDT – First order plus dead-time model	414
IPEN2, IPEN3 – N-link inverted pendulum on cart - Physical parameters	415
IPEN2pu, IPEN3pu – N-link inverted pendulum on cart - Dynamic parameters	418
MDL – Process model	421
MDLI – Process model with input-defined parameters	422
MVD – Motorized valve drive	423
NSSM – Nonlinear State-Space Model	424
NUREACT – Model of nuclear reactor	427
QCOPT – Model of quadrucopter	428
SGEN – Synchronous generator model	430
SGENTX – Synchronous generator model	432
SOPDT – Second order plus dead-time model	434
STMGEN – Model of steam generator	436
STURB – Steam turbine model	438

The MODEL library is centered around system modeling and simulation. It includes blocks like **CSSM** and **DSSM** for continuous and discrete state-space models, and **DFIR** for digital finite impulse response filters. The library offers **EKF** for Extended Kalman Filter implementations, and **FOPDT**, **SOPDT** for first and second order process time delay models. Additionally, it provides **FMUCS** and **FMUINFO** for interfacing with Functional Mock-up Units, and **MDL**, **MDLI** for generic model interfaces. Advanced functionalities are covered by blocks like **CDELSSM**, **DDELSSM** for continuous and discrete state space models of a linear system with time delay, and **MVD** for model variable delays, catering to a wide range of modeling requirements in REXYGEN system.

CDELSSM – Continuous state space model with time delay

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **CDELSSM** block (Continuous State Space Model with time DELay) simulates behavior of a linear system with time delay *del*:

$$\begin{aligned}\frac{dx(t)}{dt} &= A_c x(t) + B_c u(t - del), \quad x(0) = x_0 \\ y(t) &= C_c x(t) + D_c u(t),\end{aligned}$$

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. If **UD=off**, the matrix D_c is not used during simulation (it behaves as if it were zero).

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:

$$\begin{aligned}x((k+1)T) &= A_d x(kT) + B_{d1} u((k-d)T) + B_{d2} u((k-d+1)T), \quad x(0) = x_0 \\ y(kT) &= C_c x(kT) + D_c u(kT),\end{aligned}$$

where $k \in \{1, 2, \dots\}$ is the simulation step, T is the execution period of the block in seconds and d is a delay in simulation step such that $(d-1)T < del \leq d.T$. The period T is not entered in the block, it is determined automatically as a period of the task ([TASK](#), [QTASK](#) nebo [IOTASK](#)) containing the block.

Inputs of the simulated system **u1..u16** represent the input vector **u(t)**. For a given simulation, the first m inputs are used, where m is the number of columns of the matrix **Bc**. 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:

$$\begin{aligned} A_d &= e^{A_c T} \\ B_{d1} &= e^{A_c(T-\Delta)} \int_0^\Delta e^{A_c \tau} B_c d\tau \\ B_{d2} &= \int_0^{T-\Delta} e^{A_c \tau} B_c d\tau, \end{aligned}$$

where $\Delta = del - (d-1)T$.

Computation of discrete matrices A_d , B_{d1} and B_{d2} is based on a method described in [7], 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. Outputs of the simulated system **y1..y16** represent the state of the system **x(t)** and for a given simulation, the first p outputs are used, where p is the number of rows of the matrix **Cc**.

The output **iE** is an integer and contains information about the simulation progress:

- **0**: everything is OK, the block simulates correctly
- **-213**: incompatibility of the dimensions of the state space model matrices
- **-510**: the task is ill-conditioned (one of the working matrices is singular or close to a singular matrix)
- **xxx**: error code **xxx** of the REXYGENsystem, see more in Appendix C

This block propagates the signal quality. More information can be found in the 1.4 section.

Input

R1	Block reset	Bool
HLD	Hold current model state	Bool
u1..u16	Analog input of the block	Double (F64)

Parameter

UD	Matrix Dc usage	Bool
del	Model delay [s]	↓0.0 Double (F64)
is	Padé approximation order	↓0 ↑4 ⊙2 Long (I32)
eps	Approximation accuracy	↓0.0 ↑1.0 ⊙1e-15 Double (F64)
Ac	Matrix A of the continuous model	Double (F64)
	⊙[-0.36 -1.24 -0.18; 1 0 0; 0 1 0]	
Bc	Matrix B of the continuous model	⊙[0.5; 0; 0] Double (F64)

Cc	Matrix C of the continuous model	⊙[0.12 0.48 0.36]	Double (F64)
Dc	Matrix D of the continuous model	⊙[0]	Double (F64)
x0	Initial value of the state x	⊙[0; 0; 0]	Double (F64)

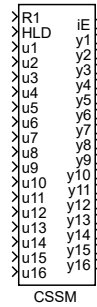
Output

iE	Error code	Error
y1..y16	Analog output of the block	Double (F64)

CSSM – Continuous state space model

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **CSSM** block (Continuous State Space Model) simulates behavior of a linear system:

$$\begin{aligned}\frac{dx(t)}{dt} &= A_c x(t) + B_c u(t), \quad x(0) = x_0 \\ y(t) &= C_c x(t) + D_c u(t),\end{aligned}$$

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. If **UD=off**, the matrix D_c is not used during simulation (it behaves as if it were zero).

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:

$$\begin{aligned}x((k+1)T) &= A_d x(kT) + B_d u(kT), \quad x(0) = x_0 \\ y(kT) &= C_c x(kT) + D_c u(kT),\end{aligned}$$

where $k \in \{1, 2, \dots\}$ 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 ([TASK](#), [QTASK](#) nebo [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:

$$\begin{aligned} A_d &= e^{A_c T} \\ B_d &= \int_0^T e^{A_c \tau} B_c d\tau \end{aligned}$$

Computation of discrete matrices A_d and B_d is based on a method described in [7], 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. Outputs of the simulated system $y1..y16$ represent the state of the system $\mathbf{x}(\mathbf{t})$ and for a given simulation, the first p outputs are used, where p is the number of rows of the matrix \mathbf{C}_c .

The output \mathbf{iE} is an integer and contains information about the simulation progress:

- **0**: everything is OK, the block simulates correctly
- **-213**: incompatibility of the dimensions of the state space model matrices
- **-510**: the task is ill-conditioned (one of the working matrices is singular or close to a singular matrix)
- **xxx**: error code **xxx** of the REXYGENsystem, see more in Appendix C

This block propagates the signal quality. More information can be found in the 1.4 section.

Input

R1	Block reset	Bool
HLD	Hold current model state	Bool
u1..u16	Analog input of the block	Double (F64)

Parameter

UD	Matrix Dc usage	Bool
is	Padé approximation order	↓0 ↑4 ⊙2 Long (I32)
eps	Approximation accuracy	↓0.0 ↑1.0 ⊙1e-15 Double (F64)
Ac	Matrix A of the continuous model	Double (F64)
	⊙[-0.36 -1.24 -0.18; 1 0 0; 0 1 0]	
Bc	Matrix B of the continuous model	⊙[0.5; 0; 0] Double (F64)
Cc	Matrix C of the continuous model	⊙[0.12 0.48 0.36] Double (F64)
Dc	Matrix D of the continuous model	⊙[0] Double (F64)
x0	Initial value of the state x	⊙[0; 0; 0] Double (F64)

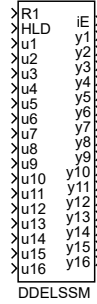
Output

iE	Error code	Error
y1..y16	Analog output of the block	Double (F64)

DDELSSM – Discrete state space model with time delay

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **DDELSSM** block (Discrete State Space Model with time DELay) simulates behavior of a linear system with time delay *del*:

$$\begin{aligned} x(k+1) &= A_d x(k) + B_d u(k-d), \quad x(0) = x_0 \\ y(k) &= C_d x(k) + D_d u(k), \end{aligned}$$

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. If **UD=off**, the matrix D_d is not used during simulation (it behaves as if it were zero). Number of steps of the delay d is the largest integer such that $d.T \leq del$, 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 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. Outputs of the simulated system **y1..y16** represent the state of the system **x(t)** and for a given simulation, the first p outputs are used, where p is the number of rows of the matrix **Cd**.

The output **iE** is an integer and contains information about the simulation progress:

- **0**: everything is OK, the block simulates correctly
- **-213**: incompatibility of the dimensions of the state space model matrices
- **xxx**: error code **xxx** of the REXYGENsystem, see more in [Appendix C](#)

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

R1	Block reset	Bool
HLD	Hold current model state	Bool
u1..u16	Analog input of the block	Double (F64)

Parameter

UD	Matrix Dd usage	Bool
del	Model delay [s]	↓0.0 Double (F64)
Ad	Matrix A of the discrete model	Double (F64)
	⊙[0.235700090 -0.904208075 -0.120785644; 0.671031354 0.477271377 -0.072129196; 0	
Bd	Matrix B of the discrete model	Double (F64)
	⊙[0.335515677; 0.200358878; 0.071773902]	
Cd	Matrix C of the discrete model	⊙[0.12 0.48 0.36] Double (F64)
Dd	Matrix D of the discrete model	⊙[0] Double (F64)
x0	Initial value of the state x	⊙[0; 0; 0] Double (F64)

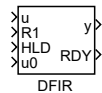
Output

iE	Error code	Error
y1..y16	Analog output of the block	Double (F64)

DFIR – Discrete finite input response filter

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **DFIR** block is a filter whose impulse response (or response to any finite length input) is of finite duration, because it settles to zero in finite time. The calculation takes place in the form of a convolutional integral (sum) - the impulse characteristic is entered in the **hk** field already in discretized form for the correct period.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

RST	Block reset	Bool
HLD	Hold	Bool
u0	Initial input value (fill buffer)	Double (F64)

Parameter

nmax	Allocated size of array	$\downarrow 10 \uparrow 10000000 \odot 100$	Long (I32)
hk	Discrete impulse response	$\odot [0.6 \ 0.3 \ 0.1]$	Double (F64)

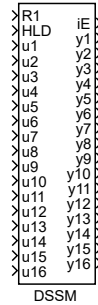
Output

y	Analog output of the block	Double (F64)
RDY	Outputs valid (ready flag)	Bool

DSSM – Discrete state space model

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **DSSM** block (Discrete State Space Model) simulates behavior of a linear system:

$$\begin{aligned} x(k+1) &= A_d x(k) + B_d u(k), \quad x(0) = x_0 \\ y(k) &= C_d x(k) + D_d u(k), \end{aligned}$$

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. If **UD=off**, the matrix D_d is not used during simulation (it behaves as if it were zero).

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. Outputs of the simulated system **y1..y16** represent the state of the system **x(t)** and for a given simulation, the first p outputs are used, where p is the number of rows of the matrix **Cd**.

The output **iE** is an integer and contains information about the simulation progress:

- **0**: everything is OK, the block simulates correctly
- **-213**: incompatibility of the dimensions of the state space model matrices
- **xxx**: error code **xxx** of the REXYGENsystem, see more in [Appendix C](#)

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

R1	Block reset	Bool
HLd	Hold current model state	Bool
u1..u16	Analog input of the block	Double (F64)
u	Analog input of the block	Double (F64)

Parameter

UD	Matrix Dd usage	Bool
Ad	Matrix A of the discrete model $\odot [0.235700090 \ -0.904208075 \ -0.120785644; \ 0.671031354 \ 0.477271377 \ -0.072129196; \ 0.000000000 \ 0.000000000 \ 0.000000000]$	Double (F64)
Bd	Matrix B of the discrete model $\odot [0.335515677; \ 0.200358878; \ 0.071773902]$	Double (F64)
Cd	Matrix C of the discrete model $\odot [0.12 \ 0.48 \ 0.36]$	Double (F64)
Dd	Matrix D of the discrete model $\odot [0]$	Double (F64)
x0	Initial value of the state x $\odot [0; \ 0; \ 0]$	Double (F64)

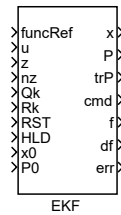
Output

iE	Error code	Error
y1..y16	Analog output of the block	Double (F64)

EKF – Extended (nonlinear) Kalman filter

Block Symbol

Licence: [MODEL](#)



Function Description

The block implements a nonlinear state estimator known as Extended Kalman filter. The goal is to provide estimates of unmeasurable state quantities of a nonlinear dynamic system described by a state space model in the form

$$dx/dt = f(x, u) + w(t), y = h(x, u) + v(t)$$

for a continuous-time case and

$$x(k+1) = f(x(k), u(k)) + w(k), y(k) = h(x(k), u(k)) + v(k)$$

for the case of a discrete-time system. The variables w, v are the process and observation noises which are both assumed to be zero mean multivariate Gaussian processes with covariance Q and R specified in the block parameters. The Extended Kalman filter is the nonlinear version of the Kalman filter which linearizes the state and output equations about the current working point. It is a predictor-corrector type algorithm which switches between open-loop prediction using the state equation and correction of the estimates by directly measured output quantities. The measurements can be supplied to the filter non-equidistantly in an arbitrary execution period of the block.

The prediction step is run in each execution period and solves the state equation by numerical integration, starting from an initial value $x0$ and initial covariance $P0$. Various numerical methods, chosen by the user specified parameter *solver*, are available to perform the integration of the vector state differential equation. A special choice of *solver* = 1 signalizes the discrete-time system case for which the numerical integration reduces to simple evaluation of the recursive formula given by the first-order difference equation in $x(k+1) = f(x(k), u(k))$. Apart from the state vector, also its covariance matrix P is propagated in time, capturing the uncertainty of the estimates in the form of their (co)variances. Please refer to the documentation of the [NSSM](#) block for more details about the available numerical integration algorithms.

The filtering correction step takes place whenever the input of the block is set to $nz > 0$. This signalizes that new vector of measurements is available at the z input and

it is used to correct the state and its covariance estimates from the prediction step. Multiple right sides of the output equation can be implemented in the cooperating **REXLANG** block. This may be useful e.g. for systems equipped with various sensors providing their data asynchronously to each other (and with respect to the block execution times) with different sampling periods. For the setting $nz = 0$, the user algorithm signalizes no output data available in the current execution period, forcing the filter to extrapolate the state estimates by performing the prediction step only.

The Extended Kalman filter is generally not an optimal filter in the sense of minimization of the mean-squared error of the obtained state estimates. However, it provides modest performance for sufficiently smooth nonlinear systems and is considered to be a de facto standard solution for nonlinear estimation. A special case is obtained by setting linear state and output equations in the cooperating **REXLANG** block. This case leads to standard linear Kalman filter which is stochastically optimal for the formulated state estimation problem.

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

funcRef	Cooperating REXLANG block reference	Reference
u	Input vector of the model	Reference
z	Output (measurement) vector of the model	Reference
nz	Index of the actual output vector set	↓1 Long (I32)
Qk	State noise covariance matrix	Reference
Rk	Output noise covariance matrix	Reference
RST	Block reset	Bool
HLD	Hold	Bool
x0	Initial state vector	Reference
P0	Initial covariance matrix	Reference

Parameter

nmax	Allocated size of output matrix (total number of items)	Long (I32)
		↓5 ↑10000 ⊕20

solver	Numeric integration method	⊙2 Long (I32)
1 Discrete equation	
2 Euler (1st order)	
3 2nd order Adams-Bashforth	
4 3rd order Adams-Bashforth	
5 4th order Adams-Bashforth	
6 5th order Adams-Bashforth	
7 4th order Runge-Kutha	
8 implicit Euler	
9 implicit Euler(more iteration)	
10 2nd order Adams-Multon implicit	
11 2nd order Adams-Multon implicit (more iteration)	
12 3rd order Adams-Multon implicit	
13 3rd order Adams-Multon implicit (more iteration)	
14 2nd order RadauIIA implicit	
15 2nd order RadauIIA implicit (more iteration)	
16 —	
17 —	
18 —	
19 —	

Output

x	Model state vector	Reference
P	Model state covariance matrix	Reference
trP	Trace of model state covariance matrix	Reference
cmd	Cooperating REXLANG block requested function	Long (I32)
f	Vector reference set by cooperating REXLANG block	Reference
df	Matrix reference set by cooperating REXLANG block	Reference
err	Error code (0 is OK, see SystemLog for details)	Long (I32)

FOPDT – First order plus dead-time model

Block Symbol

Licence: [STANDARD](#)



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:

$$P(s) = \frac{k0}{(\text{tau} \cdot s + 1)} \cdot e^{-\text{del} \cdot s}$$

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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

k0	Static gain	⊙1.0	Double (F64)
del	Dead time [s]		Double (F64)
tau	Time constant	⊙1.0	Double (F64)
nmax	Allocated size of array	↓10 ↑10000000 ⊕1000	Long (I32)

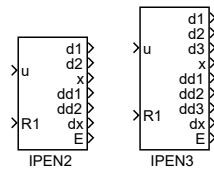
Output

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

IPEN2, IPEN3 – N-link inverted pendulum on cart - Physical parameters

Block Symbols

Licence: [MODEL](#)



Function Description

The **IPEN2** and **IPEN3** blocks simulate the dynamics of double and triple inverted pendulums on a cart, respectively. These models enable users to conduct experiments with various control strategies, making them suitable for both educational and research purposes.

The primary input to the models is an analog signal **u**, interpreted based on the **IACC** parameter setting:

- for **IACC=on**, the input **u** is assumed to be a force acting on the cart [N],
- for **IACC=off**, the models assume the input represents speed [m/s].

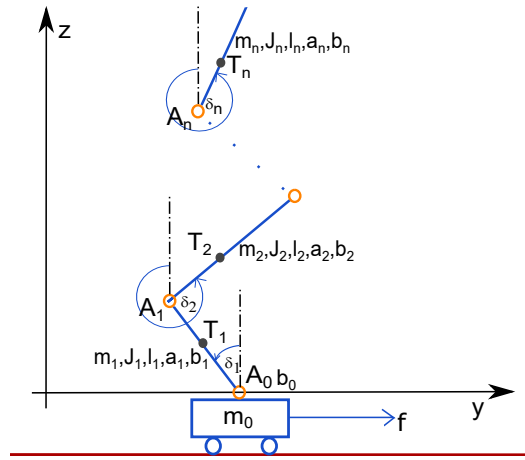
The **R1** signal is used to reset each model to its initial configuration.

Both models can be precisely configured with a series of parameters that reflect the system's physical characteristics. These include the relative center of gravity positions **a**, moments of inertia **J**, lengths **l**, and masses of the pendulums **m**, as well as damping coefficients **b** and the initial state of the system (positions **d_0**, velocities **dd_0**). A schematic representation of the system with parameters is shown below. The parameters are intuitively defined. The relative position of the center of gravity for the *i*-th pendulum, a_i , is determined by the equation

$$|A_{i-1}T_i| = a_i l_i,$$

where A_{i-1} is the position of the previous joint, T_i is the position of the pendulum's center of gravity, and l_i is the length of the pendulum.

The computation of the models adheres to the mathematical model and physical parameters detailed in the literature [8]. The **IPEN2pu** and **IPEN3pu** blocks are used for simulating the inverted pendulum models with dynamic parameters.



This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

u	Analog input of the block	Double (F64)
$R1$	Block reset	Bool

Parameter

$a1$	Relative position of center of gravity	$\odot 1.0$	Double (F64)
$a2$	Relative position of center of gravity	$\odot 1.0$	Double (F64)
$a3$	Relative position of center of gravity	$\odot 1.0$	Double (F64)
$J1$	Moment of inertia of pendulum	$\odot 1.0$	Double (F64)
$J2$	Moment of inertia of pendulum	$\odot 1.0$	Double (F64)
$J3$	Moment of inertia of pendulum	$\odot 1.0$	Double (F64)
$l1$	Length of pendulum [m]	$\odot 1.0$	Double (F64)
$l2$	Length of pendulum [m]	$\odot 1.0$	Double (F64)
$l3$	Length of pendulum [m]	$\odot 1.0$	Double (F64)
$m1$	Mass of pendulum [kg]	$\odot 1.0$	Double (F64)
$m2$	Mass of pendulum [kg]	$\odot 1.0$	Double (F64)
$m3$	Mass of pendulum [kg]	$\odot 1.0$	Double (F64)
$m0$	Mass of cart [kg]	$\odot 1.0$	Double (F64)
$b1$	Damping coefficient of pendulum	$\odot 1.0$	Double (F64)
$b2$	Damping coefficient of pendulum	$\odot 1.0$	Double (F64)
$b3$	Damping coefficient of pendulum	$\odot 1.0$	Double (F64)
$b0$	Damping coefficient of cart	$\odot 1.0$	Double (F64)
$d1_0$	Initial angle of pendulum [rad]	$\odot 1.0$	Double (F64)
$d2_0$	Initial angle of pendulum [rad]	$\odot 1.0$	Double (F64)
$d3_0$	Initial angle of pendulum [rad]	$\odot 1.0$	Double (F64)

x_0	Initial position of cart [m]	⊙1.0	Double (F64)
dd1_0	Initial angular velocity of pendulum [rad/s]	⊙1.0	Double (F64)
dd2_0	Initial angular velocity of pendulum [rad/s]	⊙1.0	Double (F64)
dd3_0	Initial angular velocity of pendulum [rad/s]	⊙1.0	Double (F64)
dx_0	Initial velocity of cart [m/s]	⊙1.0	Double (F64)
IACC	on=Input u is velocity, off=Input u is force		Bool

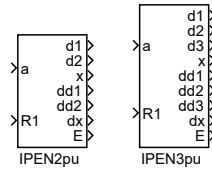
Output

d1	Angle of pendulum [rad]	Double (F64)
d2	Angle of pendulum [rad]	Double (F64)
d3	Angle of pendulum [rad]	Double (F64)
x	Position of cart [m]	Double (F64)
dd1	Angular velocity of pendulum [rad/s]	Double (F64)
dd2	Angular velocity of pendulum [rad/s]	Double (F64)
dd3	Angular velocity of pendulum [rad/s]	Double (F64)
dx	Velocity of cart [m/s]	Double (F64)
E	Error indicator	Bool

IPEN2pu, IPEN3pu – N-link inverted pendulum on cart - Dynamic parameters

Block Symbols

Licence: [MODEL](#)



Function Description

The IPEN2pu and IPEN3pu blocks simulate the dynamics of double and triple inverted pendulums on a cart, respectively. These models enable users to conduct experiments with various control strategies, making them suitable for both educational and research purposes.

For both models, the primary input is an analog signal **a**, which denotes the acceleration of the cart $[m/s^2]$. The **R1** signal is used to reset each model to its initial configuration.

Both models can be precisely configured with using the system's dynamic parameters **p1** - **p11** and the initial state of the system (positions **d_0**, velocities **dd_0**). The dynamic parameters can be determined using the physical parameters defined in the [IPEN2](#) and [IPEN3](#) blocks and the equations listed below. The model details are thoroughly described in the literature [8].

For the IPEN2pu model, the dynamic parameters are defined as follows:

$$\begin{aligned} p_1 &= \frac{(m_1 a_1^2 + m_2) l_1^2 + J_1}{m_2 a_2 l_2 l_1}, & p_2 &= \frac{a_2^2 l_2^2 m_2 + J_2}{m_2 a_2 l_2 l_1}, & p_3 &= \frac{m_1 a_1 + m_2}{m_2 a_2 l_2}, \\ p_4 &= \frac{1}{l_1}, & p_5 &= \frac{b_1}{m_2 a_2 l_2 l_1}, & p_6 &= \frac{b_2}{m_2 a_2 l_2 l_1}. \end{aligned}$$

For the IPEN3pu model, the dynamic parameters are defined as follows:

$$\begin{aligned} p_1 &= \frac{l_1(m_2 a_2 + m_3)}{a_3 l_3 m_3}, & p_2 &= \frac{l_1}{l_2}, & p_3 &= \frac{(m_1 a_1^2 + m_2 + m_3) l_1^2 + J_1}{a_3 l_3 m_3 l_2}, \\ p_4 &= \frac{(m_2 a_2^2 + m_3) l_2^2 + J_2}{a_3 l_3 m_3 l_2}, & p_5 &= \frac{a_3^2 l_3^2 m_3 + J_3}{a_3 l_3 m_3 l_2}, & p_6 &= \frac{l_1(m_1 a_1 + m_2 + m_3)}{a_3 l_3 m_3 l_2}, \\ p_7 &= \frac{a_2 m_2 + m_3}{a_3 l_3 m_3}, & p_8 &= \frac{1}{l_2}, & p_9 &= \frac{b_1}{a_3 l_3 m_3 l_2}, & p_{10} &= \frac{b_2}{a_3 l_3 m_3 l_2}, & p_{11} &= \frac{b_3}{a_3 l_3 m_3 l_2}. \end{aligned}$$

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

a	Acceleration of cart [m/s ²]	Double (F64)
R1	Block reset	Bool

Parameter

p1..p11	Dynamic model parameter (see model equation)	⊙1.0 Double (F64)
d1_0	Initial angle of pendulum [rad]	⊙1.0 Double (F64)
d2_0	Initial angle of pendulum [rad]	⊙1.0 Double (F64)
d3_0	Initial angle of pendulum [rad]	⊙1.0 Double (F64)
x_0	Initial position of cart [m]	⊙1.0 Double (F64)
dd1_0	Initial angular velocity of pendulum [rad/s]	⊙1.0 Double (F64)
dd2_0	Initial angular velocity of pendulum [rad/s]	⊙1.0 Double (F64)
dd3_0	Initial angular velocity of pendulum [rad/s]	⊙1.0 Double (F64)
dx_0	Initial velocity of cart [m/s]	⊙1.0 Double (F64)

Output

d1	Angle of pendulum [rad]	Double (F64)
d2	Angle of pendulum [rad]	Double (F64)
d3	Angle of pendulum [rad]	Double (F64)
x	Position of cart [m]	Double (F64)
dd1	Angular velocity of pendulum [rad/s]	Double (F64)
dd2	Angular velocity of pendulum [rad/s]	Double (F64)
dd3	Angular velocity of pendulum [rad/s]	Double (F64)
dx	Velocity of cart [m/s]	Double (F64)
E	Error indicator	Bool

MDL – Process model

Block Symbol

Licence: [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 **k0**, $D \geq 0$ is the time-delay **del** and $\tau_1, \tau_2 > 0$ are the system time-constants **tau1** and **tau2**.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

k0	Static gain	⊙1.0	Double (F64)
del	Dead time [s]		Double (F64)
tau1	The first time constant	⊙1.0	Double (F64)
tau2	The second time constant	⊙2.0	Double (F64)
nmax	Allocated size of array	↓10 ↑100000000 ⊙1000	Long (I32)

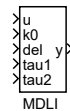
Output

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

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 `k0`, $D \geq 0$ is the time-delay `del` and $\tau_1, \tau_2 > 0$ are the system time-constants `tau1` and `tau2`. In contrary to the [MDL](#) block the system is time variant. The system parameters are determined by the input signals.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

<code>u</code>	Analog input of the block	Double (F64)
<code>k0</code>	Static gain	Double (F64)
<code>del</code>	Dead time [s]	Double (F64)
<code>tau1</code>	The first time constant	Double (F64)
<code>tau2</code>	The second time constant	Double (F64)

Parameter

<code>nmax</code>	Allocated size of array	$\downarrow 10 \uparrow 100000000 \odot 1000$	Long (I32)
-------------------	-------------------------	---	------------

Output

<code>y</code>	Analog output of the block	Double (F64)
----------------	----------------------------	--------------

MVD – Motorized valve drive

Block Symbol

Licence: [STANDARD](#)



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 $y = hilim$ (full closed $y = lolim$) position is reached. The full open (full closed) position is signaled by the end switch HS (LS). The initial position at start-up is $y = y0$. If $UP = DN = on$ or $UP = DN = off$, then the position of the valve remains unchanged (neither opening nor closing).

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

UP	Open	Bool
DN	Close	Bool

Parameter

y0	Initial valve position	Double (F64)
tv	Transition time [s]	⊙10.0 Double (F64)
hilim	Upper limit position (open)	⊙1.0 Double (F64)
lolim	Lower limit position (closed)	Double (F64)

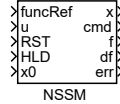
Output

y	Valve position	Double (F64)
HS	Upper end switch	Bool
LS	Lower end switch	Bool

NSSM – Nonlinear State-Space Model

Block Symbol

Licence: [MODEL](#)



Function Description

The block provides a solution to a nonlinear continuous-time state-space model in the form of

$$dx/dt = f(x, u), y = h(x, u)$$

or its discrete-time counterpart defined as

$$x(k+1) = f(x(k), u(k)), y(k) = h(x(k), u(k)).$$

The equation is discretized into a form

$$x(t) = F(x(t-T), u(t)),$$

where T is sampling period of the NSSM block.

The method used for discretization (i.e. a method to numerically solve the vector differential equation) depends on the `solver` parameter. Various methods for numerical integration are implemented including one step methods (like Runge-Kutta, Euler), multistep methods (Adams-Bashforth), and also implicit methods (Adams-Moulton). It is possible to choose different method order for each kind to find a suitable precision vs computational time trade-off.

The block does not support variable step algorithms (the time-step for the solver is always the same as the execution period of the task where the block is inserted).

The non-linear-vector function $f(x, u)$ must be implemented in the [REXLANG](#) block that is connected to the NSSM block in a special way. The input `funcRef` of the NSSM block must be connected to the output `y0` of the [REXLANG](#) block and the output `y0` can not be used internally in the code/script of the [REXLANG](#) block. The outputs `x`, `f` and `df` of the NSSM block must be connected to the inputs of the [REXLANG](#) block. These inputs must be processed in the [REXLANG](#) code as an input array. The main function of the [REXLANG](#) block must set the value of $f(x, u)$ into the `f` vector (e.g. into the input array, where `f` is connected) and the matrix $df(x, u)/dx$ into the `af` matrix.

The NSSM block calls the main-function of the [REXLANG](#) block when needed for numerical integration of the differential equation system (for example the Runge-Kutta method performs 4 calls in each execution period with different `x`-vector values). The [REXLANG](#) block should be disabled in the schematics of the algorithm to prevent its execution REXYGEN system itself. If the [REXLANG](#) must be executed by REXYGEN (e.g. for

compute output function $y = h(x, u)$), it is recommended to connect the output **cmd** of the **NSSM** block into input of the **REXLANG** block to distinguish between calling by the **NSSM** block (**cmd** = 0) and calling by **REXYGEN** system (**cmd** = -1).

Notes:

- computation of the $df(x, u)/dx$ is necessary for implicit methods only (explicit methods do not use it).
- size of the vector **x** (and also **f**, **df**) is defined by the size of the vector **x0**. The size should be changed by reset only (the **RST** input).
- **solver=1: discrete** signalizes a discrete-time state space model with the functions **f** and **h** designating the right side of the corresponding difference equation. This mode does not require numerical integration and the algorithm reduces to the execution of the code in the connected **REXLANG** block; the mode is used mainly for symmetry with the **EKF** block.
- for **NSSM** connecting the output **cmd** is necessary, because **cmd**>0 indicate number of measurement and **REXLANG** must return $f = h(x, u)$, $df = dh(x, u)/dx$.

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

funcRef	Cooperating REXLANG block reference	Reference
u	Input vector of the model	Reference
RST	Block reset	Bool
HLD	Hold	Bool
x0	Initial state vector	Reference

Parameter

nmax	Allocated size of output matrix (total number of items)	Long (I32)
	↓5 ↑10000 ○20	

solver	Numeric integration method	⊙2 Long (I32)
1 Discrete equation	
2 Euler (1st order)	
3 2nd order Adams-Bashforth	
4 3rd order Adams-Bashforth	
5 4th order Adams-Bashforth	
6 5th order Adams-Bashforth	
7 4th order Runge-Kutha	
8 implicit Euler	
9 implicit Euler(more iteration)	
10 2nd order Adams-Multon implicit	
11 2nd order Adams-Multon implicit (more iteration)	
12 3rd order Adams-Multon implicit	
13 3rd order Adams-Multon implicit (more iteration)	
14 2nd order RadauIIA implicit	
15 2nd order RadauIIA implicit (more iteration)	
16 —	
17 —	
18 —	
19 —	

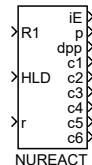
Output

x	Model state vector	Reference
y	Model output vector	Reference
cmd	Cooperating REXLANG block requested function	Long (I32)
f	Vector reference set by cooperating REXLANG block	Reference
df	Matrix reference set by cooperating REXLANG block	Reference
err	Error code (0 is OK, see SystemLog for details)	Long (I32)

NUREACT – Model of nuclear reactor

Block Symbol

Licence: [MODEL](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

R1	Block reset	Bool
HLD	Hold current model state	Bool
r	Reactivity []	Double (F64)

Parameter

p0	Initial reactor power [W]	⊙1.0	Double (F64)
b1..b6	Model parameter beta(see model equation)	⊙1.0	Double (F64)
l1..l6	Model parameter lambda(see model equation)	⊙1.0	Double (F64)
tau	Time constant [s]	⊙1.0	Double (F64)

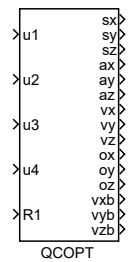
Output

iE	Error code	Bool
p	Reactor thermal power [W]	Double (F64)
dpp	Derivative of reactor thermal power [W]	Double (F64)
c1..c6	Neutron concentration (see model equation)	Double (F64)

QCOPT – Model of quadrucopter

Block Symbol

Licence: [MODEL](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

u1..u4	motor input value	Double (F64)
R1	Block reset	Bool

Parameter

m	Total mass [kg]	⊙1.0	Double (F64)
g	Gravity constant [m/s ²]	⊙1.0	Double (F64)
KT	Rotor trust coefficient [N/u]	⊙1.0	Double (F64)
KTT	Rotor torque coefficient [Nm/u]	⊙1.0	Double (F64)
Ixx	Inertia tensor entry	⊙1.0	Double (F64)
Iyy	Inertia tensor entry	⊙1.0	Double (F64)
Izz	Inertia tensor entry	⊙1.0	Double (F64)
pM	Rotors position matrix [x1 y1 z1; x2 y2 z2; x3 y3 z3; x4 y4 z4]	⊙1.0	Double (F64)

Output

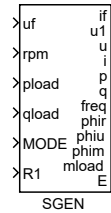
sx	Position of the mass centre [m]	Double (F64)
sy	Position of the mass centre [m]	Double (F64)
sz	Position of the mass centre [m]	Double (F64)

vx	Velocity of the mass centre[m/s]	Double (F64)
vy	Velocity of the mass centre[m/s]	Double (F64)
vz	Velocity of the mass centre[m/s]	Double (F64)
ax	Euler angle [rad]	Double (F64)
ay	Euler angle [rad]	Double (F64)
az	Euler angle [rad]	Double (F64)
ox	derivative of Euler angle [rad/s]	Double (F64)
oy	derivative of Euler angle [rad/s]	Double (F64)
oz	derivative of Euler angle [rad/s]	Double (F64)
vxb	Velocity in the body frame	Double (F64)
vyb	Velocity in the body frame	Double (F64)
vzb	Velocity in the body frame	Double (F64)

SGEN – Synchronous generator model

Block Symbol

Licence: [MODEL](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uf	Feed voltage [V]	Double (F64)
rpm	Rotation per minute	Double (F64)
pload	Load active power [W]	Double (F64)
qload	Load reactive power [VAr]	Double (F64)
MODE	1=grid operation, 0=island operation	Bool
R1	Model state reset	Bool

Parameter

pn	Nominal power [VA]	⊙1.0	Double (F64)
un	Nominal RMS voltage [V]	⊙1.0	Double (F64)
fn	Nominal frequency [Hz]	⊙1.0	Double (F64)
ifn	Nominal feed current [A]		Double (F64)
pp	Number of polpairs	↓1 ↑10000 ⊙2	Long (I32)
ZM	on=constant impedance mode, off=constant power mode		Bool
ra	Stator winding resistance [p.u.]	⊙0.011	Double (F64)
rf	Feed winding resistance [p.u.]	⊙0.0006	Double (F64)
r1d	D-axis damping winding resistance [p.u.]	⊙0.0354	Double (F64)
r1q	Q-axis damping winding resistance [p.u.]	⊙0.0428	Double (F64)
ld	D-axis stator winding self inductance [p.u.]	⊙1.05	Double (F64)
lq	Q-axis stator winding self inductance [p.u.]	⊙0.7	Double (F64)

lad	D-axis damping winding self inductance [p.u.]	⊙0.9	Double (F64)
laq	Q-axis damping winding self inductance [p.u.]	⊙0.55	Double (F64)
lff	Feed winding self inductance [p.u.]	⊙1.571	Double (F64)
lf1d	Feed damping winding mutual inductance [p.u.]	⊙0.9	Double (F64)
l11d	D-axis stator and damping winding mutual inductance [p.u.]	⊙1.1	Double (F64)
l11q	Q-axis stator and damping winding mutual inductance [p.u.]	⊙0.8067	Double (F64)

Output

if	Feed current [A]	Double (F64)
u1	1st phase actual voltage [V]	Double (F64)
u	RMS (phase) voltage [V]	Double (F64)
i	RMS (phase) current [A]	Double (F64)
p	Active power [W]	Double (F64)
q	Reactive power [VA]	Double (F64)
freq	Frequency [Hz]	Double (F64)
phir	Rotor actual angle [rad]	Double (F64)
phiu	Load/torque angle[rad]	Double (F64)
phim	Mains-genertor phase difference [rad]	Double (F64)
mload	Shaft torque [Nm]	Double (F64)
E	Error indicator	Bool

SGENTX – Synchronous generator model

Block Symbol

Licence: [MODEL](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uf	Feed voltage [p.u.]	Double (F64)
pm	Prime mover torque [p.u.]	Double (F64)
pload	Load active power [p.u.]	Double (F64)
qload	Load reactive power [p.u.]	Double (F64)
MODE	1=grid operation, 0=island operation	Bool
R1	Model state reset	Bool

Parameter

um	Mains voltage [p.u.]	⊙0.011	Double (F64)
fm	Mains frequency [p.u.]	⊙0.011	Double (F64)
pn	Nominal power [VA]	⊙10000000.0	Double (F64)
un	Nominal RMS voltage [V]	⊙6000.0	Double (F64)
fn	Nominal frequency [Hz]	⊙1.0	Double (F64)
pp	Number of pole pairs	↓1 ↑1000 ⊙2	Long (I32)
ZM	on=constant impedance mode, off=constant power mode		Bool
J	Mass of inertia [p.u.]	⊙0.0006	Double (F64)
Tdoi	D-axis transient time constant (open winding) [p.u.]	⊙0.0354	Double (F64)
Tdoi i	D-axis subtransient time constant (open winding) [p.u.]	⊙0.0354	Double (F64)
Tqoi	Q-axis transient time constant (open winding) [p.u.]	⊙0.0354	Double (F64)

Tqoii	Q-axis subtransient time constant (open winding) [p.u.]	⊙0.0354	Double (F64)
Xd	D-axis static impedance [p.u.]	⊙0.0428	Double (F64)
Xdi	D-axis transient impedance [p.u.]	⊙0.0428	Double (F64)
Xdii	D-axis subtransient impedance [p.u.]	⊙0.0428	Double (F64)
Xq	Q-axis static impedance [p.u.]	⊙0.0428	Double (F64)
Xqi	Q-axis transient impedance [p.u.]	⊙0.0428	Double (F64)
Xqii	Q-axis subtransient impedance [p.u.]	⊙0.0428	Double (F64)
r	Stator winding resistance [p.u.]	⊙0.011	Double (F64)
Xmd	Feed cross impedance [p.u.]	⊙0.011	Double (F64)
rf	Feed winding resistance [p.u.]	⊙0.011	Double (F64)
rm	Mains line resistance [p.u.]	⊙0.011	Double (F64)
xm	Mains line reactance [p.u.]	⊙0.011	Double (F64)

Output

if	Feed current [p.u.]	Double (F64)
u1	1st phase actual voltage [p.u.]	Double (F64)
u	RMS (phase-to-N) voltage [p.u.]	Double (F64)
i	RMS (phase) current [p.u.]	Double (F64)
p	Active power [p.u.]	Double (F64)
q	Reactive power [p.u.]	Double (F64)
freq	Frequency [p.u.]	Double (F64)
phir	Rotor actual angle [rad]	Double (F64)
phiu	Load/torque angle[rad]	Double (F64)
phim	Mains-genertor phase difference [rad]	Double (F64)
rpm	Rotation per minute	Double (F64)
cos	Power factor	Double (F64)
E	Error indicator	Bool

SOPDT – Second order plus dead-time model

Block Symbol

Licence: [STANDARD](#)



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.

$$\begin{aligned} \text{itf} = 1: \quad P(s) &= \frac{\text{pb1} \cdot s + \text{pb0}}{s^2 + \text{pa1} \cdot s + \text{pa0}} \cdot e^{-\text{del} \cdot s} \\ \text{itf} = 2: \quad P(s) &= \frac{\text{k0} (\text{tau} \cdot s + 1)}{(\text{tau1} \cdot s + 1) (\text{tau2} \cdot s + 1)} \cdot e^{-\text{del} \cdot s} \\ \text{itf} = 3: \quad P(s) &= \frac{\text{k0} \cdot \text{om}^2 \cdot (\text{tau}/\text{om} \cdot s + 1)}{(s^2 + 2 \cdot \text{xi} \cdot \text{om} \cdot s + \text{om}^2)} \cdot e^{-\text{del} \cdot s} \\ \text{itf} = 4: \quad P(s) &= \frac{\text{k0} (\text{tau} \cdot s + 1)}{(\text{tau1} \cdot s + 1) s} \cdot e^{-\text{del} \cdot s} \end{aligned}$$

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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

u Analog input of the block Double (F64)

Parameter

itf	Transfer function form	⊙1 Long (I32)
	1 General	
	2 Real poles	
	3 Complex poles	
	4 Integrating	
k0	Static gain	⊙1.0 Double (F64)

tau	Numerator time constant		Double (F64)
tau1	The first time constant	⊙1.0	Double (F64)
tau2	The second time constant	⊙1.0	Double (F64)
om	Natural frequency	⊙1.0	Double (F64)
xi	Relative damping coefficient	⊙1.0	Double (F64)
pb0	Numerator coefficient: s^0	⊙1.0	Double (F64)
pb1	Numerator coefficient: s^1	⊙1.0	Double (F64)
pa0	Denominator coefficient: s^0	⊙1.0	Double (F64)
pa1	Denominator coefficient: s^1	⊙1.0	Double (F64)
del	Dead time [s]		Double (F64)
nmax	Allocated size of array	↓10 ↑10000000 ⊙1000	Long (I32)

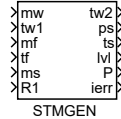
Output

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

STMGEN – Model of steam generator

Block Symbol

Licence: [MODEL](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

mw	Heating water flow from the reactor [kg/s]	Double (F64)
tw1	Reactor heating water temperature [°C]	Double (F64)
mf	Feeding water flow [kg/s]	Double (F64)
tf	Feeding water temperature [°C]	Double (F64)
ms	Output steam flow [kg/s]	Double (F64)
R1	Model reset	Bool

Parameter

dsg	Internal diameter [m]	↓0.0 ⊕4.0	Double (F64)
lsg	Length of the exchanger (pipes) [m]	↓0.0 ⊕13.2	Double (F64)
npipe	Total number of pipes	↓0 ⊕8000	Long (I32)
nseg	Number of segments (rows of pipes)	↓0 ⊕10	Long (I32)
dpipe	External pipe's diameter [m]	↓0.0 ⊕0.04	Double (F64)
thpipe	Pipe's wall thickness [m]	↓0.0 ⊕0.005	Double (F64)
hl	Lowest line of row distance from the bottom [m]	↓0.0 ⊕0.5	Double (F64)
hh	Highest line of row distance from the bottom [m]	↓0.0 ⊕3.5	Double (F64)
vExt	Volume of Main Steam Collector [m ³]	↓0.0	Double (F64)
ww	Average width of water chamber [m]	↓0.0 ⊕0.4	Double (F64)
kq	Thermal transfer constant [kW/K/m ²]	↓0.0 ⊕3.2	Double (F64)
t0	Initial temperature [°C]	↓0.0 ⊕280.0	Double (F64)
lv10	Initial water level (from the bottom) [m]	↓0.0	Double (F64)

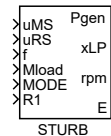
Output

tw2	Water temperature back to the reactor [°C]	Double (F64)
ps	Output steam pressure [MPa]	Double (F64)
ts	Output steam temperature [°C]	Double (F64)
lvl	Feeding water level (from the bottom) [m]	Double (F64)
P	Actual thermal power [kW]	Double (F64)
ierr	Error code	Error

STURB – Steam turbine model

Block Symbol

Licence: [MODEL](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uMS	Main valve requested position [0..1]	Double (F64)
uRS	Reheater valve requested position [0..1]	Double (F64)
f	Mains frequency [Hz]	Double (F64)
Mload	Shaft torque [Nm]	Double (F64)
MODE	1=mains mode, 0=island mode	Bool
R1	Model reset	Bool

Parameter

Kms	Main valve time constant [1/s]	⊙1.0	Double (F64)
Khp	High-pressure turbine part time constant [1/s]	⊙1.0	Double (F64)
Krs	Reheater valve time constant [1/s]	⊙1.0	Double (F64)
Krh	Reheater time constant [1/s]	⊙1.0	Double (F64)
Kip	Intermediate-pressure turbine part time constant [1/s]	⊙1.0	Double (F64)
Klp	Low-pressure turbine part time constant [1/s]	⊙1.0	Double (F64)
p	Input steam pressure [bar]	⊙1.0	Double (F64)
J	Mass inertia [kg.m ²]	⊙1.0	Double (F64)
dhhp	High-pressure turbine part enthalpy difference [J/kg]	⊙1.0	Double (F64)
dhip	Intermediate-pressure turbine part enthalpy difference [J/kg]	⊙1.0	Double (F64)
dhlp	Low-pressure turbine part enthalpy difference [J/kg]	⊙1.0	Double (F64)
B	Friction constant [Nm.s/rad]	⊙1.0	Double (F64)
Mfmax	Maximum shaft torque [Nm]	⊙1.0	Double (F64)
K1	Load model constant [Nm]	⊙1.0	Double (F64)

K2	Load model constant [Nm.s]	⊙1.0	Double (F64)
Pmin	Limit (minimal) power [W]	⊙1.0	Double (F64)
hMS_0	Main valve position initial value [0..1]	⊙1.0	Double (F64)
xHP_0	High-pressure part steam flow initial value [kg/s]	⊙1.0	Double (F64)
hRS_0	Reheater valve position initial value [0..1]	⊙1.0	Double (F64)
xRH_0	Reheater steam flow initial value [kg/s]	⊙1.0	Double (F64)
xIP_0	Intermediate-pressure part steam flow initial value [kg/s]	⊙1.0	Double (F64)
xLP_0	Low-pressure part steam flow initial value [kg/s]	⊙1.0	Double (F64)
e_0	Slip angle initial value [rad]	⊙1.0	Double (F64)
n_0	Shaft rotation frequency initial value [Hz]	⊙1.0	Double (F64)

Output

Pgen	Output shaft power [W]	Double (F64)
xLP	Output steam flow [kg/s]	Double (F64)
rpm	Rotation per minute	Double (F64)
E	Error indicator	Bool

Chapter 14

MATRIX – Blocks for matrix and vector operations

Contents

CNA – Array (vector/matrix) constant	444
MB_DASUM – Sum of the absolute values	446
MB_DAXPY – Performs $y := a*x + y$ for vectors x, y	448
MB_DCOPY – Copies vector x to vector y	450
MB_DDOT – Dot product of two vectors	452
MB_DGEMM – Performs $C := \alpha*op(A)*op(B) + \beta*C$, where $op(X) = X$ or $op(X) = X^T$	454
MB_DGEMV – Performs $y := \alpha*A*x + \beta*y$ or $y := \alpha*A^T*x + \beta*y$	456
MB_DGER – Performs $A := \alpha*x*y^T + A$	458
MB_DNRM2 – Euclidean norm of a vector	460
MB_DROT – Plain rotation of a vector	462
MB_DSCAL – Scales a vector by a constant	464
MB_DSWAP – Interchanges two vectors	466
MB_DTRMM – Performs $B := \alpha*op(A)*B$ or $B := \alpha*B*op(A)$, where $op(X) = X$ or $op(X) = X^T$ for triangular matrix A	468
MB_DTRMV – Performs $x := A*x$ or $x := A^T*x$ for triangular matrix A	470
MB_DTRSV – Solves one of the system of equations $A*x = b$ or $A^T*x = b$ for triangular matrix A	472
ML_DGEBAK – Backward transformation to ML_DGEBAL of left or right eigenvectors	474
ML_DGEBAL – Balancing of a general real matrix	476
ML_DGEBRD – Reduces a general real matrix to bidiagonal form by an orthogonal transformation	478

ML_DGECON – Estimates the reciprocal of the condition number of a general real matrix	480
ML_DGEES – Computes the eigenvalues, the Schur form, and, optionally, the matrix of Schur vectors	483
ML_DGEEV – Computes the eigenvalues and, optionally, the left and/or right eigenvectors	485
ML_DGEHRD – Reduces a real general matrix A to upper Hessenberg form	487
ML_DGELQF – Computes an LQ factorization of a real M-by-N matrix A	489
ML_DGELSD – Computes the minimum-norm solution to a real linear least squares problem	491
ML_DGEQRF – Computes an QR factorization of a real M-by-N matrix A	493
ML_DGESDD – Computes the singular value decomposition (SVD) of a real M-by-N matrix A	495
ML_DLACPY – Copies all or part of one matrix to another matrix	497
ML_DLANGE – Computes one of the matrix norms of a general matrix	499
ML_DLASET – Initializes the off-diagonal elements and the diagonal elements of a matrix to given values	501
ML_DTRSYL – Solves the real Sylvester matrix equation for quasi-triangular matrices A and B	503
MX_AT – Get Matrix/Vector element	505
MX_ATSET – Set Matrix/Vector element	506
MX_CNADD – Add scalar to each Matrix/Vector element	507
MX_CNMUL – Multiply a Matrix/Vector by a scalar	508
MX_CTODPA – Discretizes continuous model given by (A,B) to (Ad,Bd) using Pade approximations	509
MX_DIM – Matrix/Vector dimensions	511
MX_DIMSET – Set Matrix/Vector dimensions	512
MX_DSAGET – Set subarray of A into B	514
MX_DSAREF – Set reference to subarray of A into B	516
MX_DSASET – Set A into subarray of B	518
MX_DTRNSP – General matrix transposition: $B := \alpha A^T$	520
MX_DTRNSQ – Square matrix in-place transposition: $A := \alpha A^T$	522
MX_FILL – Fill real matrix or vector	524
MX_MAT – Matrix data storage block	525
MX_RAND – Randomly generated matrix or vector	526
MX_REFCOPY – Copies input references of matrices A and B to their output references	528
MX_SLFS – Save or load a Matrix/Vector into file or string	529
MX_VEC – Vector data storage block	532

MX_WRITE – Write a Matrix/Vector to the console/system log . .	533
RTOV – Vector multiplexer	535
SWVMR – Vector/matrix/reference signal switch	537
VTOR – Vector demultiplexer	538

The MATRIX library is designed for advanced matrix computations and manipulations. It encompasses a wide range of blocks such as **MB_DGEMM**, **MB_DTRMM**, and **MB_DGER** for matrix-matrix and matrix-vector operations. The library includes functions for matrix decomposition (**ML_DGEBRD**, **ML_DGEQRF**), eigenvalue problems (**ML_DGEEV**, **ML_DGEES**), and singular value decomposition (**ML_DGESDD**). Additionally, it offers utility blocks like **MX_MAT**, **MX_VEC**, and **MX_FILL** for matrix creation and manipulation, as well as specialized blocks such as **MX_DTRNSP** for matrix transposition and **MX_RAND** for generating random matrices. This library is essential for complex mathematical operations involving matrices in various applications.

Implementation notices

First element of a matrix has index (0,0), first element of a vector has index (0).

The vector is one-column-matrix, not separate object. One-row-matrix is called a row vector, but that object should not be used as vector in REXYGEN.

The matrix inputs and outputs are references. It means one block (the **MX_MAT** block or the **MX_VEC** block most often) reserve memory for the matrix and other block (using same reference) write/read same space. The **MB_DCOPY** block (and second the **MX_MAT** block) must be used to create copy of the matrix.

Some blocks using vector (**MB_DCPY**, **RTOV**, **VTOR**) not check exact dimensions (for example a 10x10 matrix is regard as 100-elements vector). Matrix is linearize into vector column by column, because a matrix is stored this way in memory (e.g. for a 10x10 matrix: element (1,0) has index 1 in vector, element (2,0) has index 2 in vector, element (0,1) has index 10 in vector, element (0,2) has index 11 in vector, etc.). These type of blocks could not be used with submatrix returned by the **MX_DSAREF** block. Behavior is undefined in this case.

The most matrix blocks has input and output matrix reference. Both are equal, but connecting input reference to output reference of previous block define execution order (the blocks are executed according signal flow in REXYGEN) and therefore computed matrix equation.

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.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Parameter

filename	CSV data file	String
TRN	Transpose loaded matrix	Bool
nmax	Allocated size of output matrix (total number of items)	Long (I32)
	↓2 ↑100000000 ⊙100	
etype	Type of elements	⊙8 Long (I32)
	1 Bool	
	2 Byte (U8)	
	3 Short (I16)	
	4 Long (I32)	
	5 Word (U16)	
	6 DWord (U32)	
	7 Float (F32)	
	8 Double (F64)	
	10 Large (I64)	
acn	Initial array value	⊙[0 1 2 3] Double (F64)

Output

`vec`

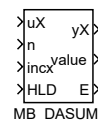
Reference to vector/matrix data

Reference

MB_DASUM – Sum of the absolute values

Block Symbol

Licence: [STANDARD](#)



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:

```
value = DASUM(N, uX, 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,
- $(N - 1) * INCX + 1 > CNT$.

See BLAS documentation [\[9\]](#) for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uX	Input reference to vector x	Reference
n	Number of processed vector elements	Long (I32)
incx	Index increment of vector x	Long (I32)
HLD	Hold	Bool

Output

yX	Output reference to vector x	Reference
-----------	------------------------------	------------------

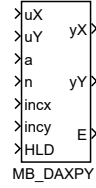
value Return value of the function
E Error indicator

Double (F64)
Bool

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:

```
DAXPY(N, a, 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 $CNTY$ referenced by uY .
- 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),
- $n < 0$,
- $(N - 1) * |INCX| + 1 > CNTX$, where $CNTX$ is a number of the vector or matrix elements referenced by uX ,
- $(N - 1) * |INCY| + 1 > CNTY$.

See BLAS documentation [\[9\]](#) for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uX	Input reference to vector x	Reference
------	-------------------------------	-----------

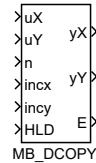
<code>uY</code>	Input reference to vector y	Reference
<code>a</code>	Scalar coefficient a	Double (F64)
<code>n</code>	Number of processed vector elements	Long (I32)
<code>incx</code>	Index increment of vector x	Long (I32)
<code>incy</code>	Index increment of vector y	Long (I32)
<code>HLD</code>	Hold	Bool

Output

<code>yX</code>	Output reference to vector x	Reference
<code>yY</code>	Output reference to vector y	Reference
<code>E</code>	Error indicator	Bool

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:

```
DCOPY(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** ≠ 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,
- $(N - 1) * |INCX| + 1 > CNTX$,
- $(N - 1) * |INCY| + 1 > CNTY$, where **CNTY** is a number of the vector or matrix elements referenced by **uY**.

See BLAS documentation [\[9\]](#) for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uX	Input reference to vector x	Reference
uY	Input reference to vector y	Reference

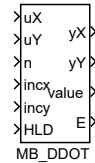
n	Number of processed vector elements	Long (I32)
incx	Index increment of vector x	Long (I32)
incy	Index increment of vector y	Long (I32)
HLD	Hold	Bool

Output

yX	Output reference to vector x	Reference
yY	Output reference to vector y	Reference
E	Error indicator	Bool

MB_DDOT – Dot product of two vectors

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 `DDOT` is called internally:

```
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 $CNTX$ 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 .

See BLAS documentation [\[9\]](#) for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uX	Input reference to vector x	Reference
uY	Input reference to vector y	Reference

<code>n</code>	Number of processed vector elements	Long (I32)
<code>incx</code>	Index increment of vector x	Long (I32)
<code>incy</code>	Index increment of vector y	Long (I32)
<code>HLD</code>	Hold	Bool

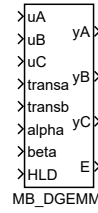
Output

<code>yX</code>	Output reference to vector x	Reference
<code>yY</code>	Output reference to vector y	Reference
<code>value</code>	Return value of the function	Double (F64)
<code>E</code>	Error indicator	Bool

MB_DGEMM – Performs $C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$,
 where $\text{op}(X) = X$ or $\text{op}(X) = X^T$

Block Symbol

Licence: [STANDARD](#)



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 BLAS function **DGEMM** is called internally:

```
DGEMM(sTRANSa, sTRANSb, M, N, KA, alpha, uA, LDA, uB, LDB, beta, uC, LDC);
```

where parameters of **DGEMM** are set in the following way:

- Integer inputs **transa** and **transb** are mapped to strings **sTRANSa** and **sTRANSb**:
 $\{0, 1\} \rightarrow \text{"N"}, \{2\} \rightarrow \text{"T"} \text{ and } \{3\} \rightarrow \text{"C"}$.
- **M** is number of rows of the matrix referenced by **uC**.
- **N** is number of columns of the matrix referenced by **uC**.
- If the input **transa** is equal to 0 or 1 then **KA** is number of columns else **KA** is number rows of the matrix referenced by **uA**.
- **LDA**, **LDB** and **LDC** are leading dimensions of matrices referenced by **uA**, **uB** and **uC**.

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),
- **transa** or **transb** is less than 0 or greater than 3,
- $KA \neq KB$; if the input **transb** is equal to 0 or 1 then **KB** is number of rows else **KB** is number of columns of the matrix referenced by **uB** (i.e. matrices $\text{op}(A)$ and $\text{op}(B)$ have to be multipliable).
- the call of the function **DGEMM** returns error using the function **XERBLA**, see the system log.

See BLAS documentation [\[9\]](#) for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uA	Input reference to matrix A		Reference
uB	Input reference to matrix B		Reference
uC	Input reference to matrix C		Reference
transa	Transposition of matrix A	↓0 ↑3	Long (I32)
transb	Transposition of matrix B	↓0 ↑3	Long (I32)
alpha	Scalar coefficient alpha		Double (F64)
beta	Scalar coefficient beta		Double (F64)
HLD	Hold		Bool

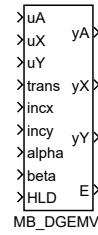
Output

yA	Output reference to matrix A	Reference
yB	Output reference to matrix B	Reference
yC	Output reference to matrix C	Reference
E	Error indicator	Bool

MB_DGEMV – Performs $y := \text{alpha} * A * x + \text{beta} * y$ or $y := \text{alpha} * A^T * x + \text{beta} * y$

Block Symbol

Licence: [STANDARD](#)



Function Description

The output references `yA`, `yX` and `yY` are always set to the corresponding input references `uA`, `uX` and `uY`. If `HLD = on` then nothing is computed otherwise the BLAS function `DGEMV` is called internally:

```
DGEMV(sTRANS, M, N, alpha, uA, LDA, uX, INCX, beta, uY, INCY);
```

where parameters of `DGEMV` are set in the following way:

- Integer input `trans` is mapped to the string `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 `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 `uA` or `uX` or `uY` is not defined (i.e. input `uA` or `uX` or `uY` is not connected),
- `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.

See BLAS documentation [\[9\]](#) for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uA	Input reference to matrix A		Reference
uX	Input reference to vector x		Reference
uY	Input reference to vector y		Reference
trans	Transposition of the input matrix	↓0 ↑3	Long (I32)
incx	Index increment of vector x		Long (I32)
incy	Index increment of vector y		Long (I32)
alpha	Scalar coefficient alpha		Double (F64)
beta	Scalar coefficient beta		Double (F64)
HLD	Hold		Bool

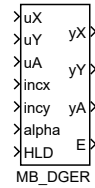
Output

yA	Output reference to matrix A		Reference
yX	Output reference to vector x		Reference
yY	Output reference to vector y		Reference
E	Error indicator		Bool

MB_DGER – Performs $A := \alpha * x * y^T + A$

Block Symbol

Licence: [STANDARD](#)



Function Description

The output references **yX**, **yY** and **yA** are always set to the corresponding input references **uX**, **uY** and **uA**. If **HLD = on** then nothing is computed otherwise the BLAS function **DGER** is called internally:

```
DGER(M, N, alpha, uX, INCX, uY, 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 **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.
- **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.

See BLAS documentation [\[9\]](#) for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uX	Input reference to vector x	Reference
uY	Input reference to vector y	Reference

uA	Input reference to matrix A	Reference
incx	Index increment of vector x	Long (I32)
incy	Index increment of vector y	Long (I32)
alpha	Scalar coefficient alpha	Double (F64)
HLD	Hold	Bool

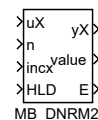
Output

yX	Output reference to vector x	Reference
yY	Output reference to vector y	Reference
yA	Output reference to matrix A	Reference
E	Error indicator	Bool

MB_DNRM2 – Euclidean norm of a vector

Block Symbol

Licence: [STANDARD](#)



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:

```
value = DNRM2(N, uX, 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`,
- $(N - 1) * |INCX| + 1 > CNT$.

See BLAS documentation [\[9\]](#) for more details.

Use the block [ML_DLANGE](#) for computation of various norms of a matrix.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

		Reference
<code>uX</code>	Input reference to vector x	Long (I32)
<code>n</code>	Number of processed vector elements	Long (I32)
<code>incx</code>	Index increment of vector x	Long (I32)
<code>HLD</code>	Hold	Bool

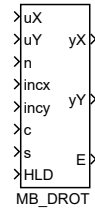
Output

<code>yX</code>	Output reference to vector <code>x</code>
<code>value</code>	Return value of the function
<code>E</code>	Error indicator

<code>Reference</code>
<code>Double (F64)</code>
<code>Bool</code>

MB_DROT – Plain rotation of a vector

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 **DROT** is called internally:

```
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 **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,
- $(N - 1) * |INCX| + 1 > CNTX$,
- $(N - 1) * |INCY| + 1 > CNTY$, where **CNTY** is a number of the vector or matrix elements referenced by **uY**.

See BLAS documentation [9] for more details.

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

uX	Input reference to vector x	Reference
-----------	-----------------------------	------------------

uY	Input reference to vector y	Reference
n	Number of processed vector elements	Long (I32)
incx	Index increment of vector x	Long (I32)
incy	Index increment of vector y	Long (I32)
c	Scalar coefficient c	Double (F64)
s	Scalar coefficient s	Double (F64)
HLD	Hold	Bool

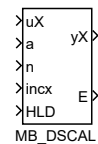
Output

yX	Output reference to vector x	Reference
yY	Output reference to vector y	Reference
E	Error indicator	Bool

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`,
- $(N - 1) * INCX + 1 > CNT$.

See BLAS documentation [\[9\]](#) for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

		Reference
<code>uX</code>	Input reference to vector x	Double (F64)
<code>a</code>	Scalar coefficient a	
<code>n</code>	Number of processed vector elements	Long (I32)
<code>incx</code>	Index increment of vector x	Long (I32)
<code>HLD</code>	Hold	Bool

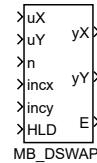
Output

yX Output reference to vector x
E Error indicator

Reference
Bool

MB_DSWAP – Interchanges two vectors

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 **DSWAP** is called internally:

```
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** ≠ 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,
- $(N - 1) * |INCX| + 1 > CNTX$,
- $(N - 1) * |INCY| + 1 > CNTY$, where **CNTY** is a number of the vector or matrix elements referenced by **uY**.

See BLAS documentation [\[9\]](#) for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uX	Input reference to vector x	Reference
uY	Input reference to vector y	Reference

n	Number of processed vector elements	Long (I32)
incx	Index increment of vector x	Long (I32)
incy	Index increment of vector y	Long (I32)
HLD	Hold	Bool

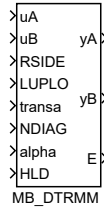
Output

yX	Output reference to vector x	Reference
yY	Output reference to vector y	Reference
E	Error indicator	Bool

MB_DTRMM – Performs $B := \alpha * \text{op}(A) * B$ or $B := \alpha * B * \text{op}(A)$, where $\text{op}(X) = X$ or $\text{op}(X) = X^T$ for triangular matrix A

Block Symbol

Licence: [STANDARD](#)



Function Description

The output references yA and yB are always set to the corresponding input references uA and uB . If $HLD = \text{on}$ then nothing is computed otherwise the BLAS function **DTRMM** is called internally:

DTRMM(*sRSIDE*, *sLUPLO*, *sTRANSa*, *sNDIAG*, *M*, *N*, *alpha*, *uA*, *LDA*, *uB*, *LDB*);

where parameters of **DTRMM** are set in the following way:

- If $RSIDE = \text{on}$ then the string *sRSIDE* is set to "R" else it is set to "L".
- If $LUPLO = \text{on}$ then the string *sLUPLO* is set to "L" else it is set to "U".
- Integer input *transa* is mapped to the string *sTRANSa*: $\{0, 1\} \rightarrow \text{"N"}, \{2\} \rightarrow \text{"T"}$ and $\{3\} \rightarrow \text{"C"}$.
- If $NDIAG = \text{on}$ then the string *sNDIAG* is set to "N" else it is set to "U".
- *M* is number of rows of the matrix referenced by *uB*.
- *N* is number of columns of the matrix referenced by *uB*.
- *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),
- *transa* is less than 0 or greater than 3,
- matrix referenced by *uA* is not square or is not compatible with the matrix referenced by *uB*,
- the call of the function **DTRMM** returns error using the function **XERBLA**, see the system log.

See BLAS documentation [\[9\]](#) for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uA	Input reference to matrix A		Reference
uB	Input reference to matrix B		Reference
RSIDE	Operation is applied from right side		Bool
LUPL0	Matrix A is a lower triangular matrix		Bool
transa	Transposition of matrix A	↓0 ↑3	Long (I32)
NDIAG	Matrix A is not assumed to be unit triangular		Bool
alpha	Scalar coefficient alpha		Double (F64)
HLD	Hold		Bool

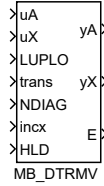
Output

yA	Output reference to matrix A	Reference
yB	Output reference to matrix B	Reference
E	Error indicator	Bool

MB_DTRMV – Performs $x := A * x$ or $x := A^T * x$ for triangular matrix A

Block Symbol

Licence: [STANDARD](#)



Function Description

The output references `yA` and `yX` are always set to the corresponding input references `uA` and `uX`. If `HLD = on` then nothing is computed otherwise the BLAS function `DTRMV` is called internally:

```
DTRMV(sLUPLO, sTRANS, sNDIAG, N, uA, LDA, uX, INCX);
```

where parameters of `DTRMV` are set in the following way:

- If `LUPLO = on` then the string `sLUPLO` is set to "L" else it is set to "U".
- Integer input `trans` is mapped to the string `sTRANS`: $\{0, 1\} \rightarrow \text{"N"}, \{2\} \rightarrow \text{"T"}$ and $\{3\} \rightarrow \text{"C"}$.
- If `NDIAG = on` then the string `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 `incx` $\neq 0$ then `INCX` is set to `incx` else `INCX` is set to 1.

The error flag `E` is set to `on` if:

- the reference `uA` or `uX` is not defined (i.e. input `uA` or `uX` is not connected),
- `trans` is less than 0 or greater than 3,
- matrix referenced by `uA` is not square,
- $(N - 1) * |INCX| + 1 > CNTX$, where `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.

See BLAS documentation [\[9\]](#) for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>uA</code>	Input reference to matrix A		Reference
<code>uX</code>	Input reference to vector x		Reference
<code>LUPL0</code>	Matrix A is a lower triangular matrix		Bool
<code>trans</code>	Transposition of the input matrix	↓0 ↑3	Long (I32)
<code>NDIAG</code>	Matrix A is not assumed to be unit triangular		Bool
<code>incx</code>	Index increment of vector x		Long (I32)
<code>HLD</code>	Hold		Bool

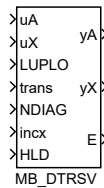
Output

<code>yA</code>	Output reference to matrix A	Reference
<code>yX</code>	Output reference to vector x	Reference
<code>E</code>	Error indicator	Bool

MB_DTRSV – Solves one of the system of equations $A * x = b$ or $A^T * x = b$ for triangular matrix A

Block Symbol

Licence: [STANDARD](#)



Function Description

The output references **yA** and **yX** are always set to the corresponding input references **uA** and **uX**. If **HLD = on** then nothing is computed otherwise the BLAS function **DTRSV** is called internally:

```
DTRSV(sLUPL0, sTRANS, sNDIAG, N, uA, LDA, uX, INCX);
```

where parameters of **DTRSV** are set in the following way:

- If **LUPL0 = on**, then the string **sLUPL0** is set to "L" else it is set to "U".
- Integer input **trans** is mapped to the string **sTRANS**: $\{0, 1\} \rightarrow \text{"N"}, \{2\} \rightarrow \text{"T"}$ and $\{3\} \rightarrow \text{"C"}$.
- If **NDIAG = on** then the string **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 **incx** $\neq 0$ then **INCX** is set to **incx** else **INCX** is set to 1.

The error flag **E** is set to **on** if:

- the reference **uA** or **uX** is not defined (i.e. input **uA** or **uX** is not connected),
- **trans** is less than 0 or greater than 3,
- matrix referenced by **uA** is not square,
- $(N - 1) * |\text{INCX}| + 1 > \text{CNTX}$, where **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.

See BLAS documentation [\[9\]](#) for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>uA</code>	Input reference to matrix A		Reference
<code>uX</code>	Input reference to vector x		Reference
<code>LUPL0</code>	Matrix A is a lower triangular matrix		Bool
<code>trans</code>	Transposition of the input matrix	↓0 ↑3	Long (I32)
<code>NDIAG</code>	Matrix A is not assumed to be unit triangular		Bool
<code>incx</code>	Index increment of vector x		Long (I32)
<code>HLD</code>	Hold		Bool

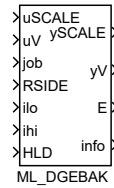
Output

<code>yA</code>	Output reference to matrix A	Reference
<code>yX</code>	Output reference to vector x	Reference
<code>E</code>	Error indicator	Bool

ML_DGEBAK – Backward transformation to ML_DGEBAL of left or right eigenvectors

Block Symbol

Licence: [MATRIX](#)



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:

```
DGEBAK(sJOB, sRSIDE, N, ilo, IHI, uSCALE, M, uV, LDV, info);
```

where parameters of `DGEBAK` are set in the following way:

- Integer input `job` is mapped to the string `sJOB`: $\{0, 1\} \rightarrow \text{"N"}, \{2\} \rightarrow \text{"P"}, \{3\} \rightarrow \text{"S"}$ and $\{4\} \rightarrow \text{"B"}$.
- 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` $\neq 0$ 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 [10] for more details.

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

uSCALE	Input reference to vector SCALE	Reference
uV	Reference to matrix of right or left eigenvectors to be transformed	Reference
job	Type of backward transformation required	↓0 ↑4 Long (I32)
RSIDE	Operation is applied from right side	Bool
ilo	Zero based low row and column index of working submatrix	Long (I32)
ihi	Zero based high row and column index of working submatrix	Long (I32)
HLD	Hold	Bool

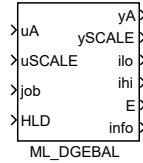
Output

ySCALE	Output reference to vector SCALE	Reference
yV	Reference to matrix of transformed right or left eigenvectors	Reference
E	Error indicator	Bool
info	LAPACK function result info. If info = -i, the i=th argument had an illegal value	Long (I32)

ML_DGEBAL – Balancing of a general real matrix

Block Symbol

Licence: [MATRIX](#)



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:

```
DGEBAL(sJOB, N, uA, LDA, ilo, ihi, uSCALE, info);
```

where parameters of `DGEBAL` are set in the following way:

- Integer input `job` is mapped to the string `sJOB`: $\{0, 1\} \rightarrow \text{"N"}, \{2\} \rightarrow \text{"P"}, \{3\} \rightarrow \text{"S"}$ and $\{4\} \rightarrow \text{"B"}$.
- `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 [10] for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uA	Input reference to matrix A		Reference
uSCALE	Input reference to vector SCALE		Reference
job	Specifies the operations to be performed on matrix A	↓0 ↑4	Long (I32)
HLD	Hold		Bool

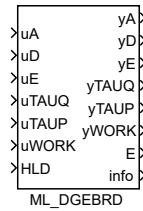
Output

yA	Output reference to matrix A		Reference
ySCALE	Output reference to vector SCALE		Reference
ilo	Zero based low row and column index of working submatrix		Long (I32)
ihi	Zero based high row and column index of working submatrix		Long (I32)
E	Error indicator		Bool
info	LAPACK function result info. If info = -i, the i=th argument had an illegal value		Long (I32)

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.

See LAPACK documentation [10] for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uA	Input reference to matrix A	Reference
uD	Diagonal elements of the bidiagonal matrix B	Reference
uE	Off-diagonal elements of the bidiagonal matrix B	Reference
uTAUQ	Reference to a vector of scalar factors of the elementary reflectors which represent the orthogonal matrix Q	Reference
uTAUP	Reference to a vector of scalar factors of the elementary reflectors which represent the orthogonal matrix P	Reference
uWORK	Input reference to working vector WORK	Reference
HLD	Hold	Bool

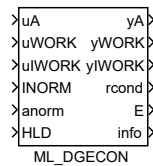
Output

yA	Output reference to matrix A	Reference
yD	Output reference to D	Reference
yE	Output reference to E	Reference
yTAUQ	Output reference to TAUQ	Reference
yTAUP	Output reference to TAUP	Reference
yWORK	Output reference to working vector WORK	Reference
E	Error indicator	Bool
info	LAPACK function result info. If info = -i, the i=th argument had an illegal value	Long (I32)

ML_DGECON – Estimates the reciprocal of the condition number of a general real matrix

Block Symbol

Licence: [MATRIX](#)



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 `DGECON` is called internally:

```
DGECON(sINORM, N, uA, LDA, anorm, rcond, uWORK, uIWORK, info);
```

where parameters of `DGECON` 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 `DGECON`.

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 `DGECON` returns error using the function `XERBLA`, see the return code `info` and system log.

See LAPACK documentation [\[10\]](#) for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>uA</code>	Input reference to matrix A	Reference
<code>uWORK</code>	Input reference to working vector WORK	Reference
<code>uIWORK</code>	Input reference to integer working vector WORK	Reference
<code>INORM</code>	Use Infinity-norm	Bool
<code>anorm</code>	Norm of the original matrix A	Double (F64)
<code>HLD</code>	Hold	Bool

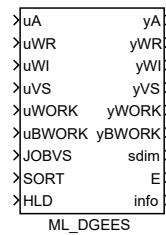
Output

<code>yA</code>	Output reference to matrix A	Reference
<code>yWORK</code>	Output reference to working vector WORK	Reference
<code>yIWORK</code>	Output reference to integer working vector WORK	Reference
<code>rcond</code>	The reciprocal of the condition number of the matrix A	Double (F64)
<code>E</code>	Error indicator	Bool
<code>info</code>	LAPACK function result info. If <code>info = -i</code> , the <code>i</code> =th argument had an illegal value	Long (I32)

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:

```
DGEES(sJOBVS, sSORT, SELECT, N, uA, LDA, sdim, uWR, uWI, uVS, LDVS, uWORK,
      LWORK, uBWORK, 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 [10] for more details.

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

<code>uA</code>	Input reference to matrix <code>A</code>	Reference
<code>uWR</code>	Input reference to vector of real parts of eigenvalues	Reference
<code>uWI</code>	Input reference to vector of imaginary parts of eigenvalues	Reference
<code>uVS</code>	Input reference to orthogonal matrix of Schur vectors	Reference
<code>uWORK</code>	Input reference to working vector <code>WORK</code>	Reference
<code>uBWORK</code>	Input reference to Boolean working vector <code>WORK</code>	Reference
<code>JOBVS</code>	If true then Schur vectors are computed	Bool
<code>SORT</code>	If true then eigenvalues are sorted	Bool
<code>HLD</code>	Hold	Bool

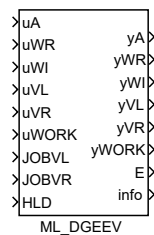
Output

<code>yA</code>	Output reference to matrix <code>A</code>	Reference
<code>yWR</code>	Output reference to vector of real parts of eigenvalues	Reference
<code>yWI</code>	Output reference to vector of imaginary parts of eigenvalues	Reference
<code>yVS</code>	Output reference to <code>VS</code>	Reference
<code>yWORK</code>	Output reference to working vector <code>WORK</code>	Reference
<code>yBWORK</code>	Output reference to Boolean working vector <code>WORK</code>	Reference
<code>sdim</code>	If <code>SORT</code> then number of eigenvalues for which <code>SELECT</code> is true else 0	Long (I32)
<code>E</code>	Error indicator	Bool
<code>info</code>	LAPACK function result <code>info</code> . If <code>info = -i</code> , the <code>i</code> =th argument had an illegal value	Long (I32)

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 [10] for more details.

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

<code>uA</code>	Input reference to matrix <code>A</code>	Reference
<code>uWR</code>	Input reference to vector of real parts of eigenvalues	Reference
<code>uWI</code>	Input reference to vector of imaginary parts of eigenvalues	Reference
<code>uVL</code>	Input reference to matrix of left eigenvectors	Reference
<code>uVR</code>	Input reference to matrix of right eigenvectors	Reference
<code>uWORK</code>	Input reference to working vector <code>WORK</code>	Reference
<code>JOBVL</code>	If true then left eigenvectors are computed	Bool
<code>JOBVR</code>	If true then right eigenvectors are computed	Bool
<code>HLD</code>	Hold	Bool

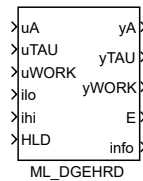
Output

<code>yA</code>	Output reference to matrix <code>A</code>	Reference
<code>yWR</code>	Output reference to vector of real parts of eigenvalues	Reference
<code>yWI</code>	Output reference to vector of imaginary parts of eigenvalues	Reference
<code>yVL</code>	Output reference to <code>VL</code>	Reference
<code>yVR</code>	Output reference to <code>VR</code>	Reference
<code>yWORK</code>	Output reference to working vector <code>WORK</code>	Reference
<code>E</code>	Error indicator	Bool
<code>info</code>	LAPACK function result info. If <code>info = -i</code> , the <code>i</code> -th argument had an illegal value	Long (I32)

ML_DGEHRD – Reduces a real general matrix A to upper Hessenberg form

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 `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` $\neq 0$ 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 [10] for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>uA</code>	Input reference to matrix A	Reference
<code>uTAU</code>	Input reference to vector of scalar factors of the elementary reflectors	Reference
<code>uWORK</code>	Input reference to working vector WORK	Reference
<code>ilo</code>	Zero based low row and column index of working submatrix	Long (I32)
<code>ihi</code>	Zero based high row and column index of working submatrix	Long (I32)
<code>HLD</code>	Hold	Bool

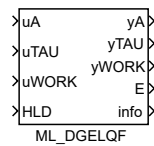
Output

<code>yA</code>	Output reference to matrix A	Reference
<code>yTAU</code>	Output reference to vector of scalar factors of the elementary reflectors	Reference
<code>yWORK</code>	Output reference to working vector WORK	Reference
<code>E</code>	Error indicator	Bool
<code>info</code>	LAPACK function result info. If info = -i, the i=th argument had an illegal value	Long (I32)

ML_DGELQF – Computes an LQ 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 **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.

See LAPACK documentation [10] for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uA	Input reference to matrix A	Reference
uTAU	Input reference to vector of scalar factors of the elementary reflectors	Reference
uWORK	Input reference to working vector WORK	Reference
HLD	Hold	Bool

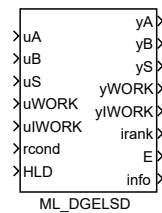
Output

yA	Output reference to matrix A	Reference
yTAU	Output reference to vector of scalar factors of the elementary reflectors	Reference
yWORK	Output reference to working vector WORK	Reference
E	Error indicator	Bool
info	LAPACK function result info. If info = -i, the i=th argument had an illegal value	Long (I32)

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:

```
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 `uIWORK` 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 [10] for more details.

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

<code>uA</code>	Input reference to matrix A	Reference
<code>uB</code>	Input reference to matrix B	Reference
<code>uS</code>	Input reference to vector of singular values	Reference
<code>uWORK</code>	Input reference to working vector WORK	Reference
<code>uIWORK</code>	Input reference to integer working vector WORK	Reference
<code>rcond</code>	Used to determine the effective rank of A	Double (F64)
<code>HLD</code>	Hold	Bool

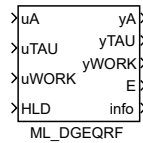
Output

<code>yA</code>	Output reference to matrix A	Reference
<code>yB</code>	Output reference to matrix B	Reference
<code>yS</code>	Output reference to vector of singular values	Reference
<code>yWORK</code>	Output reference to working vector WORK	Reference
<code>yIWORK</code>	Output reference to integer working vector WORK	Reference
<code>irank</code>	Effective rank of A	Long (I32)
<code>E</code>	Error indicator	Bool
<code>info</code>	LAPACK function result info. If <code>info = -i</code> , the <code>i</code> -th argument had an illegal value	Long (I32)

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:

```
DGEQRF(M, N, uA, LDA, uTAU, uWORK, LWORK, 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.

See LAPACK documentation [10] for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>uA</code>	Input reference to matrix A	Reference
<code>uTAU</code>	Input reference to vector of scalar factors of the elementary reflectors	Reference
<code>uWORK</code>	Input reference to working vector WORK	Reference
<code>HLD</code>	Hold	Bool

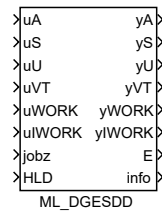
Output

<code>yA</code>	Output reference to matrix A	Reference
<code>yTAU</code>	Output reference to vector of scalar factors of the elementary reflectors	Reference
<code>yWORK</code>	Output reference to working vector WORK	Reference
<code>E</code>	Error indicator	Bool
<code>info</code>	LAPACK function result info. If <code>info = -i</code> , the <code>i</code> =th argument had an illegal value	Long (I32)

ML_DGESDD – Computes the singular value decomposition (SVD) of a real M-by-N matrix A

Block Symbol

Licence: [MATRIX](#)



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`: $\{0,1\} \rightarrow \text{"A"} , \{2\} \rightarrow \text{"S"} , \{3\} \rightarrow \text{"O"} \text{ and } \{4\} \rightarrow \text{"N"} .$
- `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 `MINMN`, 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*MINMN`,

- the call of the function `DGESDD` returns error using the function `XERBLA`, see the return code `info` and system log.

See LAPACK documentation [10] for more details.

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

<code>uA</code>	Input reference to matrix A	Reference
<code>uS</code>	Input reference to vector of singular values	Reference
<code>uU</code>	Input reference to matrix containing left singular vectors of A	Reference
<code>uVT</code>	Input reference to matrix containing right singular vectors of A	Reference
<code>uWORK</code>	Input reference to working vector WORK	Reference
<code>uIWORK</code>	Input reference to integer working vector WORK	Reference
<code>jobz</code>	Specifies options for computing	Long (I32)
<code>HLD</code>	Hold	Bool

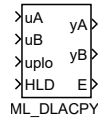
Output

<code>yA</code>	Output reference to matrix A	Reference
<code>yS</code>	Output reference to vector of singular values	Reference
<code>yU</code>	Output reference to matrix containing left singular vectors of A	Reference
<code>yVT</code>	Output reference to matrix containing right singular vectors of A	Reference
<code>yWORK</code>	Output reference to working vector WORK	Reference
<code>yIWORK</code>	Output reference to integer working vector WORK	Reference
<code>E</code>	Error indicator	Bool
<code>info</code>	LAPACK function result info. If <code>info = -i</code> , the <code>i</code> -th argument had an illegal value	Long (I32)

ML_DLACPY – Copies all or part of one matrix to another matrix

Block Symbol

Licence: [STANDARD](#)



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:

```
DLACPY(sUPLO, M, N, uA, LDA, uB, LDA);
```

where parameters of **DLACPY** are set in the following way:

- Integer input **uplo** is mapped to the string **sUPLO**: $\{0, 1\} \rightarrow \text{"A"}, \{2\} \rightarrow \text{"U"}$ and $\{3\} \rightarrow \text{"L"}$.
- **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**.

See LAPACK documentation [\[10\]](#) for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uA	Input reference to matrix A	Reference
uB	Input reference to matrix B	Reference

uplo	Part of the matrix to be copied	Long (I32)
	0 All	
	1 All	
	2 Upper	
	3 Lower	
HLD	Hold	Bool

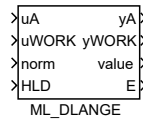
Output

yA	Output reference to matrix A	Reference
yB	Output reference to matrix B	Reference
E	Error indicator	Bool

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**: $\{0, 1\} \rightarrow \text{"F"}$ (Frobenius norm), $\{2\} \rightarrow \text{"M"}$ ($\max(\text{abs}(A(i, j)))$), $\{3\} \rightarrow \text{"1"}$ (one norm) and $\{4\} \rightarrow \text{"I"}$ (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 $\text{LWORK} \geq \text{M}$. **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).

See LAPACK documentation [\[10\]](#) for more details.

Use the block [MB_DNRM2](#) for computation of Frobenius norm of a vector.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uA	Input reference to matrix A		Reference
uWORK	Input reference to working vector WORK		Reference
norm	The selected matrix norm	↓0 ↑4	Long (I32)
HLD	Hold		Bool

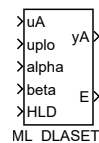
Output

yA	Output reference to matrix A		Reference
yWORK	Output reference to working vector WORK		Reference
value	Return value of the function		Double (F64)
E	Error indicator		Bool

ML_DLASET – Initializes the off-diagonal elements and the diagonal elements of a matrix to given values

Block Symbol

Licence: [STANDARD](#)



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(sUPL0, M, N, alpha, beta,uA, LDA);
```

where parameters of **DLASET** are set in the following way:

- Integer input **uplo** is mapped to the string **sUPL0**: $\{0, 1\} \rightarrow \text{"A"}, \{2\} \rightarrow \text{"U"}$ and $\{3\} \rightarrow \text{"L"}$.
- **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).

See LAPACK documentation [\[10\]](#) for more details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uA	Input reference to matrix A	Reference
uplo	Part of the matrix to be copied	Long (I32)
	0 All	
	1 All	
	2 Upper	
	3 Lower	
alpha	Scalar coefficient alpha	Double (F64)
beta	Scalar coefficient beta	Double (F64)
HLD	Hold	Bool

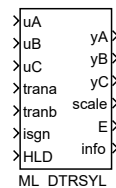
Output

yA	Output reference to matrix A	Reference
E	Error indicator	Bool

ML_DTRSYL – Solves the real Sylvester matrix equation for quasi-triangular matrices A and B

Block Symbol

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:

```
DTRSYL(sTRANA, sTRANB, M, N, uA, LDA, uB, LDB, uC, LDC, scale, info);
```

where parameters of **DTRSYL** are set in the following way:

- Integer inputs **trana** and **tranb** are mapped to strings **sTRANA** and **sTRANB**: $\{0, 1\} \rightarrow \text{"N"}, \{2\} \rightarrow \text{"T"}$ 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 **uB**.
- **LDA**, **LDB** and **LDC** are leading dimensions of matrices referenced by **uA**, **uB** and **uC**.
- **scale** is returned scaling factor to avoid overflow.
- **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),
- **trana** or **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 [10] for more details.

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

<code>uA</code>	Input reference to matrix A		Reference
<code>uB</code>	Input reference to matrix B		Reference
<code>uC</code>	Input reference to matrix C		Reference
<code>trana</code>	Transposition of matrix A	↓0 ↑3	Long (I32)
<code>tranb</code>	Transposition of matrix B	↓0 ↑3	Long (I32)
<code>isgn</code>	Sign in the equation (1 or -1)	↓-1 ↑1	Long (I32)
<code>HLD</code>	Hold		Bool

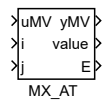
Output

<code>yA</code>	Output reference to matrix A	Reference
<code>yB</code>	Output reference to matrix B	Reference
<code>yC</code>	Output reference to matrix C	Reference
<code>scale</code>	Scale	Double (F64)
<code>E</code>	Error indicator	Bool
<code>info</code>	LAPACK function result info. If <code>info = -i</code> , the <code>i</code> =th argument had an illegal value	Long (I32)

MX_AT – Get Matrix/Vector element

Block Symbol

Licence: [STANDARD](#)



Function Description

The function block MX_AT returns the value (output **value**) of the matrix element at the **i**-th row and **j**-th column or 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 $i \geq m$, 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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

			Reference
uMV	Input reference to a matrix or vector		
i	Row index of the element	↓0	Long (I32)
j	Column index of the element	↓0	Long (I32)

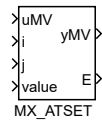
Output

			Reference
yMV	Output reference to a matrix or vector		
value	Value of element at position (i,j)		Long (I32)
E	Error indicator		Bool

MX_ATSET – Set Matrix/Vector element

Block Symbol

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 $i \geq m$, 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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

			Reference
uMV	Input reference to a matrix or vector		
i	Row index of the element	↓0	Long (I32)
j	Column index of the element	↓0	Long (I32)
value	Value which should be set to the element at position (<i>i</i> , <i>j</i>)		Long (I32)

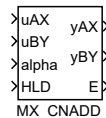
Output

			Reference
yMV	Output reference to a matrix or vector		
E	Error indicator		Bool

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** or **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**.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uAX	Input reference to the matrix A or vector X	Reference
uBY	Input reference to the matrix B or vector Y	Reference
alpha	Scalar coefficient alpha	Double (F64)
HLD	Hold	Bool

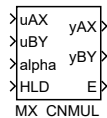
Output

yAX	Output reference to the matrix A or vector X	Reference
yBY	Output reference to the matrix B or vector Y	Reference
E	Error indicator	Bool

MX_CNMUL – Multiply a Matrix/Vector by a scalar

Block Symbol

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`.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>uAX</code>	Input reference to the matrix A or vector X	Reference
<code>uBY</code>	Input reference to the matrix B or vector Y	Reference
<code>alpha</code>	Scalar coefficient alpha	Double (F64)
<code>HLD</code>	Hold	Bool

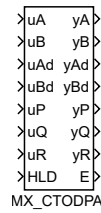
Output

<code>yAX</code>	Output reference to the matrix A or vector X	Reference
<code>yBY</code>	Output reference to the matrix B or vector Y	Reference
<code>E</code>	Error indicator	Bool

MX_CTODPA – Discretizes continuous model given by (A,B) to (Ad,Bd) using Pade approximations

Block Symbol

Licence: [STANDARD](#)



Function Description

Function block **MX_CTODPA** discretizes a continuous state space model using Padé approximations of matrix exponential and its integral and scaling technique ([7]). The used technique is similar to method 3 Scaling and squaring described in [11].

The output references **yA**, **yB**, **yAd**, **yBd**, **yP**, **yQ** and **yR** are always set to the corresponding input references **uA**, **uB**, **uAd**, **uBd**, **uP**, **uQ** and **uR**. If **HLD = on** then nothing is computed otherwise the function **mCtoD** is called internally:

```
mCtoD(nRes, uAd, uBd, uA, uB, N, M, is, Ts, eps, uP, uQ, uR);
```

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 **uA**.
- **M** is number of columns of the input matrix referenced by **uB**.
- **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 **uA** or **uB** or **uAd** or **uBd** or **uP** or **uQ** or **uR** is not defined (i.e. input **uA** or **uB** or **uAd** or **uBd** or **uP** or **uQ** or **uR** is not connected),
- number of columns of the matrix referenced by **uA** is not equal to **N**,
- number of rows of the matrix referenced by **uB** is not equal to **M**,
- number of elements of any matrix referenced by **uAd**, **uP**, **uQ** or **uR** is less than $N * N$,
- number of elements of the matrix referenced by **uBd** is less than $N * M$,

- the return code `nRes` of the function `mCtoD` is not equal to zero.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>uA</code>	Input reference to matrix A	Reference
<code>uB</code>	Input reference to matrix B	Reference
<code>uAd</code>	Input reference to discretized matrix A	Reference
<code>uBd</code>	Input reference to discretized matrix B	Reference
<code>uP</code>	Input reference to a helper matrix	Reference
<code>uQ</code>	Input reference to a helper matrix	Reference
<code>uR</code>	Input reference to a helper matrix	Reference
<code>HLD</code>	Hold	Bool

Parameter

<code>is</code>	Pade approximation order	↓0 ↑4 ⊙2	Long (I32)
<code>eps</code>	Approximation accuracy	↓1e-20 ↑0.001 ⊙1e-15	Double (F64)

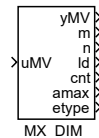
Output

<code>yA</code>	Output reference to matrix A	Reference
<code>yB</code>	Output reference to matrix B	Reference
<code>yAd</code>	Output reference to discretized matrix A	Reference
<code>yBd</code>	Output reference to discretized matrix B	Reference
<code>yP</code>	Output reference to a helper matrix	Reference
<code>yQ</code>	Output reference to a helper matrix	Reference
<code>yR</code>	Output reference to a helper matrix	Reference
<code>E</code>	Error indicator	Bool

MX_DIM – Matrix/Vector dimensions

Block Symbol

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).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uMV	Input reference to a matrix or vector	Reference
------------	---------------------------------------	-----------

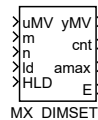
Output

yMV	Output reference to a matrix or vector	Reference
m	Number of matrix rows	Long (I32)
n	Number of matrix columns	Long (I32)
ld	Leading dimension (\geq number of rows)	Long (I32)
cnt	Count of used matrix elements	Long (I32)
amax	Count of allocated elements (\geq number of rows * number of columns)	Long (I32)
etype	Type of elements	Long (I32)
	1 Bool	
	2 Byte (U8)	
	3 Short (I16)	
	4 Long (I32)	
	5 Word (U16)	
	6 DWord (U32)	
	7 Float (F32)	
	8 Double (F64)	
	10 Large (I64)	

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

$$\text{cnt} = \text{ld} * (\text{n} - 1) + \text{m} \leq \text{amax} ,$$

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 $\text{m} < 1$ or $\text{m} > \text{ld}$,
- the number of columns $\text{n} < 1$,
- the required number of elements $\text{cnt} > \text{amax}$.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

		Reference
uMV	Input reference to a matrix or vector	
m	Number of matrix rows	Long (I32)
n	Number of matrix columns	Long (I32)
ld	Leading dimension (\geq number of rows)	Long (I32)

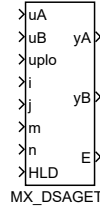
Output

<code>yMV</code>	Output reference to a matrix or vector	Reference
<code>cnt</code>	Count of used matrix elements	Long (I32)
<code>amax</code>	Count of allocated elements (\geq number of rows * number of columns)	Long (I32)
<code>E</code>	Error indicator	Bool

MX_DSAGET – Set subarray of A into B

Block Symbol

Licence: [STANDARD](#)



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 $i \geq \mathbf{MA}$ 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 $j \geq \mathbf{NA}$ 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 * N**.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uA	Input reference to matrix A	Reference
uB	Input reference to matrix B	Reference
uplo	Part of the matrix to be copied	Long (I32)
	0 All	
	1 All	
	2 Upper	
	3 Lower	
i	Index of the subarray first row	Long (I32)
j	Index of the subarray first column	Long (I32)
m	Number of matrix rows	Long (I32)
n	Number of matrix columns	Long (I32)
HLD	Hold	Bool

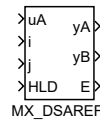
Output

yA	Output reference to matrix A	Reference
yB	Output reference to matrix B	Reference
E	Error indicator	Bool

MX_DSAREF – Set reference to subarray of A into B

Block Symbol

Licence: [STANDARD](#)



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),
- $0 > i \geq M$.
- $0 > j \geq N$.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

		Reference
uA	Input reference to matrix A	
i	Index of the subarray first row	Long (I32)
j	Index of the subarray first column	Long (I32)
HLD	Hold	Bool

Parameter

ay	Output reference of the subarray	⊙[0 0]	Double (F64)
----	----------------------------------	--------	--------------

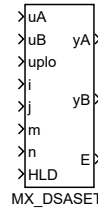
Output

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 $i \leq 0$ then **I** is set to 0 else if $i \geq \text{MB}$ 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 $j \leq 0$ then **J** is set to 0 else if $j \geq \text{NB}$ 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**.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

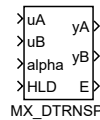
uA	Input reference to matrix A	Reference
uB	Input reference to matrix B	Reference
uplo	Part of the matrix to be copied	Long (I32)
	0 All	
	1 All	
	2 Upper	
	3 Lower	
i	Index of the subarray first row	Long (I32)
j	Index of the subarray first column	Long (I32)
m	Number of matrix rows	Long (I32)
n	Number of matrix columns	Long (I32)
HLD	Hold	Bool

Output

yA	Output reference to matrix A	Reference
yB	Output reference to matrix B	Reference
E	Error indicator	Bool

MX_DTRNSP – General matrix transposition: $B := \alpha * A^T$

Block Symbol

Licence: [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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uA	Input reference to matrix A	Reference
uB	Input reference to matrix B	Reference
alpha	Scalar coefficient alpha	Double (F64)
HLD	Hold	Bool

Output

yA	Output reference to matrix A
yB	Output reference to matrix B
E	Error indicator

Reference
Reference
Bool

MX_DTRNSQ – Square matrix in-place transposition: $A := \alpha * A^T$

Block Symbol

Licence: [STANDARD](#)

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, 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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

		Reference
<code>uA</code>	Input reference to matrix A	
<code>alpha</code>	Scalar coefficient alpha	Double (F64)
<code>HLD</code>	Hold	Bool

Output

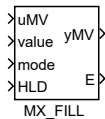
yA Output reference to matrix A
E Error indicator

Reference
Bool

MX_FILL – Fill real matrix or vector

Block Symbol

Licence: [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),
- $0 > \text{mode} > 4$.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>uMV</code>	Input reference to a matrix or vector	Reference
<code>value</code>	Fill value of matrix/vector	Double (F64)
<code>mode</code>	Fill mode	Long (I32)
	0 Value	
	1 Value	
	2 Ones	
	3 Diagonal value	
	4 Diagonal ones	
<code>HLD</code>	Hold	Bool

Output

<code>yMV</code>	Output reference to a matrix or vector	Reference
<code>E</code>	Error indicator	Bool

MX_MAT – Matrix data storage block

Block Symbol

Licence: [STANDARD](#)



Function Description

The function block **MX_MAT** allocates memory (during the block initialization) for $m * n$ 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 .

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Parameter

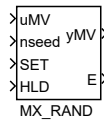
m	Number of matrix rows	↓1 ↑1000000000 ⊙10	Long (I32)
n	Number of matrix columns	↓1 ↑1000000000 ⊙10	Long (I32)
ld	Leading dimension (\geq number of rows)	↓0 ↑1000000000	Long (I32)
etype	Type of elements	⊙8	Long (I32)
	1 Bool		
	2 Byte (U8)		
	3 Short (I16)		
	4 Long (I32)		
	5 Word (U16)		
	6 DWord (U32)		
	7 Float (F32)		
	8 Double (F64)		
	10 Large (I64)		

Output

yMat	Output reference to a matrix	Reference
-------------	------------------------------	-----------

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).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

		Reference
uMV	Input reference to a matrix or vector	
nseed	Random number seed	Long (I32)
SET	Set initial value of random number generator to nseed on rising edge	Bool
HLD	Hold	Bool

Parameter

<code>BIP</code>	Bipolar random values flag	<code>Bool</code>
<code>scale</code>	Random values multiplication factor	$\odot 1.0$ <code>Double (F64)</code>

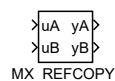
Output

<code>yMV</code>	Output reference to a matrix or vector	<code>Reference</code>
<code>E</code>	Error indicator	<code>Bool</code>

MX_REFCOPY – Copies input references of matrices A and B to their output references

Block Symbol

Licence: [STANDARD](#)



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**.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uA	Input reference to matrix A	Reference
uB	Input reference to matrix B	Reference

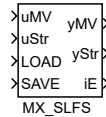
Output

yA	Output reference to matrix A	Reference
yB	Output reference to matrix B	Reference

MX_SLFS – Save or load a Matrix/Vector into file or string

Block Symbol

Licence: [STANDARD](#)



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 **REXYGEN** system file name rules applies, ie 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:

- **1: level-triggered file:** data are converted from a file to a matrix when **LOAD = on** and from a matrix to a file when **SAVE = on**; if both signals are active, it is an error and no action is taken,
- **2: edge-triggered file:** data are converted from a file to a matrix on the rising edge (**off→on**) of the **LOAD** input and from a matrix to a file on the rising edge of the **SAVE** input; if there are rising edges on both signals, it is an error and no action is taken,
- **3: level-triggered string:** data are converted from the **uStr** input to a matrix when **LOAD = on** and from a matrix to the **yStr** output when **SAVE = on**; if both signals are active, it is an error and no action is taken,
- **4: edge-triggered string:** data are converted from the **uStr** input to a matrix on the rising edge of the **LOAD** input and from a matrix to the **yStr** output on the

rising edge of the **SAVE** input; if there are rising edges on both signals, it is an error and no action is taken,

- **5: continuous string to matrix:** data are converted from the **uStr** input to a matrix at each step of the algorithm,
- **6: continuous matrix to string:** data are converted from a matrix to the **yStr** output at each step of the algorithm,
- **7: continuous file to matrix:** data are converted from a file to a matrix at each step of the algorithm,
- **8: continuous matrix to file:** data are converted from a matrix to a file at each step of the algorithm.

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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uMV	Input reference to a matrix or vector	Reference
uStr	Input string (to convert into matrix) or filename	String
LOAD	Trigger to move data to matrix/vector	Bool
SAVE	Trigger to move data from matrix/vector	Bool

Parameter

mode	Triggering mode	⊙2 Long (I32)
1	level-triggered file	
2	edge-triggered file	
3	level-triggered string	
4	edge-triggered string	
5	continuous string to matrix	
6	continuous matrix to string	
7	continuous file to matrix	
8	continuous matrix to file	

format	String/file format	⊙1	Long (I32)
	1 CSV		
	2 CSV(semicolon)		
	3 JSON		
	4 MATLAB		
prec	Number of digits for single value	↓0 ↑20 ⊙6	Long (I32)
TRN	Transposition flag		Bool
nmax	Allocated size of string	↓0	Long (I32)

Output

yMV	Output reference to a matrix or vector	Reference
yStr	String representation of the matrix/vector	String
iE	Error code	Error

MX_VEC – Vector data storage block

Block Symbol

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**.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Parameter

n	Number of vector elements	↓1 ↑10000000000 ⊙10	Long (I32)
etype	Type of elements	⊙8	Long (I32)
	1 Bool		
	2 Byte (U8)		
	3 Short (I16)		
	4 Long (I32)		
	5 Word (U16)		
	6 DWord (U32)		
	7 Float (F32)		
	8 Double (F64)		
	10 Large (I64)		

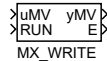
Output

yVec	Output reference to a vector	Reference
-------------	------------------------------	-----------

MX_WRITE – Write a Matrix/Vector to the console/system log

Block Symbol

Licence: [STANDARD](#)



Function Description

The function block **MX_WRITE** 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**. 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),
- $3 > \text{mchars} > 25$,
- $0 > \text{mdec} > \text{mchars} - 2$.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uMV	Input reference to a matrix or vector	Reference
RUN	Enable execution	Bool

Parameter

Symbol	Matrix/vector symbolic name for console or log output	⊙A	String
mchars	Number of characters per single element	↓3 ↑25 ⊙8	Long (I32)
mdec	Number of decimal digits per single element	↓0 ↑23 ⊙4	Long (I32)
mode	Severity mode of writing	⊙3	Long (I32)
	0 None		
	1 None		
	2 Verbose		
	3 Information		
	4 Warning		
	5 Error		

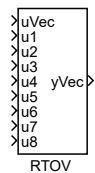
Output

yMV	Output reference to a matrix or vector	Reference
E	Error indicator	Bool

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. Parametr **etype** determines the type of input values and the type of values in the internal vector. If the vector (or matrix) connected to the input **uVec** has a different type, it will be converted.

ATTENTION: Up to version 2.50.10.x output vector is one-row-matrix. Later version (2.51.0.9525 and later) use one-column-matrix. This change was necessary for consistence in matrix operation.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uVec	Vector signal	Reference
u1..u8	Analog input of the block	Double (F64)

Parameter

nmax	Allocated size of vector	↓0 ⊙8	Long (I32)
offset	Index of the first input in vector	↓0	Long (I32)
n	Number of valid inputs	↓0 ↑8 ⊙8	Long (I32)

etype	Type of elements	⊙8 Long (I32)
	1 Bool	
	2 Byte (U8)	
	3 Short (I16)	
	4 Long (I32)	
	5 Word (U16)	
	6 DWord (U32)	
	7 Float (F32)	
	8 Double (F64)	
	10 Large (I64)	

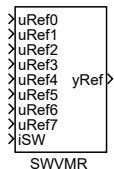
Output

yVec	Vector signal	Reference
------	---------------	-----------

SWVMR – Vector/matrix/reference signal switch

Block Symbol

Licence: [STANDARD](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uRef0..uRef7	Vector signal	Reference
iSW	Active signal selector	Long (I32)

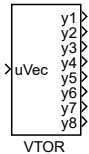
Output

yRef	Vector signal	Reference
------	---------------	-----------

VTOR – Vector demultiplexer

Block Symbol

Licence: [STANDARD](#)



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. The **etype** parameter can be used to define the type of the output values. If the vector connected to the input **uVec** has a different type, it will be converted.

This block propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uVec	Vector signal	Reference
-------------	---------------	-----------

Parameter

n	Number of valid outputs	↓0 ↑8 ⊙8	Long (I32)
offset	Index of the first output	↓0	Long (I32)
etype	Type of elements	⊙8	Long (I32)
	1 Bool		
	2 Byte (U8)		
	3 Short (I16)		
	4 Long (I32)		
	5 Word (U16)		
	6 DWord (U32)		
	7 Float (F32)		
	8 Double (F64)		
	10 Large (I64)		

Output

y1..y8	Analog output of the block	Double (F64)
---------------	----------------------------	--------------

Chapter 15

OPTIM – Optimization blocks

Contents

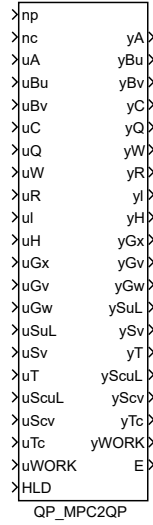
QP_MPC2QP – Conversion of MPC problem to quadratic programming	540
QP_OASES – Quadratic programming using active set method	547
QP_UPDATE – Update matrices/vectors of quadratic programming	552
SOLNP – Nonlinear optimization solver	557

The OPTIM library is tailored for optimization algorithms and processes. It includes [QCEDPOPT](#) for Quadratic Cost Economic Dispatch Problem optimization, providing advanced tools for handling complex optimization problems. The library also features blocks like [QP_MPC2QP](#) and [QP_OASES](#) for Quadratic Programming, essential in Model Predictive Control (MPC) scenarios. Additionally, [QP_UPDATE](#) is available for updating quadratic program parameters. This library is particularly useful in systems requiring high-level optimization solutions, such as in advanced control and decision-making algorithms.

QP_MPC2QP – Conversion of MPC problem to quadratic programming

Block Symbol

Licence: [ADVANCED](#)



Function Description

Quadratic Programming (QP) is a standard technique which suites very well to solve Model based Predictive Control (MPC) problems [12]. Quadratic Programming is an optimization technique that minimizes the sum of quadratic form and linear form.

The QP_MPC2QP block converts a linear MPC problem with quadratic optimization criterion to a quadratic programming problem. The block is compatible with the block [QP_UPDATE](#) and the QP solver [QP_OASES](#).

MPC problem formulation

The MPC problem consists of a discrete linear time invariant state space model

$$\begin{aligned} x_{k+1} &= Ax_k + B_u u_k + B_v v_k, \\ y_k &= Cx_k, \end{aligned} \quad (15.1)$$

where $x \in \mathbb{R}^n$ is the state vector, $u \in \mathbb{R}^{m_u}$ is the input vector, $v \in \mathbb{R}^{m_v}$ is the disturbance vector and $y \in \mathbb{R}^p$ is the output vector. Matrices $A \in \mathbb{R}^{n \times n}$, $B_u \in \mathbb{R}^{n \times m_u}$, $B_v \in \mathbb{R}^{n \times m_v}$ and $C \in \mathbb{R}^{p \times n}$ are referenced by inputs **uA**, **uBu**, **uBv** and **uC**. The model based predictive control problem is formulated as an optimization problem – minimization of the quadratic

optimality criterion (cost function) in the form

$$J = \sum_{k=1}^{n_p} \{ \hat{x}_k^T Q \hat{x}_k + \hat{x}_k^T W + \hat{u}_{k-1}^T R \hat{u}_{k-1} \}, \quad (15.2)$$

where symmetric and positive (semi-)definite matrices $Q \in \mathbb{R}^{n \times n}$ and $R \in \mathbb{R}^{m_u \times m_u}$ and the vector $W \in \mathbb{R}^n$ are referenced by inputs \mathbf{uQ} , \mathbf{uR} and \mathbf{uW} , and n_p is the prediction horizon (input \mathbf{np}).

Additional constraints on the state x and the output y may be required for the minimization process:

$$x_{\min} \leq x_k \leq x_{\max} \quad (15.3)$$

$$y_{\min} \leq y_k \leq y_{\max} \quad (15.4)$$

Predictor

From the state equation with the initial condition x_0 it holds

$$\begin{aligned} x_1 &= Ax_0 + B_u u_0 + B_v v_0 \\ x_2 &= Ax_1 + B_u u_1 + B_v v_1 \\ &= A^2 x_0 + AB_u u_0 + B_u u_1 + AB_v v_0 + B_v v_1 \\ &\vdots \\ x_k &= A^k x_0 + [A^{k-1} B_u \quad \dots \quad AB_u \quad B_u] \begin{bmatrix} u_0 \\ \vdots \\ u_{k-1} \end{bmatrix} \\ &\quad + [A^{k-1} B_v \quad \dots \quad AB_v \quad B_v] \begin{bmatrix} v_0 \\ \vdots \\ v_{k-1} \end{bmatrix} \end{aligned}$$

Thus, for the prediction horizon n_p we have

$$\begin{aligned} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_p} \end{bmatrix}}_X &= \underbrace{\begin{bmatrix} B_u & 0 & \dots & 0 \\ AB_u & B_u & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{n_p-1} B_u & A^{n_p-2} B_u & \dots & B_u \end{bmatrix}}_{S_u} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n_p-1} \end{bmatrix}}_U \\ &\quad + \underbrace{\begin{bmatrix} B_v & 0 & \dots & 0 \\ AB_v & B_v & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{n_p-1} B_v & A^{n_p-2} B_v & \dots & B_v \end{bmatrix}}_{S_v} \underbrace{\begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n_p-1} \end{bmatrix}}_V + \underbrace{\begin{bmatrix} A \\ A^2 \\ \vdots \\ A^{n_p} \end{bmatrix}}_T x_0 \end{aligned}$$

i.e.

$$X = S_u U + S_v V + T x_0 \quad (15.5)$$

Similarly, for the output equation we can get

$$\begin{aligned} \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n_p} \end{bmatrix}}_Y &= \begin{bmatrix} Cx_1 \\ Cx_2 \\ \vdots \\ Cx_{n_p} \end{bmatrix} = \underbrace{\begin{bmatrix} CB_u & 0 & \dots & 0 \\ CAB_u & CB_u & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{n_p-1}B_u & CA^{n_p-2}B_u & \dots & CB_u \end{bmatrix}}_{S_{cu}} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n_p-1} \end{bmatrix} \\ &+ \underbrace{\begin{bmatrix} CB_v & 0 & \dots & 0 \\ CAB_v & CB_v & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{n_p-1}B_v & CA^{n_p-2}B_v & \dots & CB_v \end{bmatrix}}_{S_{cv}} \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n_p-1} \end{bmatrix} + \underbrace{\begin{bmatrix} CA \\ CA^2 \\ \vdots \\ CA^{n_p} \end{bmatrix}}_{T_c} x_0 \end{aligned}$$

i.e.

$$Y = S_{cu}U + S_{cv}V + T_c x_0 \quad (15.6)$$

and standard QP matrices A_{eq} and b_{eq}

$$A_{eq} = S_{cu}L, \quad b_{eq} = -S_{cv}V - T_c x_0 \quad (15.7)$$

Predictor for control horizon less than prediction horizon

Until now, it was assumed that optimal control would be sought over the entire prediction horizon n_p . For a long prediction horizon, this leads to time-consuming optimization, which can be accelerated by choosing a control horizon n_c (input `nc`) smaller than the prediction horizon n_p . Then U can be written as

$$U = \left[\begin{array}{c} u_0 \\ u_1 \\ \vdots \\ u_{n_c-1} \\ u_{n_c-1} \\ \vdots \\ u_{n_p-1} \end{array} \right] \left\{ \begin{array}{l} n_c \\ n_p - n_c \end{array} \right\}$$

Note that the input u_k is the difference of the state and for control horizon n_c it holds $u_{k+n_c-1} = u_{k+n_c} = u_{k+n_c+1} = \dots = u_{k+n_p-1} = 0$ (for the step k). Then it can be written

as

$$U = \begin{matrix} n_c \\ n_p - n_c \end{matrix} \left\{ \begin{matrix} \begin{bmatrix} I_{m_u} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & I_{m_u} \\ & & & \mathbf{0} \end{bmatrix} \\ \underbrace{\hspace{10em}}_L \end{matrix} \right\} \begin{bmatrix} u_0 \\ \vdots \\ u_{n_c-1} \end{bmatrix} \triangleq LU_{n_c} \quad (15.8)$$

where $U \in \mathbb{R}^{n_p \cdot m_u}$, $U_{n_c} \in \mathbb{R}^{n_c \cdot m_u}$ and $I_{m_u} \in \mathbb{R}^{m_u \times m_u}$ is identity matrix.

The equations (15.5) and (15.6) are modified for U_{n_c} to

$$X = S_u LU_{n_c} + S_v V + T x_0 \quad (15.9)$$

$$Y = S_{cu} LU_{n_c} + S_{cv} V + T_c x_0 \quad (15.10)$$

Matrices $S_u L \in \mathbb{R}^{n_p \cdot n \times n_c \cdot m_u}$, $S_v \in \mathbb{R}^{n_p \cdot n \times n_p \cdot m_v}$, $T \in \mathbb{R}^{n_p \cdot n \times n}$, $S_{cu} L \in \mathbb{R}^{n_p \cdot p \times n_c \cdot m_u}$, $S_{cv} \in \mathbb{R}^{n_p \cdot p \times n_p \cdot m_v}$ and $T_c \in \mathbb{R}^{n_p \cdot p \times n}$ are computed by this block, must be allocated e.g. by the [MX_MAT](#) blocks and references to the preallocated matrices must be connected to the block inputs `uSuL`, `uSv`, `uT`, `uScuL`, `uScv` and `uTc`.

The default value of the matrix $L \in \mathbb{R}^{n_p \cdot m_u \times n_c \cdot m_u}$ in equation 15.8 selects the first n_c subvectors $u_i, i = 0, \dots, n_c - 1$ from U . The block also allows to select n_c subvectors u_i with arbitrary indices from $0, \dots, n_p$, which are contained in the integer vector of dimension n_c referenced by the input `ul`. The elements of this vector must form an increasing sequence. If the input `ul` is not connected, the default value of L is used (the same value of L is obtained if the vector referenced by `ul` is equal to $[0, 1, \dots, n_c - 1]^T$).

Conversion of MPC with the same prediction and control horizons to QP

The standard form of cost function for QP is

$$J_{QP} = h U^T H U + U^T G \quad (15.11)$$

where U is a vector of optimal control sequence, H is a symmetric and positive (semi-) definite Hessian matrix, G is a gradient vector and h is a scalar constant which is usually equal to 1 or 1/2.

The cost function (15.2) can be modified to the form

$$\begin{aligned}
J &= \sum_{k=1}^{n_p} \{x_k^T Q x_k + x_k^T W + u_{k-1}^T R u_{k-1}\} \\
&= [x_1 \quad x_2 \quad \dots \quad x_{n_p}] \underbrace{\begin{bmatrix} Q & & \\ & Q & \\ & & \ddots \\ & & & Q \end{bmatrix}}_{\bar{Q} = \bar{Q}^T} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_p} \end{bmatrix} + [x_1 \quad x_2 \quad \dots \quad x_{n_p}] \underbrace{\begin{bmatrix} W \\ W \\ \vdots \\ W \end{bmatrix}}_{\bar{W}} \\
&\quad + [u_0 \quad u_1 \quad \dots \quad u_{n_p-1}] \underbrace{\begin{bmatrix} R & & \\ & R & \\ & & \ddots \\ & & & R \end{bmatrix}}_{\bar{R} = \bar{R}^T} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n_p-1} \end{bmatrix} \\
&= X^T \bar{Q} X + X^T \bar{W} + U^T \bar{R} U \\
&= (U^T S_u^T + V^T S_v^T + x_0^T T^T) \bar{Q} (S_u U + S_v V + T x_0) \\
&\quad + (U^T S_u^T + V^T S_v^T + x_0^T T^T) \bar{W} + U^T \bar{R} U \\
&= U^T S_u^T \bar{Q} S_u U + U^T S_u^T \bar{Q} S_v V + U^T S_u^T \bar{Q} T x_0 \\
&\quad + V^T S_v^T \bar{Q} S_u U + V^T S_v^T \bar{Q} S_v V + V^T S_v^T \bar{Q} T x_0 \\
&\quad + x_0^T T^T \bar{Q} S_u U + x_0^T T^T \bar{Q} S_v V + x_0^T T^T \bar{Q} T x_0 \\
&\quad + U^T S_u^T \bar{W} + V^T S_v^T \bar{W} + x_0^T T^T \bar{W} + U^T \bar{R} U \\
&= U^T (S_u^T \bar{Q} S_u + \bar{R}) U + U^T S_u^T (2\bar{Q} S_v V + 2\bar{Q} T x_0 + \bar{W}) \\
&\quad + \underbrace{V^T S_v^T (\bar{Q} S_v V + 2\bar{Q} T x_0 + \bar{W}) + x_0^T T^T (\bar{Q} T x_0 + \bar{W})}_{J_{\text{dif}}} \triangleq J_{QP} + J_{\text{dif}}
\end{aligned} \tag{15.12}$$

where J_{dif} is a constant independent of U . From here follows

$$J_{QP} = U^T (S_u^T \bar{Q} S_u + \bar{R}) U + U^T S_u^T (2\bar{Q} S_v V + 2\bar{Q} T x_0 + \bar{W}) \tag{15.13}$$

Comparing this equation with (15.11), it is obvious that

$$\begin{aligned}
H &= \frac{1}{h} (S_u^T \bar{Q} S_u + \bar{R}) \\
G &= S_u^T (2\bar{Q} S_v V + 2\bar{Q} T x_0 + \bar{W})
\end{aligned} \tag{15.14}$$

Conversion of MPC with control horizon less than prediction horizon

Similarly as in previous subsection we can get for $n_c < n_p$

$$\begin{aligned} H &= \frac{1}{h} L^T (S_u^T \bar{Q} S_u + \bar{R}) L \\ G &= (S_u L)^T (2 \bar{Q} S_v V + 2 \bar{Q} T x_0 + \bar{W}) \triangleq G_v V + G_x x_0 + G_w \end{aligned} \quad (15.15)$$

where matrix L is defined by (15.8). The Hessian matrix H is a constant matrix for all steps k of the MPC. But gradient vector G is generally changing in each step k because vectors V and x_0 are changing. Therefore, G is composed of parts G_v , G_x and G_w , which are already constant vectors. The matrix H and vectors G_v , G_x and G_w are computed by this function block and are referenced by inputs **uH**, **uGv**, **uGx** and **uGw** which must be preallocated. The scalar constant h is the function block parameter.

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

np	Prediction horizon	↓1 ↑1000000	Long (I32)
nc	Control horizon	↓1 ↑1000000	Long (I32)
uA	Input reference to system matrix A		Reference
uBu	Input reference to input matrix Bu of control vector u		Reference
uBv	Input reference to input matrix Bv of disturbance vector v		Reference
uC	Input reference to output matrix C		Reference
uQ	Input reference to symmetric matrix Q in cost function		Reference
uW	Input reference to vector W in cost function		Reference
uR	Input reference to symmetric matrix R in cost function		Reference
ul	Input reference to integer index vector l		Reference
uH	Input reference to Hessian matrix H		Reference
uGx	Input reference to part of gradient vector G corresponding to state vector x		Reference
uGv	Input reference to part of gradient vector G corresponding to disturbance vector v		Reference
uGw	Input reference to part of gradient vector G corresponding to vector W		Reference
uSuL	Input reference to work matrix Su*L		Reference
uSv	Input reference to work matrix Sv		Reference
uT	Input reference to work matrix T		Reference
uScuL	Input reference to work matrix Scu*L		Reference
uScv	Input reference to work matrix Scv		Reference
uTc	Input reference to work matrix Tc		Reference
uWORK	Input reference to matrix WORK		Reference
HLD	Hold		Bool

Output

yA	Output reference to system matrix A	Reference
yBu	Output reference to input matrix Bu of control vector u	Reference
yBv	Output reference to input matrix Bv of disturbance vector v	Reference
yC	Output reference to output matrix C	Reference
yQ	Output reference to symmetric matrix Q in cost function	Reference
yW	Output reference to vector W in cost function	Reference
yR	Output reference to symmetric matrix R in cost function	Reference
yl	Output reference to integer index vector l	Reference
yH	Output reference to Hessian matrix H	Reference
yGx	Output reference to part of gradient vector G corresponding to state vector x	Reference
yGv	Output reference to part of gradient vector G corresponding to disturbance vector v	Reference
yGw	Output reference to part of gradient vector G corresponding to vector W	Reference
ySuL	Output reference to work matrix Su*L	Reference
ySv	Output reference to work matrix Sv	Reference
yT	Output reference to work matrix T	Reference
yScuL	Output reference to work matrix Scu*L	Reference
yScv	Output reference to work matrix Scv	Reference
yTc	Output reference to work matrix Tc	Reference
yWORK	Output reference to matrix WORK	Reference
E	Error indicator	Bool

QP_OASES – Quadratic programming using active set method

Block Symbol

Licence: [ADVANCED](#)



Function Description

The `QP_OASES` block solves a quadratic programming problem using active set method [13]:

$$\begin{aligned} \min_{x \in \mathbb{R}^{nV}} \quad & \frac{1}{2}x^T Hx + x^T G, \\ \text{s. t.} \quad & \text{lb}A \leq Ax \leq \text{ub}A, \\ & \text{lb} \leq x \leq \text{ub}, \end{aligned}$$

where nV is number of variables, nC is number of constraints, the Hessian matrix $H \in \mathbb{R}^{nV \times nV}$ is symmetric and positive (semi-)definite, the gradient vector $G \in \mathbb{R}^{nV}$, the constraint matrix $A \in \mathbb{R}^{nC \times nV}$, bound vectors $\text{lb}, \text{ub} \in \mathbb{R}^{nV}$ and constraint vectors $\text{lb}A, \text{ub}A \in \mathbb{R}^{nC}$.

The block wraps the `qpOASES` library¹, the use of which is described in the manual [14].

The output references `yH`, `yG`, `yA`, `yLB`, `yUB`, `yLBA`, `yUBA`, `yXopt` and `yYopt` are always set to the corresponding input `uH`, `uG`, `uA`, `uLB`, `uUB`, `uLBA`, `uUBA`, `uXopt` and `uYopt`. If the input `uQP` is not connected, the particular quadratic problem (QP) is allocated in the first execution of the function block (see below) and the output `yQP` is set to the reference of the allocated QP. If `uQP` is connected (to the `yQP` output of the previous `QP_OASES` block), the `yQP` output is set to `uQP` and the block works with an already allocated QP.

The block uses internal variables `nV` and `nC`. The value of `nV` is set to the number of rows of the vector G referenced by `uG`, the value of `nV` is set to the number of rows of

¹`qpOASES` which is distributed under the GNU Lesser General Public License, see Appendix A of [14].

the matrix A referenced by `uA`. If the reference `uA` is not defined (the matrix A is not connected), the value `nC` = 0.

To solve the QP problem, a `QProblem` object is created in the generic case (see Chapter 3 of [14]). However, the block can also solve the following special cases depending on the input references and the `hessianType` parameter:

uH not connected. In this case, it is assumed that Hessian matrix has a trivial value of the identity or zero matrix. The `hessianType` parameter must be equal to `HST_ZERO` or `HST_IDENTITY`, see Section 4.5 of the manual [14].

uA not connected. In this case, the constraint matrix A is not used (`nC` = 0), the `QProblemB` object is created, see Section 4.3 of the manual [14]. The `hessianType` parameter can be any allowed value.

VAR = on. If the input `VAR` = on during the first time the block is executed, an object of class `SQProblem` is created, see Section 4.2 of the manual [14]. In this case, all input matrices and vectors can change in each execution step in which `VAR` = on.

To obtain the solution of the QP problem, at least one of the input references `uXopt` and `uYopt` must be defined (connected to a vector). If connected to `uXopt`, the `yXopt` output will refer to the primal solution $X_{opt} \in \mathbb{R}^{nV}$, if connected to `uYopt`, the `yYopt` output will refer to the dual solution $Y_{opt} \in \mathbb{R}^{nV+nC}$ of the QP problem. If both inputs are connected, both solutions will be obtained in each step. The optimal objective function value is indicated on the output `objval`.

The integer input `unWSR` specifies the maximum number of working set recalculations to be performed during the initial homotopy, see Section 3.2 of the manual [14]. Output `ynWSR` contains the number of working set recalculations actually performed. If the double input `utime` is connected and has a positive value, it contains the maximum allowed CPU time in seconds for the whole initialisation. The actually required CPU time for the initialization is indicated on the output `ytime`.

At least one vector must be connected to the `uXopt` or `uYopt` inputs and both has be connected to obtain the solution of the QP problem. If `uXopt` is connected, the `yXopt` output will refer to the primary X_{opt} solution, if `uYopt` is connected, the `yYopt` output will refer to the dual Y_{opt} solution of the QP task. If both inputs are connected, both solutions will be obtained in each step.

If the input `INIT` = on then the particular allocated QP problem is re-initialized. The `INIT` should be on for only a single period (edge) because no solution is computed during the QP initialisation. If `HLD` = on then nothing is computed.

The error flag `E` is set to on and the error code `iE` is set to zero if:

- the reference `uG` or `uLB` or `uUB` is not defined (i.e. input `uG` or `uLB` or `uUB` is not connected),
- the reference `uA` is defined and `uLBA` or `uUBA` is not defined (i.e. input `uA` is connected and `uLBA` or `uUBA` is not connected),

- both references `uXopt` and `uYopt` are not defined (i.e. neither of the inputs `uXopt` and `uYopt` is connected),
- the Hessian matrix H referenced by `uH` has a different number of rows and columns than `nV`,
- the number of rows of vectors referenced by `uLB` and `uUB` is not equal to `nV` (or the number of their columns is not equal to 1),
- the number of rows of vectors referenced by `uLBA` and `uUBA` is not equal to `nC` (or the number of their columns is not equal to 1) if the matrix A referenced by `uA` is connected,
- the number of rows of the vector referenced by `uXopt` is not equal to `nV` or the number of rows of the vector referenced by `yOpt` is not equal to `nV+nC` (or the number of their columns is not equal to 1),
- the internal space for transposed copies of matrices H or A is too small.

If the flag `E` is set to `on` and the error code `iE` is not zero then `iE` indicates the qpOASES error code, see the include file `MessageHandling.hpp` from qpOASES library.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>uQP</code>	Input reference to quadratic programming problem	Reference
<code>uH</code>	Input reference to Hessian matrix H	Reference
<code>uG</code>	Input reference to gradient vector G	Reference
<code>uA</code>	Input reference to constraint matrix A	Reference
<code>uLB</code>	Input reference to lower bound vector LB	Reference
<code>uUB</code>	Input reference to upper bound vector LB	Reference
<code>uLBA</code>	Input reference to lower constraints' bound vector LB	Reference
<code>uUBA</code>	Input reference to upper constraints' bound vector LB	Reference
<code>uXopt</code>	Input reference to primal optimal solution	Reference
<code>uYopt</code>	Input reference to dual optimal solution	Reference
<code>unWSR</code>	Maximum number of initial working set recalculations	Long (I32)
<code>utime</code>	Maximum allowed CPU time in seconds for the whole initialisation	Double (F64)
<code>VAR</code>	Indicates that matrices H and A are time varying	Bool
<code>INIT</code>	Calls <code>init()</code> function instead of <code>hotstart()</code> in each block execution	Bool
<code>HLD</code>	Hold	Bool

Parameter

nVmax	Maximum number of optimization variables nV	Long (I32)
nCmax	Maximum number of optimization constraints nC	Long (I32)
hessianType	Hessian matrix type	Long (I32)
printLevel	Print level	Long (I32)
enableRamping	Enable ramping	Bool
enableFarBounds	Enable use of far bounds	Bool
enableFlippingBounds	Enable use of flipping bounds	Bool
enableRegularisation	Enable regularisation of semidefinite Hessian matrix	Bool
enableFullLITests	Enable use of condition-hardened linear independence tests	Bool
enableNZCTests	Enable nonzero curvature tests	Bool
enableDriftCorrection	Frequency of drift corrections (0 = off)	Long (I32)
enableCholeskyRefact	Frequency of full refactorization of projected Hessian (0 = off)	Long (I32)
enableEqualities	Equalities shall be always treated as active constraints	Bool
terminationTolerance	Termination tolerance	Double (F64)
boundTolerance	If upper and lower limits differ less than this tolerance, they are regarded equal, i.e. as equality constraint	Double (F64)
boundRelaxation	Initial relaxation of bounds to start homotopy and initial value for far bounds.	Double (F64)
epsNum	Numerator tolerance for ratio tests	Double (F64)
epsDen	Denominator tolerance for ratio tests	Double (F64)
maxPrimalJump	Maximum allowed jump in primal variables in nonzero curvature tests	Double (F64)
maxDualJump	Maximum allowed jump in dual variables in linear independence tests	Double (F64)
initialRamping	Start value for ramping strategy	Double (F64)
finalRamping	Final value for ramping strategy	Double (F64)
initialFarBounds	Initial size of Far Bounds	Double (F64)
growFarBounds	Factor to grow Far Bounds	Double (F64)
initialStatusBounds	Initial status of bounds at first iteration	Long (I32)
epsFlipping	Tolerance of squared entry of Cholesky diagonal which triggers flipping bounds	Double (F64)
numRegularisationSteps	Maximum number of successive regularisation steps	Long (I32)
epsRegularisation	Scaling factor of identity matrix used for Hessian regularisation	Double (F64)
numRefinementSteps	Maximum number of iterative refinement steps	Long (I32)
epsIterRef	Early termination tolerance for iterative refinement	Double (F64)
epsLITests	Tolerance for linear independence tests	Double (F64)
epsNZCTests	Tolerance for nonzero curvature tests	Double (F64)

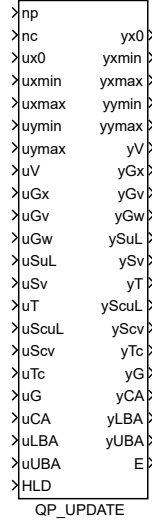
Output

yQP	Output reference to quadratic programming problem	Reference
yH	Output reference to Hessian matrix H	Reference
yG	Output reference to gradient vector G	Reference
yA	Output reference to constraint matrix A	Reference
yLB	Output reference to lower bound vector LB	Reference
yUB	Output reference to upper bound vector LB	Reference
yLBA	Output reference to lower constraints' bound vector LB	Reference
yUBA	Output reference to upper constraints' bound vector LB	Reference
yXopt	Output reference to primal optimal solution	Reference
yYopt	Output reference to dual optimal solution	Reference
ynWSR	Number of performed initial working set recalculations	Long (I32)
ytime	Elapsed CPU time in seconds for the whole initialisation	Double (F64)
objval	Optimal objective function value	Double (F64)
E	Error indicator	Bool
iE	Error code	Long (I32)

QP_UPDATE – Update matrices/vectors of quadratic programming

Block Symbol

Licence: [ADVANCED](#)



Function Description

The QP_UPDATE function block cooperates with the [QP_MPC2QP](#) block which converts the MPC problem described by equations (15.1)–(15.4) with prediction horizon n_p and control horizon n_c (inputs **np** and **nc**), to quadratic programming and pre-computes the Hessian matrix H , parts of the gradient vector G_x , G_v , G_w , matrices determining state constraints $S_u L$, S_v , T , and matrices determining output constraints $S_{cu} L$, S_{cv} , T_c . Besides the constant Hessian matrix H , the other vectors and matrices are connected to input references **uGx**, **uGv**, **uGw**, **uSuL**, **uSv**, **uT**, **uScuL**, **uScv** and **uTc**.

This block updates the QP problem for the given time instant with current values of state vector initial condition x_0 , state vector bounds x_{\min} and x_{\max} , output vector bounds y_{\min} and y_{\max} , vector V (see eq. (15.5)) of disturbance prediction vectors v_k , $k = 0, \dots, n_p - 1$. These vectors are referenced by inputs **ux0**, **uxmin**, **uxmax**, **uymin**, **uymax** and **uV**.

First, the gradient vector G referenced by the input **uG** is updated according to the equation (15.15):

$$G = G_x x_0 + G_v V + G_w.$$

The state constraints (15.3) can be rewritten using (15.9) for the prediction horizon n_p to

$$X_{\min} - S_v V - T x_0 \leq S_u L U_{n_c} \leq X_{\max} - S_v V - T x_0,$$

where X_{\min} resp. X_{\max} is a vector composed of n_p copies of the x_{\min} resp. x_{\max} vector. Similarly, the output constraints (15.4) can be rewritten using (15.10) to

$$Y_{\min} - S_{cv}V - T_c x_0 \leq S_{cu}LU_{n_c} \leq Y_{\max} - S_{cv}V - T_c x_0, .$$

where Y_{\min} resp. Y_{\max} is a vector composed of n_p copies of the y_{\min} resp. y_{\max} vector.

The more compact form of these two equations is

$$\underbrace{\begin{bmatrix} X_{\min} - S_v V - T x_0 \\ Y_{\min} - S_{cv} V - T_c x_0 \end{bmatrix}}_{\text{lbA}} \leq \underbrace{\begin{bmatrix} S_u L \\ S_{cu} L \end{bmatrix}}_{\text{CA}} U_{n_c} \leq \underbrace{\begin{bmatrix} X_{\max} - S_v V - T x_0 \\ Y_{\max} - S_{cv} V - T_c x_0 \end{bmatrix}}_{\text{ubA}}, \quad (15.16)$$

where the matrix CA and vectors lbA , ubA are computed by this block, must be allocated e.g. by the **MX_MAT** blocks and references to the preallocated matrices must be connected to the block inputs **uCA**, **uLBA** and **uUBA**.

The last equation 15.16 is a general form of QP constraints. It covers both equality or inequality constraints for states and outputs.

If no state constraints are required, leave the **uxmin**, **uxmax**, **uSuL**, **uSv** and **uT** inputs disconnected. Then equation 15.16 gets the form

$$\underbrace{[X_{\min} - S_v V - T x_0]}_{\text{lbA}} \leq \underbrace{[S_u L]}_{\text{CA}} U_{n_c} \leq \underbrace{[X_{\max} - S_v V - T x_0]}_{\text{ubA}}. \quad (15.17)$$

Similarly, if no output constraints are required, leave the **uymin**, **ymax**, **uScuL**, **uScv** and **uTc** inputs disconnected. Then equation 15.16 gets the form

$$\underbrace{[Y_{\min} - S_{cv} V - T_c x_0]}_{\text{lbA}} \leq \underbrace{[S_{cu} L]}_{\text{CA}} U_{n_c} \leq \underbrace{[Y_{\max} - S_{cv} V - T_c x_0]}_{\text{ubA}}. \quad (15.18)$$

The output references **yx0**, **yxmin**, **ymax**, **yymin**, **ymax**, **yV**, **yGx**, **yGv**, **yGw**, **ySuL**, **ySv**, **yT**, **yScuL**, **yScv**, **yTc**, **yG**, **yCA**, **yLBA** and **yUBA** are always set to the corresponding input **ux0**, **uxmin**, **uxmax**, **uymin**, **ymax**, **uV**, **uGx**, **uGv**, **uGw**, **uSuL**, **uSv**, **uT**, **uScuL**, **uScv**, **uTc**, **uG**, **uCA**, **uLBA** and **yUBA**.

If **HLD** = **on** then nothing is computed.

The error flag **E** is set to **on** if:

- the prediction horizon **np** < 1 or control horizon **nc** < 1, or **nc** > **np**,
- the reference **ux0** is not defined or the element type of the array it references is not **Double** (F64),
- the internal variable **bStateConstr** = **on** and at least one of the references **uxmin**, **uxmax** is not defined, or the element type of at least one of the arrays they reference is not **Double** (F64),

- the internal variable `bOutputConstr` = `on` and the reference `uymin` is defined and the element type of the array it references is not `Double` (F64),
- the internal variable `bOutputConstr` = `on` and the reference `uymax` is defined and the element type of the array it references is not `Double` (F64),
- the reference `uV` is defined and the element type of the array it references is not `Double` (F64),
- the reference `uG` is defined and at least one of the references `uGx`, `uGv`, `uSuL`, `uSv` or `uT` is not defined,
- the reference `uG` is defined and the element type of the array it references is not `Double` (F64), or the reference `uGx` is defined and the element type of the array it references is not `Double` (F64), or the reference `uGv` is defined and the element type of the array it references is not `Double` (F64), or the reference `uGw` is defined and the element type of the array it references is not `Double` (F64),
- the reference `uSuL` is defined and the element type of the array it references is not `Double` (F64), or the reference `uSv` is defined and the element type of the array it references is not `Double` (F64), or the reference `uT` is defined and the element type of the array it references is not `Double` (F64),
- the reference `uScuL` is defined and the element type of the array it references is not `Double` (F64), or the reference `uScv` is defined and the element type of the array it references is not `Double` (F64), or the reference `uTc` is defined and the element type of the array it references is not `Double` (F64),
- the reference `uCA` or `uLBA` or `uUBA` or the element type of at least one of the arrays they reference is not `Double` (F64),
- the arrays referenced by defined references are too small or have incompatible dimensions.

If `E` = `on`, see the system log for details.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>np</code>	Prediction horizon	↓1 ↑1000000	Long (I32)
<code>nc</code>	Control horizon	↓1 ↑1000000	Long (I32)
<code>ux0</code>	Input reference to initial condition vector <code>x0</code> of the state vector		Reference
<code>uxmin</code>	Input reference to vector of low limits of the state vector elements		Reference
<code>uxmax</code>	Input reference to vector of high limits of the state vector elements		Reference

uymin	Input reference to vector of low limits of the output inequalities	Reference
uymax	Input reference to vector of high limits of the output inequalities	Reference
uV	Input reference to vector predicted disturbances	Reference
uGx	Input reference to part of gradient vector G corresponding to state vector x	Reference
uGv	Input reference to part of gradient vector G corresponding to disturbance vector v	Reference
uGw	Input reference to part of gradient vector G corresponding to vector W	Reference
uSuL	Input reference to work matrix $Su*L$	Reference
uSv	Input reference to work matrix Sv	Reference
uT	Input reference to work matrix T	Reference
uScuL	Input reference to work matrix $Scu*L$	Reference
uScv	Input reference to work matrix Scv	Reference
uTc	Input reference to work matrix Tc	Reference
uG	Input reference to gradient vector G	Reference
uCA	Input reference to QP constraints matrix CA	Reference
uLBA	Input reference to lower constraints' bound vector LB	Reference
uUBA	Input reference to upper constraints' bound vector LB	Reference
HLD	Hold	Bool

Output

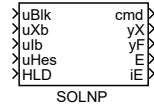
yx0	Output reference to initial condition vector $x0$ of the state vector x	Reference
yxmin	Output reference to vector of low limits of the state vector elements	Reference
yxmax	Output reference to vector of high limits of the state vector elements	Reference
yymin	Output reference to vector of low limits of the output inequalities	Reference
yymax	Output reference to vector of high limits of the output inequalities	Reference
yV	Output reference to vector predicted disturbances	Reference
yGx	Output reference to part of gradient vector G corresponding to state vector x	Reference
yGv	Output reference to part of gradient vector G corresponding to disturbance vector v	Reference
yGw	Output reference to part of gradient vector G corresponding to vector W	Reference
ySuL	Output reference to work matrix $Su*L$	Reference
ySv	Output reference to work matrix Sv	Reference
yT	Output reference to work matrix T	Reference

yScuL	Output reference to work matrix Scu^*L	Reference
yScv	Output reference to work matrix Scv	Reference
yTc	Output reference to work matrix Tc	Reference
yG	Output reference to gradient vector G	Reference
yCA	Output reference to QP constraints matrix CA	Reference
yLBA	Output reference to lower constraints' bound vector LB	Reference
yUBA	Output reference to upper constraints' bound vector LB	Reference
E	Error indicator	Bool

SOLNP – Nonlinear optimization solver

Block Symbol

Licence: [MATRIX](#)



Function Description

The **SOLNP** block is designed to solve a general Nonlinear Programming Problem (NLP) in the form

$$\min f(x)$$

subject to

$$\begin{aligned} g(x) &= 0, \\ l_h &\leq h(x) \leq u_h, \\ l_x &\leq x \leq u_x, \end{aligned}$$

where

- $x \in \mathbb{R}^n$ is a vector of decision variables to be optimized,
- $f(x)$ is a cost function (objective function),
- $g(x)$ is a vector of equality constraints,
- $h(x)$ is a vector of inequality constraints,
- l_h is a vector of lower bounds on inequality constraints,
- u_h is a vector of upper bounds on inequality constraints,
- l_x is a vector of lower bounds on decision variables,
- u_x is a vector of upper bounds on decision variables.

The **SOLNP** solver tries to find optimal solution $x^* = \arg \min_x f(x)$ with the use of Augmented Lagrangian Method and Sequential Quadratic Programming (SQP).

For technical details, refer to [15] and [16]. In general, f , g , and h are any nonlinear smooth functions.

Regarding implementation, the objective function, equality, and inequality constraints can be defined separately, for example, via the **REXLANG** block which allows the definition of custom functions. Its output **y0** is then propagated back to **SOLNP**'s input **uBlk**.

The following auxiliary variables may be used in the formulation of the optimization problem to be solved by **SOLNP** block:

- Vector of initial estimates: x_0 .
- Matrix x_b defining range of decision variables, referred to as simple bounds: $x_b = [l_x, u_x]$.
- Matrix uIb defining inequality constraints: $uIb = [l_h, u_h]$ (optional, only used if the optimization problem includes inequality constraints).

The block can be configured in one of three ways depending on how the input vector uXb is constructed:

1. $uXb = [x_0, x_b]$: The solver will use the given initial estimate and simple bounds.
2. $uXb = x_0$: The solver will only use the given initial estimate.
3. $uXb = x_b$: The solver only uses the given simple bounds. Initial estimate is constructed automatically as $x_0 = \frac{l_x + u_x}{2}$.

The vector uXb may have one, two, or three columns; and the number of rows corresponds to the number of decision variables (dimension of x).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uBlk	Associated REXLANG block reference	Reference
uXb	Initial estimate and simple bounds	Reference
uIb	Bounds on inequality constraints (optional)	Reference
uHes	User-supplied Hessian matrix (optional)	Reference
HLD	Hold	Bool

Parameter

rho	Penalty parameter for the augmented Lagrangian objective function	Double (F64) ⊙1.0
MaxIterMajor	Maximum number of major iterations	↓0 ⊙400 Long (I32)
MaxIterMinor	Maximum number of minor iterations	↓0 ⊙200 Long (I32)
delta	Perturbation parameter for numerical gradient calculation	Long (I32) ↓1.79769e+308 ↑double
tolerance	Tolerance parameter for feasibility and optimality	Long (I32) ↓1.79769e+308 ↑double
nmax	Allocated size of array	↓10 ↑100000 ⊙100 Long (I32)

Output

cmd	Phase identification for the associated REXLANG block	Long (I32)
yX	Input vector for the associated REXLANG block	Reference

yF	Output vector for the associated REXLANG block	Reference
E	Error indicator	Bool
iE	Error code	Long (I32)

Chapter 16

SPEC – Special blocks

Contents

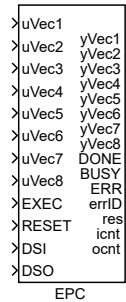
EPC – External program call	562
HTTP – Block for generating HTTP GET or POST requests (obsolete)	565
HTTP2 – Block for generating HTTP requests	567
RDC – Remote data connection	569
REXLANG – User programmable block	573
SMTP – Block for sending e-mail alerts via SMTP	593
STEAM – Steam and water properties	595
UART – UART communication block	600

The SPEC library encompasses a diverse set of functional blocks designed to integrate a wide range of functionalities into automation, control systems, and communication protocols. From facilitating precise thermodynamic calculations with the [STEAM](#) block to enabling seamless data communication through [UART](#) and [SMTP](#), the library serves as a comprehensive toolkit for engineers and developers. It includes specialized blocks for executing external programs ([EPC](#)), handling web-based requests ([HTTP2](#)). Additionally, it offers unique input-output solutions ([RDC](#)) and a versatile programming environment with [REXLANG](#).

EPC – External program call

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **EPC** block executes an external program when a rising edge (**off**→**on**) is detected at the **EXEC** input. The program's name and parameters are specified in the **cmd** parameter, formatted exactly as they would be in the operating system's command line.

Data can be passed between the **REXYGEN** system and the external program via files. The file format, specified by the **format** parameter, is text-based to ensure compatibility and ease of use with various software tools. For instance, data can be loaded in **MATLAB** with the command:

```
values = load('-ASCII', 'epc_inputVec1');
```

and in **SCILAB** with:

```
values = read('epc_inputVec1', -1, 32);
```

Adjust the file name and number of columns according to your specific project needs. Data can also be retrieved back into the **REXYGEN** system using the same file format.

The **EPC** block operates in two modes:

Basic Mode: Triggered by a rising edge on **EXEC**, this mode reads the current input data and stores it in designated files specified by the **ifns** parameter. Each *i*-th input vector **uVec<i>** is saved to its corresponding *i*-th file.

Sampling Mode: In this mode, data from the input vectors are continually saved to files at each algorithm cycle, facilitating real-time data logging.

In both modes, data from one time instant are saved in one line in the file. Similarly, data from output files are retrieved and mapped to the block outputs (**yVec<i>**), with each line from the *i*-th output file going to the *i*-th output vector.

The inputs working in the *sampling mode* are defined by the **sl** list¹. 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.

To merge multiple signals into one vector, use the **RTOV** block. To split a vector into individual signals, use the **VTOR** block. These blocks can be chained for processing vectors of any size, increasing the block's flexibility in handling complex data structures.

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:
`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 alphanumeric 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 also used for execution of some operating system functions.

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

	uVec1...uVec8 Input vector signal	Reference
EXEC	External program is called on rising edge	Bool
RESET	Block reset	Bool
DSI	Disable inputs sampling	Bool
DSO	Disable outputs sampling	Bool

¹E.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.

Parameter

<code>cmd</code>	Operating system command to execute		String
<code>ifns</code>	Input filenames (separated by semicolon)		String
		\odot <code>epc_uVec1;epc_uVec2</code>	
<code>ofns</code>	Output filenames (separated by semicolon)		String
		\odot <code>epc_yVec1;epc_yVec2</code>	
<code>sl</code>	List of sampled inputs (comma-separated)	$\downarrow 0 \uparrow 255 \odot 85$	Long (I32)
<code>ifm</code>	Maximum number of input samples	$\odot 10000$	Long (I32)
<code>format</code>	Format of input and output files	$\odot 1$	Long (I32)
	1 Space-delimited values		
	2 CSV (decimal point and commas)		
	3 CSV (decimal comma and semicolons)		
<code>nmax</code>	Maximum output vector length	$\downarrow 2 \uparrow 1000000 \odot 100$	Long (I32)

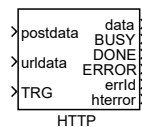
Output

<code>yVec1..yVec8</code>	Output vector signal	Reference
<code>DONE</code>	External program finished	Bool
<code>BUSY</code>	External program running	Bool
<code>ERR</code>	Error indicator	Bool
<code>errID</code>	Error code	Error
<code>res</code>	External program return code	Long (I32)
<code>icnt</code>	Current input sample	Long (I32)
<code>ocnt</code>	Current output sample	Long (I32)

HTTP – Block for generating HTTP GET or POST requests (obsolete)

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **HTTP** block is obsolete, replace it by the [HTTP2](#) block.

The **HTTP** block performs a single shot HTTP 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.

A HTTP request is started by the rising edge (**off**→**on**) on the **TRG** input. Then the **BUSY** output is set to **on** until a request is finished, which is signaled by the **DONE**=**on** 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** output 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 ([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 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`.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>postdata</code>	Data to put in HTTP POST request	String
<code>urldata</code>	Data to append to URL address	String
<code>TRG</code>	Trigger of the selected action	Bool

Parameter

<code>url</code>	URL address to send the HTTP request to		String
<code>method</code>	HTTP request type	⊙1	Long (I32)
	1 GET		
	2 POST		
	3 PUT		
	4 DELETE		
	5 HEAD		
	6 TRACE		
	7 PATCH		
	8 OPTIONS		
	9 CONNECT		
<code>user</code>	User name		String
<code>password</code>	Password		String
<code>certificate</code>	Authentication certificate		String
<code>VERIFY</code>	Enable server verification (valid certificate)		Bool
<code>postmime</code>	MIME encoding for POST request	⊙application/json	String
<code>acceptmime</code>	MIME encoding for GET request	⊙application/json	String
<code>timeout</code>	Timeout interval	⊙5.0	Double (F64)
<code>BLOCKING</code>	Wait for the operation to finish		Bool
<code>nmax</code>	Allocated size of string	↓0 ↑65520	Long (I32)
<code>postmax</code>	Allocated memory for POST request data	↓128 ↑65520 ⊙256	Long (I32)
<code>datamax</code>	Allocated memory for HTTP response	↓128 ↑10000000 ⊙1024	Long (I32)

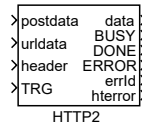
Output

<code>data</code>	Response data	String
<code>BUSY</code>	Sending HTTP request	Bool
<code>DONE</code>	HTTP request processed	Bool
<code>ERROR</code>	Error indicator	Bool
<code>errId</code>	Error code	Error
<code>hterror</code>	HTTP response	Long (I32)

HTTP2 – Block for generating HTTP requests

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **HTTP2** block performs a single shot HTTP 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 rising edge (**off**→**on**) on the **TRG** input. Then the **BUSY** output is set to **on** until a request is finished, which is signaled by the **DONE**=**on** output. In case of an error, the **ERROR** output is set to **on**. The **errId** output carries last error identified by **REXYGEN** system error code. The **herror** output carries a HTTP status code. All data sent back by server to client is stored in 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`.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>postdata</code>	Data to put in HTTP POST request	String
<code>urldata</code>	Data to append to URL address	String
<code>header</code>	Additional header fields	String
<code>TRG</code>	Trigger of the selected action	Bool

Parameter

<code>url</code>	URL address to send the HTTP request to	String
<code>method</code>	HTTP request type	⊙1 Long (I32)
	1 GET	
	2 POST	
	3 PUT	
	4 DELETE	
	5 HEAD	
	6 TRACE	
	7 PATCH	
	8 OPTIONS	
	9 CONNECT	
<code>user</code>	User name	String
<code>password</code>	Password	String
<code>certificate</code>	Authentication certificate	String
<code>VERIFY</code>	Enable server verification (valid certificate)	Bool
<code>postmime</code>	MIME encoding for POST request	⊙application/json String
<code>acceptmime</code>	MIME encoding for GET request	⊙application/json String
<code>timeout</code>	Timeout interval	⊙5.0 Double (F64)
<code>BLOCKING</code>	Wait for the operation to finish	Bool
<code>nmax</code>	Allocated size of string	↓0 ↑65520 Long (I32)
<code>postmax</code>	Allocated memory for POST request data	↓128 ↑65520 ⊙4096 Long (I32)
<code>datamax</code>	Allocated memory for HTTP response	↓128 ↑10000000 ⊙64000 Long (I32)

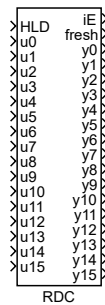
Output

<code>data</code>	Response data	String
<code>BUSY</code>	Sending HTTP request	Bool
<code>DONE</code>	HTTP request processed	Bool
<code>ERROR</code>	Error indicator	Bool
<code>errId</code>	Error code	Error
<code>hterror</code>	HTTP response	Long (I32)

RDC – Remote data connection

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **RDC** block is a special input-output block. The values are passed between two blocks with the same number, but on different computers (or between two Simulinks or between Simulink and the **REXYGEN** system). Values are passed using the UDP/IP protocol. Communication works on all local LAN networks and on Internet links. The algorithm performs the following operations at 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).
- If the data is valid, all outputs **y0** to **y15** are set to the values contained in the packet received.

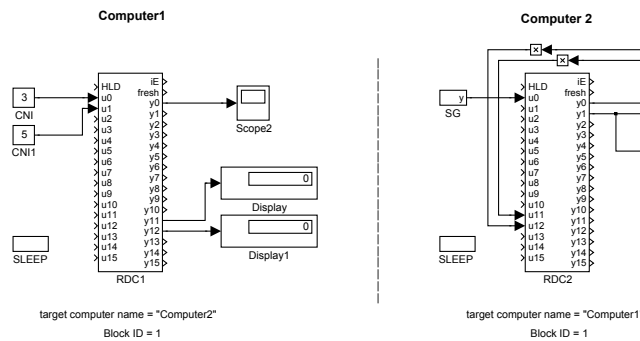
- The **fresh** output is updated. In case of error, the error code is displayed by the **err** output.

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 with it its **invoke** number.

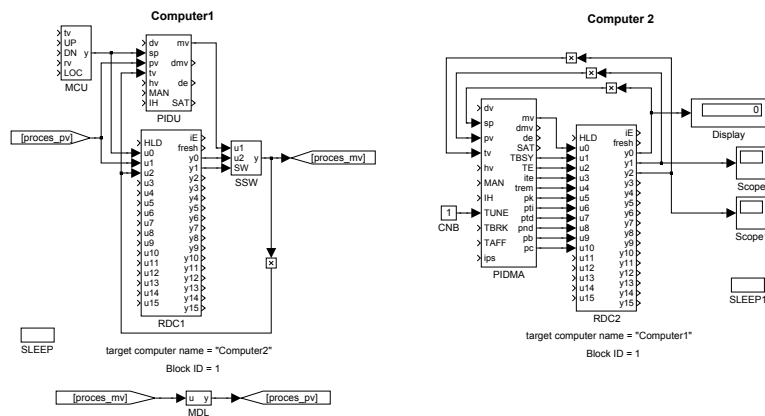
Note: All **RDC** blocks in the same application must have the same **local port** and the count of **RDC** blocks is limited to 64 for implementation reasons. If there are two applications using the **RDC** block running on the same machine, then each of them must use a different **local port**.

Examples

The constants 3 and 5 are sent from **Computer1** to **Computer2**, where they appear at the **y0** and **y1** outputs of the **RDC2** block. The constants are then summed and multiplied and sent back to **Computer1** via the **u11** and **u12** inputs of the **RDC2** block. The displays connected to the **y11** and **y12** outputs of the **RDC1** block show the results of mathematical operations $3 + 5$ and $(3 + 5) * 5$. The signal from the **SG** generator running on **Computer2** is transmitted to the **y0** output of the **RDC1** block, where it can be easily displayed.



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.



This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
u0..u15	Signal to send to the remote RDC block	Double (F64)

Parameter

target	Remote computer name/IP address	String
rport	Remote port number	⊙1288 Word (U16)
lport	Local port number	⊙1288 Word (U16)
id	Block ID	↓1 ↑32767 ⊙1 Long (I32)
period	Minimum communication period [s]	Double (F64)

Output

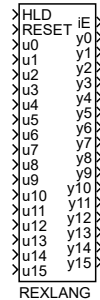
iE	Error code	Long (I32)
	0 No error	
	-1 Maximum number of blocks exceeded (>64)	
	-2 Local ports mismatch	
	-3 Error opening socket	
	-4 Error assigning local port	
	-5 Error setting the so-called non-blocking socket mode	
	-6 Error resolving IP address	
	-10 ... Error initializing the socket library	
	1 Initialization successful, no data received	
	2 Packet consistency error or service/application conflict	
	4 Error receiving packet	
	8 Error sending packet	
fresh	Elapsed time since the last received packet [s]	Double (F64)

y0..y15	Signal received from remote RDC block	Double (F64)
---------	---------------------------------------	--------------

REXLANG – User programmable block

Block Symbol

Licence: [REXLANG](#)



Function Description

The REXYGEN system includes a variety of standard function blocks to meet the common requirements of control applications. However, specific scenarios might require or benefit from a custom-defined function. For such cases, the **REXLANG** block is ideally suited. This block facilitates the creation of user-defined algorithms, with the method of coding specified by the **srctype** parameter. Available options include:

- **1: C-like** – This scripting language is similar to C (or Java). The code is stored in a text file with the extension **.c**. The language follows the C-like syntax below.
- **2: STL** – The code is saved in a text file with the extension **.stl**. The code follows the **IEC61131-3** standard, which is implemented with some restrictions similar to those in a C-like script, such as no structures and limited data types (INT, REAL, and STRING). Variables for function blocks are declared as global variables in **VAR_INPUT**, outputs as **VAR_OUTPUT**, and parameters as **VAR_PARAMETER**. It includes standard functions as specified, with system and communication functions identical to those in the C-like option.
- **3: RLB** – A binary file generated from the compilation of C-like or STL scripts. This option is suitable for users who prefer not to disclose their block's source code.
- **4: ILL** – A text file containing mnemocodes akin to assembler instructions. This option is currently unsupported.

Scripting Language (C-like)

The scripting language employed in **REXLANG** is akin to C, but there are notable differences and limitations:

- Supported data types include `double`, `long` and `string`. Types like `int`, `short`, and `bool` are internally converted to `long`. The `float` type is converted to `double`. The `typedef` type is not available.
- Pointers and structures are not implemented. Arrays can be defined and indexing is possible using the `[]` operator. Inputs, outputs, and parameters of the block cannot be arrays.
- The `' , '` operator is not implemented.
- The preprocessor supports commands like `#include`, `#define`, `#ifdef .. [#else ..] #endif`, and `#ifndef .. [#else ..] #endif`. However, `#pragma` and `#if .. [#else ..] #endif` are not supported.
- While standard ANSI C libraries are not implemented, many mathematical functions from `math.h` and other functions are available (detailed below).
- Keywords `input`, `output`, and `parameter` are used to reference the `REXLANG` block's inputs, outputs, and parameters. System functions for execution control and diagnostics are implemented.
- The `main()` function is executed periodically during runtime. Other functions like `init()` (executed once at startup), `exit()` (executed upon control algorithm stop), and `parchange()` (executed upon parameter changes) are also available.
- Functions and procedures without parameters must declare `void` explicitly.
- Overloading of identifiers is not allowed, i.e., keywords and built-in functions cannot share names with identifiers. Local and global variables must have distinct names.
- User-defined return values from `main()`, `init()`, and `exit()` are written to the `iE` output. Values < -99 will stop algorithm execution (requiring a `RESET` input for further execution). Return values are categorized as follows:

<code>iE >= 0</code>	indicates no errors.
<code>0 > iE >= -99</code>	signifies a warning, without affecting function block execution.
<code>iE < -99</code>	implies an error, halting the function block execution.

Scripting Language Syntax

The syntax of the scripting language is rooted in C, but with some modifications:

- `<type> input(<input number>) <variable name>;` for input variables.
- `<type> output(<output number>) <variable name>;` for output variables.
- `<type> parameter(<parameter number>) <variable name>;` for parameter variables.

The `input` and `parameter` variables are read-only, while `output` variables can be read from and written to.

Example:

```
double input(1) input_signal; // Declaration of a variable - type double,
    which corresponds with the u1 input of the block.
long output(2) output_signal; // Declaration of a variable - type long,
    which corresponds with the y2 output of the block.

input_signal = 3; // not allowed, inputs are read-only
if (input_signal > 1) output_signal = 3 + input_signal; // correct
```

Available Functions

The scripting language encompasses a broad spectrum of functions, including those for mathematical calculations, vector operations, string handling, and system-level commands. The functions are categorized and described in detail as follows:

- **Mathematical Functions** (aligned with ANSI C's `math.h` library):
This category includes functions like `atan`, `sin`, `cos`, `exp`, `log`, `sqrt`, `tan`, `asin`, `acos`, `fabs`, `fmod`, `sinh`, `cosh`, `tanh`, `pow`, `atan2`, `ceil`, `floor`, and `abs`. Note that `abs` is specifically designed for integer values, while the rest operate with `double` type variables. The `fabs` function is used to calculate the absolute value of a decimal number.
- **Vector Functions** (not part of ANSI C):
This set includes specialized functions for vector manipulation. Vectors are represented as arrays of values. Examples of array initializations can be found at the end of the block description. Functions in this category include:

```
double max([n,] val1, ..., valn)
    Returns the maximum value among the provided elements. The first parameter, indicating the number of items, is optional.

double max(n, vec)
    Finds the maximum value in the vec vector.

double min([n,] val1, ..., valn)
    Similar to max, but returns the minimum value.

double min(n, vec)
    Finds the minimum value in the vec vector.

double poly([n,] x, an, ..., a1, a0)
    Calculates the value of a polynomial  $y = a_n * x^n + \dots + a_1 * x + a_0$ . The first parameter is optional.

double poly(n, x, vec)
    Computes the polynomial value  $y = \text{vec}[n] * x^n + \dots + \text{vec}[1] * x + \text{vec}[0]$ .

double scal(n, vec1, vec2)
    Calculates the scalar product of two vectors:  $y = \text{vec1}[0] * \text{vec2}[0] + \dots + \text{vec1}[n-1] * \text{vec2}[n-1]$ .
```

```
double scal(n, vec1, vec2, skip1, skip2)
    A variant of scal that allows skipping elements:  $y = \text{vec1}[0] * \text{vec2}[0] + \text{vec1}[\text{skip1}] * \text{vec2}[\text{skip2}] + \dots + \text{vec1}[(n-1) * \text{skip1}] * \text{vec2}[(n-1) * \text{skip2}]$ . 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)
    Computes the convolution of two vectors:  $y = \text{vec1}[0] * \text{vec2}[n-1] + \text{vec1}[1] * \text{vec2}[n-2] + \dots + \text{vec1}[n-1] * \text{vec2}[0]$ .
```

```
double sum(n, vec)
    Sums the elements of a vector:  $y = \text{vec}[0] + \text{vec}[1] + \dots + \text{vec}[n-1]$ .
```

```
double sum([n,] val1, ..., valn)
    Sums the provided values. The count parameter is optional.
```

```
[] subarray(idx, vec)
    Returns a subarray of vec starting from index idx. 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).
```

Note: The functions `max`, `min`, `poly`, `scal`, `conv`, and `sum` are overloaded, meaning they have multiple variants based on the parameters. Parameters are strictly type-checked, requiring casting for non-double types. For instance:

```
double res = max(dVal, (double)iVal, 1.0, (double)2);
```

casts `iVal` to double. If a parameter is not a double, an error stating "no function of this prototype was found" is reported.

- **String Functions** (This section covers functions analogous to those found in ANSI C's `string.h` library, providing a range of operations for string manipulation and analysis:)

```
string strstr(str, idx, len)
    Extracts a substring from str, starting at index idx and spanning len characters.
```

```
long strlen(str)
    Returns the length of the string str, measured in characters.
```

```
long strfind(str, substr[, offset])
    Finds the first occurrence of substr within str and returns its position. The search starts from the character with index offset (if not specified, then from the beginning). The parameter substr can also be a character.
```

```
long strrfind(str, substr)
    Identifies the last occurrence of substr within str and provides its index.
```

strreplace(str, pattern, substr)
 Replaces all instances of **pattern** in **str** with **substr**. This modification is done in-place, directly altering **str**.

strupr(str)
 Converts all characters in **str** to uppercase.

strlwr(str)
 Transforms **str** to all lowercase characters.

strtrim(str)
 Removes leading and trailing whitespace from **str**.

long str2long(str[, default])
 Converts **str** to an integer. If conversion fails, the optional second parameter (default value) is returned, or 0 if not provided.

double str2double(str[, default])
 Turns **str** into a decimal number. Similar to **str2long**, it returns an optional default value or 0 on failure.

string long2str(num[, radix])
 Converts an integer **num** to a string, with an optional **radix** parameter specifying the base (default is 10). The output string does not indicate the numeral system (no prefixes like 0x for hexadecimal).

string double2str(num)
 Converts a decimal number **num** to a string representation.

strcpy(dest, src)
 Copies the content of **src** into **dest**. For ANSI C compatibility, **dest = src** achieves the same result.

strcat(dest, src)
 Appends **src** to the end of **dest**. As in ANSI C, **dest = dest + src** performs the same operation.

strcmp(str1, str2)
 Compares two strings **str1** and **str2**. The construction **str1 == str2** can be used for the same purpose, providing ANSI C compatibility.

float2buf(buf, x[,endian])
 Converts the real number **x** into an array **buf** of four elements, each representing an octet of the number in IEEE 754 single precision format (known as float). The function is useful for filling communication buffers. Optional 3rd parameter has the following meaning:
 0 processor native endian (default),
 1 little endian,
 2 big endian.

double2buf(buf, x[,endian])
 Similar to **float2buf**, but for double precision, storing eight elements (double type).

double buf2float(buf[, endian])
 The inverse of **float2buf**.

`double buf2double(buf[, endian])`

The inverse of `double2buf`.

`long RegExp(str, regexp, capture[])`

Matches `str` against the regular expression `regexp`, storing captured groups in `capture`. Returns the number of captures or a negative error code. The regular expression syntax includes standard constructs like:

`(?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.

Example:

```
RegExp("48,string1,string2","^((\d+),([^\,]+),",capture);
```

```
Result: capture=["48,string1","48","string1"]
```

`long ParseJson(json, cnt, names[], values[])`

This function processes a `json` string, extracting the values of specified objects. The `names` array should list the properties of interest (access subitems with `.` and array indices with `[]`, for instance, `"cars[1].model"`). Corresponding values are then populated in the `values` array. The `cnt` parameter determines the number of objects to be parsed, which should match the length of both the `names` and `values` arrays. This function returns the total number of successfully parsed values, or a negative value if an error occurs during parsing.

Note: String variables are declared as in ANSI C (`char <variable name>[<max`

`chars>];`). For function arguments, use `char <variable name>[]` or `string <variable name>`.

- **System functions** (not part of ANSI C)

Archive(arc, type, id, lvl_cnt, value)

This function archives a value into the system's archival subsystem. **arc** serves as a bitmask to specify the target archives (e.g., for archives 3 and 5, set **arc** = 20, which is 10100 in binary or 20 in decimal). Archive numbering starts from 1, with a maximum of 15 archives (archive 0 is reserved for internal system logs). The **type** parameter defines the data type, with options:

- 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 represents a unique archive item ID, **lvl_cnt** denotes an alarm level or the number of elements for Group types, and **value** is the data to be archived.

Trace(id, val)

Displays both the **id** and **val** values, mainly used for debugging purposes. **id** is a user-defined constant ranging from 0 to 9999 for easy message identification. **val** can be any data type, including strings. Output appears 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 *Target→Diagnostic messages* and tick the *Information* checkbox in the

Function block messages box. Logging has to be also enabled for the particular block by ticking the *Enable logging* checkbox in the *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 *System log*.

TraceError(id, val) TraceWarning(id, val) TraceVerbose(id, val)

Similar to **Trace**, these commands categorize the output into Error, Warning, or Verbose logging groups. *Error* messages are always logged. For *Warning* and *Verbose* messages, enable the respective message groups in the *Diagnostic messages* menu.

Suspend(sec)

Suspends script execution if it exceeds the specified time in **sec** during a sampling period. The script resumes from the suspension point upon the next block execution. Use **Suspend(0)** to pause the script immediately.

double GetPeriod()

Returns the block's sampling period in seconds.

double CurrentTime()

Function provides the current time in an internal format, often used with **ElapsedTime()**.

double ElapsedTime(new_time, old_time)

Calculates elapsed time between **new_time** and **old_time** in seconds. Function **CurrentTime()** is typically used for **new_time**.

double Random()

Generates a pseudo-random number from the $(0, 1)$ interval. The generator is initialized before the **init()** function, ensuring a consistent sequence.

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([n,]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. Supports both binary files (with **double** values) and text files (values on separate lines). The file is identified by **fileid**, and **idx** indicates the index (for binary files) or line number (for text files).

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 `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 `idx` is the line number and starts at zero. Value `idx=-1` means next line (e.g. sequential writing).

`void SaveValue(fileid, idx, value)`

Stores a value in a file, with parameters functioning similarly to `LoadValue`.

`void GetSystemTime(time)`

Returns the system time in UTC (modifiable via OS settings). The `time` parameter must be an array of at least 8 `long` items. 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 Sleep(seconds)`

Halts the block's algorithm (and whole task) for the defined time. Use this block with extreme caution and only if there is no other possibility to achieve the desired behavior 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 GetExtInt(ItemID)`

Returns the value of input/output/parameter of arbitrary block in REXYGEN algorithm. 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 `GETPI` function block. If the value cannot be obtained (e.g. invalid or non-existing `ItemID`, data type conflict, etc.), the `REXLANG` block issues an error and must be reset.

`long GetExtLong(ItemID)`

See `GetExtInt(ItemID)`.

`double GetExtReal(ItemID)`

Similar to `GetExtInt(ItemID)` but for decimal numbers.

`double GetExtDouble(ItemID)`

See `GetExtReal(ItemID)`.

`string GetExtString(ItemID)`

Similar to `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)`

Executes a single step of any block within the `REXYGEN` algorithm, identified by `ItemID`. The structure of the string parameter `ItemID` is the same as in e.g. the `sc` parameter of the [GETPI](#) function block. It's recommended to call only halted blocks², which should be in the same task as the `REXLANG` block. The function returns result code of the calling block (see `REXYGEN` error codes).

`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.

`long GetInArrMax(input)`

Returns the maximum (allocated) size 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.

`long GetInArrLong(input, row, col)`

Similar to `GetInArrDouble(...)`, but the value is of type long.

`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(72, "/dev/mem", <physical address>, <area size>)`.

²set checkbox `Halt` on the property page `Runtime` in the parameters dialog of the block

`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 suite of functions facilitates communication over various channels including TCP/IP, UDP/IP, serial lines (RS-232 or RS-485), SPI bus, and I2C bus. Below is a concise list of these functions. For comprehensive details, refer to the example projects in the REXYGEN system.

`long OpenFile(string filename)`

This function opens a file and returns an identification number (handle) of the file. If the function returns a negative value, the file opening was unsuccessful.

`long OpenCom(string comname, long baudrate, long parity)`

Opens a serial line and returns its handle. From REXYGEN version 3.0 onwards, virtual ports can be specified as `comname`. For detailed information about virtual ports, see the [UART](#) block description. If a negative value is returned, the opening failed. The parity setting options are 0 for none, 1 for odd, and 2 for even.

`long OpenUDP(string localname, long lclPort, string remotename, long remPort)`

Opens a UDP socket and returns its handle. If the function returns a negative value, the socket opening was unsuccessful. The function can open either an IPv4 or IPv6 socket based on the `remotename`, `localname`, or operating system settings if a DNS name is used. Optional settings include an empty `localname` (any interface), an empty `remotename` or 0 for `remPort` (unused), and 0 for `lclPort` (assigned by the UDP/IP stack).

`long OpenTCPSvr(string localname, long lclPort)`

This function opens a TCP socket in server (listening) mode. It returns a handle for the socket, and a negative return value indicates an unsuccessful opening. The function can open either IPv4 or IPv6 sockets depending on `remotename`, `localname`, or operating system settings if a DNS name is used. You can set an empty `localname` to mean any interface.

`long OpenTCPcli(string remotename, long remPort)`

Opens a TCP socket in client mode and returns its handle. A negative return value indicates failure to open the socket. The function opens either an IPv4 or IPv6 socket based on `remotename`, `localname`, or operating system settings if a DNS name is used. Note that this function does not wait for the connection to be established, which might take a few milliseconds on a local network or a few seconds for remote locations. If `Write()` or `Read()` are called before the connection is established, an error code -307 (file open error) is returned.

`long OpenI2C(string devicename)`

Opens the I2C bus and returns its handle. If the function returns a negative value, the opening was not successful.

```

long OpenSPI(string devicename)
    Opens the SPI bus and returns its handle. A negative return value indicates
    an unsuccessful opening.

long OpenDevice(string filename)
    Similar to OpenFile(), but the functions Write() and Read() are non-
    blocking. If data cannot be read or written, the function immediately
    returns a -1 error code.

long OpenMemory(string devicename, long baseaddr, long size)
    Maps physical memory and returns the associated handle. If a negative
    value is returned, the operation was unsuccessful.

long OpenSHM(string devicename, long deviceid, long size, long flags)
    Maps shared memory (Linux only, using ftok() and shmget()) and re-
    turns the associated handle. The first and second parameters serve to
    identify the memory area (they must be the same for all cooperating enti-
    ties). The size parameter specifies the size of the shared memory area in
    bytes. The flags parameter represents standard Linux flags and permis-
    sions (if set to 0, which is the default value, the following rights are set:
    create the area if it does not exist, and allow everyone to read and write).

void Close(long handle)
    Closes a socket, serial line, file, or any device opened with the Open...
    functions.

void SetOptions(long handle, long params[])
    Configures the parameters of a socket or serial line. The array size must
    be at least:
        22 for serial line (on Windows parameters for SetCommState()
            and SetCommTimeouts() in following order: BaudRate,
            fParity, Parity, StopBits, ByteSize, fDtrControl, fRtsCon-
            trol, fAbortOnError, fBinary, fErrorChar, fNull, fDsrSen-
            sitivity, fInX, fOutX, fOutxCtsFlow, fOutxDsrFlow, FTX-
            ContinueOnXoff, ReadIntervalTimeout, ReadTotalTimeout-
            Constant, ReadTotalTimeoutMultiplier, WriteTotalTime-
            outConstant, WriteTotalTimeoutMultiplier; linux use differ-
            ent function, but meaning of the parameters is as same as
            possible),
        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 ad-
            dress flag, 3rd item is Packet Error Checking flag, 4th item
            is number of retries, 5th item is timeout),
    other handle types are not supported

void GetOptions(long handle, long params[])
    Reads the parameters of a socket or serial line and stores them in the

```

params array. The array size must accommodate the specific device requirements (see **SetOptions**).

long Accept(long hListen)

Accepts a connection on a listening socket, returning a communication socket handle or an error code.

long Read(long handle, long buffer[], long count[, offset])

Receives data from a serial line or socket, returning the number of bytes read or an error code. The **count** parameter defines the maximum number of bytes to read. Each byte of incoming data is put to the **buffer** array of type **long** in the corresponding order. The function has one more (optional) **offset** parameter that can be used when reading data from memory when the handle is created using the **OpenSHM()** or **OpenMemory()**.

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[, offset])

Sends data over 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. The function has one more (optional) **offset** parameter that can be used to write data to memory when the handle is created using the **OpenSHM()** or **OpenMemory()**.

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)

Reads a line from a (text) file, serial line, or socket, storing the characters in **data** up to its allocated size. The function returns the actual size of the line or an error code.

long DeleteFile(string filename)
 Deletes a file, returning 0 on success or a negative error code on failure.

long RenameFile(string filename, string newfilename)
 Renames a file, returning 0 on success or a negative error code on failure.

bool ExistFile(string filename)
 Checks if a file or device exists (can be opened for reading), returning true or false.

long I2C(long handle, long addr, long bufW[], long cntW, long bufR[], long cntR)
 Handles communication over the I2C bus, particularly on Linux systems with I2C capabilities (e.g., Raspberry Pi). The function sends and receives data to/from a slave device using **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)
 Executes a transaction over the SPI bus, particularly on Linux systems with SPI capabilities. 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)
 Sets the position for Read/Write commands. Parameter **mode** means:

- 1 offset from begin of the file,
- 2 offset from current position,
- 3 offset from end of the file.

long Recv(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)
 Computes a 16-bit Cyclic Redundancy Code (CRC), commonly used as a checksum or hash in various communication protocols.

data A byte array (represented by a long array) or a string for which the hash is computed.

length The number of bytes in the input array or text. Use -1 to process the entire string.

init The initial vector for the CRC computation.
poly The control polynomial used in the computation.
flags Configuration flags

- 1 Reverses the bit order in both the input bytes and the resulting CRC.
- 2 The resulting CRC is XORed with 0xFFFF.
- 4 If **data** is a long array, all 4 bytes in a long are processed (LSB first).
- 8 Similar to flag 4, but processes MSB first.

offset The index of the first byte to be processed in the data array (usually 0).

Note: Similar functions exist for computing 32-bit and 8-bit CRCs: `long crc32(data, length, init, poly, flags, offset)` and `long crc8(data, length, init, poly, flags, offset)`. The initial vector, control polynomial, and flags for various protocols can be found at <https://crccalc.com/>

Examples:

- MODBUS: `crc16("123456789", -1, 0xFFFF, 0x8005, 1, 0)`
- DECT-X: `crc16("123456789", -1, 0, 0x0589, 0, 0)`

Additional Note: The `crc8(...)` and `crc32(...)` functions also exist, supporting 8-bit and 32-bit CRC calculations with the same parameter structure.

Remarks

- Data types of inputs `u0..u15`, outputs `y0..y15`, and parameters `p0..p15` are determined during the source code compilation.
- Error codes `< -99` require a **RESET** input for restarting the **REXLANG** block after addressing the cause of the error.
- ATTENTION!!! It is possible to read inputs in the `init()` function, but since other blocks usually do not set outputs in the init phase, there will always be 0. Outputs can be set, but usually this is not done.
- The **srcname** parameter can be specified with an absolute path. Otherwise, the file is searched for in the current directory and the specified directories (see the **LibraryPath** parameter of the **PARAM** block).
- Vector function parameters are primarily of type **double**, with the exception of the **n** parameter, which is 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`³ 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.

- It's crucial to remember that arrays in the scripting language behave similarly to arrays in C: indexing begins at 0 and there is no automatic boundary checking. For instance, if you declare `double vec[10], x;`, the array `vec` will have elements indexed from 0 to 9. Accessing `vec[10]` does not trigger a syntax or runtime error, but the returned value is undefined since it's beyond the array bounds. Additionally, assigning a value to `vec[11]` (e.g., `vec[11] = x;`) can be particularly dangerous, as it may inadvertently overwrite adjacent memory locations. This could lead to unpredictable behavior or cause the program to crash.
- During the compilation process, if there are syntax errors, the compiler reports a **parser error** along with the line number where the error occurred. These reports specifically indicate issues with syntax. However, if the syntax appears correct and an error is still reported, it's advisable to check for conflicts involving identifiers, keywords, or function names, as these can also cause errors not immediately evident as syntax-related.
- 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 the results of temporary computations are stored in the stack, which includes:
 - Global and local **static** variables are permanently located at the stack's base.
 - Return addresses for function calls.
 - Function parameters.
 - Variables local to functions.
 - The return value of a function.
 - Temporary computational results. For example, in the expression `a = b + c;`, `b` and `c` are first pushed onto the stack. Their sum is then calculated, the operands are popped off the stack, and the result is pushed onto the stack.

³The 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.

Simple variables such as `long` or `double` occupy one stack slot each. For arrays, the total occupied size matters, irrespective of the element type.

- When arrays are passed to functions, they are referenced rather than copied. This means only one stack slot is used for the reference, and the function operates directly on the original array.
- If the allocated stack size is insufficient (less than the space needed for global variables plus 10), it is automatically doubled, with an additional 100 slots for computational needs, function parameters, and local variables, especially when few global variables are defined.
- With basic debug level, various checks are conducted during script execution. These include the initialization of read values and array index boundary checks. Additionally, a few uninitialized values are inserted at both the start and end of each declared array for boundary checking, and NOP instructions with source file line numbers are added to the `*.ill` file.
- At full debug level, an additional check for invalid data range accesses (such as stack overflows) is enabled.
- In this context, an 'instruction' refers to a processor-independent mnemonic code. These codes are stored in the `*.ill` file.
- The `OpenCom()` function sets binary non-blocking mode without timeouts, 8 data bits, 1 stop bit, no parity, 19200 Baud. Optionally, the `baudrate` and `parity` parameters can be adjusted in the `OpenCom()` function.
- Accessing text files is significantly slower than binary files. However, text files offer the advantage of being viewable and editable without specialized software.
- The block does not automatically invoke the `parchange()` function. This function must be manually called within the `init()` function if needed.
- The `OpenFile()` function opens files in the data directory of the REXYGEN system (i.e., in Linux by default in `\rex\data`, on Windows `C:\ProgramData\REX Controls\REX_<version>\RexCore`). Subdirectories are allowed, but `..` is not permitted. Links are followed.

Debugging the code

Use the `Trace` command mentioned above.

Examples of array initialization

Below are several examples of array initialization in a C-like language. Arrays can be used in vector functions.

```

double vec0[20]; // Uninitialized array/vector. Global variables
                  including arrays are automatically initialized to 0.
double vec1[20] = {0, 1.1, 2.2, 3, 4}; // Partially initialized array/
vector.
double vec2[] = {10, 11.1, 12.2, sqrt(2), 14}; // Initialized array with
automatically determined size (5 elements).

char str0[30]; // Uninitialized text string with a capacity for 29
characters plus a null terminator.
char str1[30] = "hello"; // Initialized text string.

int mat0[5][10]; // Uninitialized matrix with 5 rows and 10 columns.
int mat1[5][6] = {{100, 101, 102}, {110, 111, 112, 113}, {120, 121, 122,
123, 124}}; // Partially initialized matrix.
int mat2[5][6] = {{200, 201, 102}, {210, 211, 212, 213}, {220, 221, 222,
223, 224, 225}, {230}, {240,241}};

char strV0[4][30]; // Uninitialized vector of four text strings.
char strV1[4][30] = {"name", "surname", "address"}; // Partially
initialized vector of text strings.

```

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

```

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
RESET	Block reset	Bool
u0..u15	Input signal	Any

Parameter

srcname	Source file name	Ⓒsrcfile.c	String
---------	------------------	------------	--------

srctype	Coding of source file	⊙1	Long (I32)
	1 C-like		
	2 STL		
	3 RLB		
	4 ILL		
debug	Debug level	⊙3	Long (I32)
	1 No check		
	2 Basic check		
	3 Full check		
stack	Stack size in bytes (0=automatic)		Long (I32)
strs	Total size of buffer for strings (number of characters, 0=automatic)		Long (I32)
p0..p15	Parameter		Any

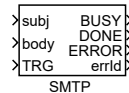
Output

iE	Error code	Error
	i REXYGEN error code	
y0..y15	Output signal	Any

SMTP – Block for sending e-mail alerts via SMTP

Block Symbol

Licence: [ADVANCED](#)



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 content of the message is defined by the **subj** and **body** inputs. The **from** and **to** parameters specify the sender and recipient of the message. The message is sent when a rising edge (**off**→**on**) is detected at the **TRG** input. The **BUSY** output is set to **on** until the request is completed, which is signaled by the **DONE** output. In case of an error, the **ERROR** output is set to **on**. The **errId** output carries the last error identified by the 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 may be multiple message recipients. In such a case, individual e-mail addresses must be separated by a comma and no space character should 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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

subj	Subject of the e-mail message	String
body	Body of the e-mail message	String
TRG	Trigger of the selected action	Bool

Parameter

server	SMTP server address	String
to	E-mail of the recipient	String
from	E-mail of the sender	String
tls	Encryption method	⊙1 Long (I32)
	1 None	
	2 StartTLS	
	3 TLS	
user	User name	String
password	Password	String
domain	domain	String
auth	Authentication method	⊙1 Long (I32)
	1 Login	
	2 Plain	
certificate	Authentication certificate	String
VERIFY	Enable server verification (valid certificate)	Bool
timeout	Timeout interval	Double (F64)
BLOCKING	Wait for the operation to finish	Bool

Output

BUSY	Sending e-mail	Bool
DONE	E-mail has been sent	Bool
ERROR	Error indicator	Bool
errId	Error code	Error

STEAM – Steam and water properties

Block Symbol

Licence: [STANDARD](#)



Function Description

The **STEAM** blok can be used to calculate various thermodynamic properties of water and steam, such as enthalpy, entropy, saturation temperature and pressure, density, viscosity, and others. The inputs of the block are typically temperature and pressure, but calculations can be done in the opposite direction, for example, determining the temperature from the enthalpy. The calculations are based on the international standard IAPWS IF-97, details of which can be found at <http://www.iapws.org/relguide/IF97-Rev.pdf>.

The units for temperature can be set by the **tunit** parameter, for pressure by the **punit** parameter. Energy units are in kilojoules [kJ], for example, enthalpy is expressed in kJ/kg, heat capacity in kJ/kg/K, which corresponds to the definitions in IF-97. Other quantities are given in SI units, such as density in kg/m³.

The block function has a name in the format `<output property>_<1st input property><2nd input property>`, where the properties include:

- **T** - Temperature
- **p** - Pressure
- **h** - Enthalpy [kJ/kg]
- **v** - Specific volume [m³/kg]
- **rho** - Density [kg/m³]
- **s** - Specific entropy
- **u** - Specific internal energy [kJ/kg]
- **Cp** - Specific isobaric heat capacity [kJ/kg/K]
- **Cv** - Specific isochoric heat capacity [kJ/kg/K]
- **w** - Speed of sound [m/s]
- **my** - Viscosity
- **tc** - Thermal Conductivity

- **st** - Surface Tension
- **x** - Vapour fraction
- **vx** - Vapour Volume Fraction

The output property can have attribute:

- **sat** - Saturated value, i.e. for situation when water (liquid) is changed into steam (vapour)
- **V** - Steam (vapour) for saturated conditions
- **L** - Water (liquid) for saturated conditions

Examples

- **h_pT** output is enthalpy for given pressure (1st input) and temperature (2nd input); for example, for pressure 100 kPa and temperature 120 °C, the enthalpy is 2716 kJ/kg.
- **Tsat_p** saturated temperature (i.e. boiling temperature) for given pressure (1st input); for example, for pressure 100 kPa, the boiling temperature is 100 °C.
- **hL_p** enthalpy of (liquid) water for saturated conditions given by pressure (1st input); for example, at 100 kPa (and temperature 100 °C, to be at saturated conditions), the medium can contain any ratio of water and steam, the function (block output) will be the enthalpy for the situation when the medium is water (without steam), i.e. 417 kJ/kg.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

func	Function type	⊙1 Long (I32)
	1 Tsat_p	
	2 T_ph	
	3 T_ps	
	4 T_hs	
	5 psat_T	
	6 p_hs	
	7 p_hrho	
	8 hV_p	
	9 hL_p	
	10 hV_T	
	11 hL_T	
	12 h_pT	
	13 h_ps	
	14 h_px	
	15 h_prho	
	16 h_Tx	
	17 vV_p	
	18 vL_p	
	19 vV_T	
	20 vL_T	
	21 v_pT	
	22 v_ph	
	23 v_ps	
	24 sV_p	
	25 sL_p	
	26 sV_T	
	27 sL_T	
	28 s_pT	
	29 s_ph	
	30 CpV_p	
	31 CpL_p	
	32 CpV_T	
	33 CpL_T	
	34 Cp_pT	
	35 Cp_ph	
	36 Cp_ps	
	37 CvV_p	
	38 CvL_p	
	39 CvV_T	
	40 CvL_T	
	41 Cv_pT	
	42 Cv_ph	
	43 Cv_ps	
	44 wV_p	
	45 wL_p	
	46 wV_T	
	47 wL_T	
	48 w_pT	
	49 w_ph	
	50 w_ps	
	51 my_pT	
	52 my_ph	
	53 my_ps	
	54 tcL_p	
	55 tcV_p	
	56 tcL_T	
	57 tcV_T	

punit	Used unit for pressure	⊙1	Long (I32)
	1 MPa		
	2 bar		
	3 kPa		
tunit	Used unit for temperature	⊙1	Long (I32)
	1 K		
	2 °C		

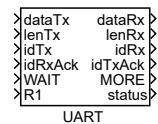
Output

y	Analog output of the block	Double (F64)
E	Error indicator	Bool

UART – UART communication block

Block Symbol

Licence: [STANDARD](#)



Function Description

The **UART** block allows you to read and write data via the Universal Asynchronous Receiver-Transmitter. The **port** parameter specifies device name. There it is possible to use two name types:

- the address of the physical device – Usually `/dev/ttyS*` for Linux target or `COM*` for Windows. Replace `"*"` symbol according to the chosen serial port!
- the virtual address – **REXYGEN** enables the creation of a virtual UART with which you can communicate inside **REXYGEN** with other blocks supporting UART such as [REXLANG](#), [PYTHON](#), another **UART** block or **Modbus driver**. On Linux devices, the virtual port is marked with the prefix `pty:` (pseudo terminal) and it is possible to connect to it from another application running on the device. On Windows devices, it is possible to use the prefix `vcom`, which enables communication within **REXYGEN**. Virtual port examples: `pty:/tmp/vslave`, `vcom:vmaster`.

UART communication has several general properties that are set using parameters such as **baudrate**, **parity**, **databits** and **stopbits**. Each packet that is received or transmitted is assigned a unique ID. The ID of the next packet is always one higher than the ID of the previous packet. Once the maximum ID is reached, the next ID assigned will be 0. The maximum ID value is determined by the **maxId** parameter. Data is sent with the rising edge of the **idTx** input. If there is still data in the buffer to be sent, the **WAIT** output is set to **on**.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

			Reference
dataTx	Vector reference to transmitted data		
lenTx	Transmitted data length (0 = whole vector)	↓0	Long (I32)
idTx	ID of the transmitted data packet	↓0	Long (I32)
idRxAck	ID of the last processed received data packet	↓0	Long (I32)
WAIT	Transmission suspended flag (data is buffered)		Bool
R1	Block reset		Bool

dataRx	Vector reference to received data		Reference
lenRx	Received data length	↓0	Long (I32)
idRx	ID of the received data packet	↓0	Long (I32)
idTxAck	ID of the last processed transmitted data packet	↓0	Long (I32)
MORE	Additional data in the receive buffer flag		Bool
status	Internal status indicator		Long (I32)
	0 No Error		
	-1 Failed to open port		
	1 Transmit buffer overflow		
	2 Transmit data error		
	256 ... Received data error		

Parameter

port	Communication device name		String
baudrate	Baudrate [bit/s] (0 = not set)	↓0 ↑4000000	Long (I32)
parity	Parity		Long (I32)
	0 Not set		
	1 NoParity		
	2 OddParity		
	3 EvenParity		
databits	Number of data bits (0 = not set)	↓0 ↑3	Long (I32)
stopbits	Number of stop bits (0 = not set)	↓0 ↑2	Long (I32)
maxId	Max value used as ID of a packet	↓2 ↑10000000 ⊕4	Long (I32)
maxLen	Maximum length of the received data	↓1 ↑10000000 ⊕64	Long (I32)
nmax	Allocated size of array	↓8 ↑10000000 ⊕256	Long (I32)

Chapter 17

LANG – Language blocks

Contents

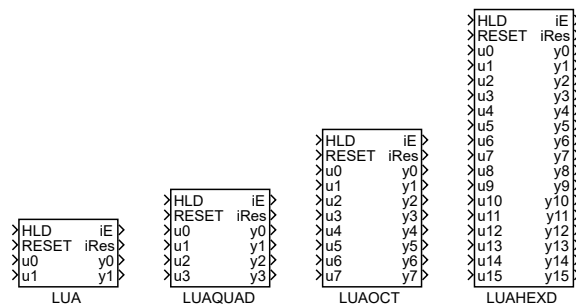
LUA, LUAQUAD, LUAOCT, LUAHEXD – User programmable blocks in Lua	604
PYTHON – User programmable block in Python	608

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. For these purposes, the blocks from the LANG library, or the [REXLANG](#) block, can be used.

LUA, LUAQUAD, LUAOCT, LUAHEXD – User programmable blocks in Lua

Block Symbols

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 **LUA** block can be used.

The **LUA** block implements an user-defined algorithm written in a Lua 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 Lua environment.

There are four variants of the **LUA** block with different number of inputs and outputs. The inputs and outputs of the block are indexed from 0 to 15. The inputs and outputs of the block can be of any data type supported by the REXYGEN system.

Scripting language

The scripting language is a standard Lua v5.4 language (see [17]). Every block references a script stored in a *.lua source file or it directly contains the script in a parameter of the block. The script 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.

Additionally, the **compile()** function is executed once during the compilation process of the REXYGEN configuration. This function can be used to modify the table **REX.ctx**

that then after the compilation gets serialized and stored in the configuration and is later deserialized during the initialization of the block on the target device. This way the script can create a shared context between the compilation environment and the target device. Since the script is to some extent executed both on the host and target device, the items `REX.host` and `REX.target` are available to the script to distinguish between the two environments.

Scripts on the target device

Lua interpreter can load a script from specified file or directly from a string. If the parameter `embedded` is set to `on` the script referenced by the block in the parameter `src` gets embedded in the REXYGEN configuration during compilation process and will be loaded during initialization of the block once the configuration is downloaded and executed on the target device. Otherwise the script is loaded from a file on the target device that is again specified by the parameter `src`.

Data exchange API

For the purpose of data exchange between a Lua interpreter and REXYGEN system an object `REX` containing the data exchange API was developed and is injected into every instance of a LUA block interpreter.

I/O objects

The range of input and output objects depends on the used block. The most compact LUA block contains 2 inputs and 2 outputs, LUAQUAD 4 inputs and 4 outputs, LUAOCT 8 inputs and 8 outputs and the largest LUAHEXD 16 inputs and 16 outputs.

`REX.u0` - `REX.u15`

– objects representing block *inputs* in Lua environment

`REX.p0` - `REX.p15`

– objects representing block *parameters* in Lua environment

`REX.y0` - `REX.y15`

– objects representing block *outputs* in Lua environment

Access to values

The I/O objects must have its data type specified during compilation of the REXYGEN configuration. Default data type is `double` but it can be changed using the function `type()`.

Example of setting the data type of the block IOs:

```
REX.u0.type("int")
```

```
REX.y0.type("string")
```

All I/O objects contain a function `v()`. That performs a conversion from REXYGEN data types to Lua data types. The value then can be stored in variables and used in the block algorithm.

Example of reading a value of the block input:

```
local x = REX.u0.v()
```

```
local myU0 = REX.u(0)
local x = myU0.v()
```

Setting the REXYGEN value from the Lua script can be achieved with function **setV()** that performs a conversion from Lua 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.setV(42)
```

```
local myY0 = REX.y(0)
myY0.setV(42)
```

Arrays

Arrays can be manipulated through function **v()** but direct conversions between REXYGEN arrays and Lua tables are not very memory efficient. In addition input objects support direct access to the values of the REXYGEN array connected to the block input using functions **arr()** and **setArr()**.

Example of reading and writing a value of the block input for matrix:

```
local x = REX.u0.arr(0, 0)
REX.u0.setArr(0, 0, 42)
```

External items

The object **REX** contains a function **Item()** that returns a handle to an external REXYGEN item based on a connection string specified in a parameter of the function.

Example of creating a handle to an external item and setting its value:

```
local cns = REX.Item("myproject_task.CNS:scv")
cns.setV("abc")
```

Tracing

The object **REX** contains functions **Trace**, **TraceError**, **TraceWarning**, **TraceVerbose** and **TraceInfo** that can be used to write messages into REXYGEN system log.

Example of logging a message:

```
REX.Trace("abc")
```

Additional features

REX.RexDataPath() – **RexDataPath()** is a function that returns a path to a data folder of the REXYGEN 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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
RESET	Block reset	Bool
u0..u15	Input signal	Any

Parameter

src	Source file name or source	⊙program.lua	String
embedded	Embedding of the script in the REXYGEN executive program	⊙on	Bool
limit	Limit of the Lua interpreter execution per period	↓0	Long (I32)
p0..p15	Parameter		Any

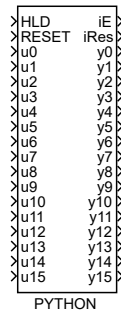
Output

iE	Error code	Error
	i REXYGEN error code	
iRes	Execution result code	Long (I32)
y0..y15	Output signal	Any

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 [18]). 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.

```
from PyRexExt import REX
```

I/O objects

`REX.u0` - `REX.u15`
 – objects representing block *inputs* in Python environment

`REX.p0` - `REX.p15`
 – objects representing block *parameters* in Python environment

`REX.y0` - `REX.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:

```
x = REX.u0.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 = REX.u0[0]
```

Example of writing a value to the block output for multidimensional array:

```
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:

```
cns = REX.Item("myproject_task.CNS:scv")
cns.v = "abc"
```

Tracing

The object `REXYGEN` 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:

```
REX.Trace("abc")
```

Additional features

`REX.RexDataPath` – `RexDataPath` is a string constant that contains a path to a data folder of the REXYGEN 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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
RESET	Block reset	Bool
u0..u15	Input signal	Any

Parameter

<code>srcname</code>	Source file name	⊙ <code>program.py</code>	String
<code>embedded</code>	Embedding of the script in the REXYGEN executive program	⊙ <code>on</code>	Bool
<code>limit</code>	Limit of the Python interpreter execution per period	↓1 ⊙10000	Long (I32)
<code>limitMode</code>	Python interpreter limit mode		Long (I32)
	0 no limit		
	1 instructions per period, continue after interrupt		
	2 instructions per period, start from scratch after interrupt		
	3 milliseconds per period, continue after interrupt		
	4 milliseconds per period, start from scratch after interrupt		
<code>p0..p15</code>	Parameter		Any

Output

<code>iE</code>	Error code		Error
	i REXYGEN error code		
<code>iRes</code>	Execution result code		Long (I32)
<code>DONE</code>	Script execution finished		Bool
<code>y0..y15</code>	Output signal		Any

Chapter 18

DSP – Digital Signal Processing blocks

Contents

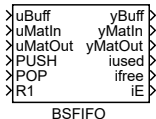
BSFIFO – Binary Structure - Queueing serialize and deserialize	614
BSGET, BSGETOCT – Binary Structure - Get a single value of given type	616
BSGETV, BSGETOCTV – Binary Structure - Get matrix (all values of same given type)	618
BSSET, BSSETOCT – Binary Structure - Set a single value of given type	620
BSSETV, BSSETOCTV – Binary Structure - Set matrix (all values of same given type)	621
FFT – Fast Fourier Transform	622
MOSS – Motion smart sensor	624
PCI – PCI Bus Memory Access	626
PSD – Power Spectral Density	627

The DSP library is tailored for advanced digital signal processing. It includes blocks like [FFT](#) for Fast Fourier Transform operations and [PSD](#) for Power Spectral Density analysis. The library also features [BSFIFO](#), [BSGET](#), [BSGETV](#), [BSSET](#), and [BSSETV](#) for buffer storage and retrieval, enabling efficient data handling in signal processing tasks. In addition, the library contains a [MOSS](#) block - an advanced filter for incremental sensors. This collection of blocks is essential for sophisticated signal analysis and manipulation in digital systems.

BSFIFO – Binary Structure - Queueing serialize and deserialize

Block Symbol

Licence: [ADVANCED](#)



Function Description

This block sequentially adds or removes data to/from the buffer (passed to the **uBuff** input). The elementary unit in a buffer is a column. All matrices (ie matrices or vectors fed to the inputs **uBuff**, **uMatIn**, **uMatOut**) must have the same column size in bytes. Data is organized as either a queue (if **REV=off**) or a stack (if **REV=on**). The behavior of the block depends on the inputs in this way:

- If **PUSH=on**, the content of the **uMatIn** matrix (all defined columns) is inserted into the buffer.
- If **POP=on**, the number of columns determined by the **col** parameter is removed from the buffer and this data is inserted into the **uMatOut** matrix (it must be of sufficient size).
- If **R1=on**, the data is reloaded (mainly the number of valid columns) into the block buffer. Own data is transmitted by reference and is therefore shared. This signal has priority and blocks **PUSH**, **POP** signals.

Error states (e.g. mismatched matrix dimensions, insufficient space in some matrices, lack of data in the buffer) are indicated on the **iE** output and by a message in the **SystemLog**.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uBuff	Input reference to a binary structure	Reference
uMatIn	Input reference to a matrix or vector	Reference
uMatOut	Input reference to a matrix or vector	Reference
PUSH	Enable push data	Bool
POP	Enable pop data	Bool
R1	Block reset	Bool

Parameter

OW	Overwrite oldest items in buffer	Bool
REV	Pop last pushed item first	Bool
col	Number of output (pop) columns	⊙1 Long (I32)

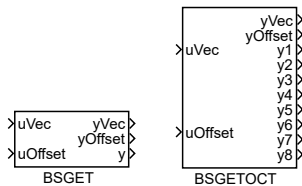
Output

yBuff	Output reference to a binary structure	Reference
yMatIn	Output reference to a matrix or vector	Reference
yMatOut	Output reference to a matrix or vector	Reference
iused	Used bytes in buffer	Long (I32)
ifree	Free bytes in buffer	Long (I32)
iE	Error code	Error

BSGET, BSGETOCT – Binary Structure - Get a single value of given type

Block Symbols

Licence: [ADVANCED](#)



Function Description

This group of blocks is used for obtaining values from a binary structure (byte array). The [BSSET](#) and [BSSETOCT](#) blocks can be used to write to the binary structure. If binary structures are received using communication, it is possible to process them directly in the block mediating communication. Typically this is a [REXLANG](#) or [PYTHON](#) programmable block. Using structures, however, it is possible to transfer data within the [REXYGEN](#) application as well. The binary structure is fed in the form of an array (vector) of bytes to the **uVec** input. The **uOffset** input specifies the offset (in bytes) of the desired value from the beginning of the structure. The value type is specified by the **type** parameter.

The **yOffset** output is the start of the next element in the structure. This is advantageous for chaining: if the structure contains several elements in a row, it is possible to connect the input **uOffset** to the output **yOffset** of the previous block and it is not necessary to calculate the offset.

The only difference between the blocks is that **BSGET** gets a single value. The **BSGETOCT** block is able to receive up to 8 values (the number is determined by the **m** parameter).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uVec	Input reference to a binary structure	Reference
uOffset	Offset to start in the input Binary Structure (in bytes)	Long (I32)

Parameter

BE	Big-Endian byte order (default is Little-Endian, e.g. Intel)	Bool
m	Number of active items	Long (I32) ↓1 ↑8 ⊖8
type1..type8	Data type of item	Long (I32) ↓2 ↑10 ⊖2

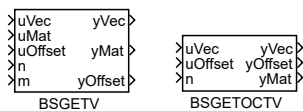
Output

<code>yVec</code>	Output reference to a binary structure	Reference
<code>yOffset</code>	Offset after the last processed byte of the input Binary Structure (in bytes), for easy chaining	Long (I32)
<code>y1..y8</code>	Scalar value output (scalar type defined by parameter)	Double (F64)

BSGETV, BSGETOCTV – Binary Structure - Get matrix (all values of same given type)

Block Symbols

Licence: [ADVANCED](#)



Function Description

This group of blocks is used for obtaining values from a binary structure (byte array). The [BSSETV](#) and [BSSETOCTV](#) blocks can be used to write to the binary structure. The meaning of most of the parameters is the same as the [BSGET](#) block, but these blocks retrieve several values of the same type and store them in an array (matrix). A matrix always has **m** rows and **n** columns. For the **BSGETV** block, all elements are of the same type (determined by the **type** parameter) and the data is filled into the matrix fed to the **uMat** input. The **BSGETOCTV** block loads up to 8 vectors. Each row of the matrix can be of a different type. The block allocates the matrix itself. The matrix is available at the **yMat** output.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uVec	Input reference to a binary structure	Reference
uOffset	Offset to start in the input Binary Structure (in bytes)	Long (I32)
n	Number of matrix columns	Long (I32)

Parameter

BE	Big-Endian byte order (default is Little-Endian, e.g. Intel)	Bool
m	Number of active items	↓1 ↑8 ⊙8 Long (I32)
nmax	Allocated size of output matrix (total number of items)	Long (I32)
		↓1 ⊙32
type	Data type of item	↓2 ↑10 ⊙2 Long (I32)
type1..type8	Data type of item	↓2 ↑10 ⊙2 Long (I32)

Output

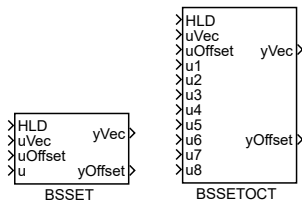
yVec	Output reference to a binary structure	Reference
yMat	Matrix value output	Reference

<code>yoffset</code>	Offset after the last processed byte of the input Binary Structure (in bytes), for easy chaining	Long (I32)
----------------------	--	------------

BSSET, BSSETOCT – Binary Structure - Set a single value of given type

Block Symbols

Licence: [ADVANCED](#)



Function Description

This group of blocks is used for setting values into a binary structure (byte array). The function is the inverse of the [BSGET](#) and [BSGETOCT](#) blocks, i.e. all signals have the same meaning, only the data is copied in the opposite direction - from the **u** input to the binary structure represented by the byte array connected to the **uVec** input. The block modifies the binary structure only if **HLD=off**.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
uVec	Input reference to a binary structure	Reference
uOffset	Offset to start in the input Binary Structure (in bytes)	Long (I32)
u1..u8	Scalar value input (scalar type defined by parameter)	Double (F64)

Parameter

m	Number of active items	↓1 ↑8 ⊙8	Long (I32)
BE	Big-Endian byte order (default is Little-Endian, e.g. Intel)		Bool
type1..type8	Data type of item	↓2 ↑10 ⊙2	Long (I32)

Output

yVec	Output reference to a binary structure	Reference
yOffset	Offset after the last processed byte of the input Binary Structure (in bytes), for easy chaining	Long (I32)

BSSETV, BSSETOCTV – Binary Structure - Set matrix (all values of same given type)

Block Symbols

Licence: [ADVANCED](#)



Function Description

This group of blocks is used to set the matrix of values into a binary structure (byte array). The function is the inverse of the [BSGETV](#) and [BSGETOCTV](#) blocks, i.e. all signals have the same meaning, only the data is copied in the opposite direction - from the **uMat** input to the binary structure represented by the byte array connected to the **uVec** input. The block modifies the binary structure only if **HLD=off**.

Unlike the [BSGETV](#) block, the numbers of rows and columns are not specified, but are determined from the actual size of the matrix connected to the **uMat** input.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uVec	Input reference to a binary structure	Reference
HLD	Hold	Bool
uMat	Reference of matrix for input values	Reference
uOffset	Offset to start in the input Binary Structure (in bytes)	Long (I32)

Parameter

m	Number of active items	↓1 ↑8 ⊖8	Long (I32)
BE	Big-Endian byte order (default is Little-Endian, e.g. Intel)		Bool
type1...type8	Data type of item	↓2 ↑10 ⊖2	Long (I32)

Output

yVec	Output reference to a binary structure	Reference
yOffset	Offset after the last processed byte of the input Binary Structure (in bytes), for easy chaining	Long (I32)

FFT – Fast Fourier Transform

Block Symbol

Licence: [ADVANCED](#)



Function Description

The function block **FFT** computes Fast Fourier Transform using the PocketFFT package [19]. The PocketFFT C99 implementation is based on FFTPack (in Fortran) [20], which is based on the chapter in [21].

Input data (a vector or a matrix) of the block are referenced by the **uc** input. If the input **uc** refers to a column vector (number of columns **m** = 1) with the number of elements **n**, the FFT will be calculated from a single signal with **n** elements. If **uc** refers to a matrix with **n** rows and **m** columns, the FFT will be computed **m** times (once for each column).

Input data is processed according to the **mode** parameter (see below), which determines whether the calculation will be performed for real or complex data and forward or backward FFT. More precisely, one of the **rfft_forward()**, **rfft_backward()**, **cfft_forward()** or **cfft_backward()** functions of the PocketFFT package is called according to the **mode** parameter.

Output data are referenced by the **uf** input. The data has the same number of **n** rows and **m** columns as the data referenced by **uc**. If the **uf** input is connected to preallocated vector/matrix then the FFT algorithm output data are referenced by the **yf** output. If the **uf** input is not connected, the FFT is calculated in place and the output data is stored in the array referenced by the **yc** output and the original data referenced by **uc** is overwritten.

The **uc** reference is copied to the **yc** output, the **uf** reference is copied to the **yf** output.

The **HLD** input allows the user to temporarily stop the FFT calculation.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uc	Input reference to input/output data	Reference
uf	Input reference to optional output data	Reference
HLD	Hold	Bool

Parameter

mode	FFT computation mode	⊙1	Long (I32)
1 Real forward		
2 Real backward		
3 Complex forward		
4 Complex backward		

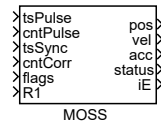
Output

yc	Output reference to input/output data	Reference
yf	Output reference to optional output data	Reference
E	Error indicator	Bool

MOSS – Motion smart sensor

Block Symbol

Licence: [ADVANCED](#)



Function Description

The **MOSS** block implements an advanced filter for incremental (quadrature) position sensors. The block requires special hardware, as correct operation necessitates knowledge not only of the current value from the sensor but also the timestamp of the last pulse, the direction of movement at the last pulse, and the timestamp of the reference moment (from the same source as the pulse timestamp). The output of the block is not only the filtered position but also the velocity and acceleration. For proper operation, it is necessary to appropriately select the **alpha** parameter. A smaller value reduces noise but increases the signal delay.

If no pulse is received in the stalled time interval, sensor is considered stopped and the outputs (**pos**, **vel**, **acc**) are set to 0. If **pos** is greater then **maxpos**, the internal position processed by the Kalman filter is decremented by an integer multiple of **maxpos** and incremented back for output. This causes the filter algorithm to calculate small enough numbers and not reduce accuracy due to rounding errors. The default value should not normally be changed. If no pulse is received for a long time, the predictor output will drift. To overcome this drift, if no pulse is detected for longer then **mindivert** time, the output position is clamped to ± 1 pulse from input (measured) position.

Note 1: It may seem impossible to determine the position more accurately than the quantization error (± 1 pulse) of the measurement, but knowing the velocity allows for a more accurate estimation of the position. Furthermore, it may appear that velocity cannot be determined more accurately than as a ratio of the number of pulses and the difference in timestamps, but by plotting both signals, it is evident that the **MOSS** signal improves. Determining acceleration through differentiation leads to utterly unusable values.

Note 2: Internally, the block implements a Kalman filter for a system with 2 integrators (i.e., input acceleration, output position). The filter is discretized anew in each period, where the discretization period is the current difference in timestamps. To derive the filter, it is necessary to know the input and output disturbances. If we consider white (Gaussian) noise, it suffices to know their ratio, which is the **alpha** parameter.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

tsPulse	Time stamp of last pulse	DWord (U32)
cntPulse	Last state of pulse counter	DWord (U32)
tsSync	Time stamp of synchronization pulse	DWord (U32)
cntCorr	Last pulse correction	Double (F64)
flags	Input status flags (1: POS, 2: NEG, 4: RUN)	DWord (U32)
R1	Block reset	DWord (U32)

Parameter

freq	Frequency of source time stamp [Hz]	$\downarrow 0.0 \odot 100000000.0$	Double (F64)
stall	Time for activation of halt state [s]	$\downarrow 0.0 \odot 0.08$	Double (F64)
alpha	Design parameter of Kalman filter	$\downarrow 0.0 \uparrow 200.0 \odot 26.0$	Double (F64)
maxpos	Rounding error optimization of Kalman filter	$\downarrow 0.0 \odot 1e+10$	Double (F64)
mindivert	Time for activation of predictor diversion limiter [s]	$\downarrow 0.0 \odot 0.003$	Double (F64)

Output

pos	Filtered position	Double (F64)
vel	Filtered velocity	Double (F64)
acc	Filtered acceleration	Double (F64)
status	Output status flags (1: POS, 2: NEG, 4: RUN, 8: INIT, 16: PULSE, 32: STALLED, 64: DIVERT)	Long (I32)
iE	Error code	Error

PCI – PCI Bus Memory Access

Block Symbol

Licence: [ADVANCED](#)



Function Description

The PCI block provides access to PCI device resources defined by the path:

`/sys/bus/pci/devices/<device>/resource<resource>`

The data is presented at the output `y` of the block as a reference to a byte array. The device is specified using the `pci_device` parameter, which accepts two formats:

- The device slot, as displayed in `lspci` command output, formatted as:
`<domain>:<bus>:<slot>.<func>`.
- A combination of the vendor and device IDs with an optional index, formatted as:
`[<vendor>]:[<device>][#<index>]`.

Both vendor and device IDs are hexadecimal numbers that can be retrieved from the `lspci -nn` command output, without the leading "0x". You can omit either the vendor or device ID. If the index is not specified, the first device matching the provided vendor and device IDs is selected.

The default setting for the `size` parameter is 0, which uses the entire resource. However, this can lead to the allocation of a large virtual memory area.

Note: Block is supported only on Linux platforms.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Parameter

<code>pci_device</code>	PCI device: <code><domain>:<bus>:<slot>.<func></code> or <code>id</code>	String
<code>resource</code>	PCI resource number	Long (I32)
<code>offset</code>	Offset in pci resource	Large (I64)
<code>size</code>	Size of data to open (0 for whole resource)	Large (I64)

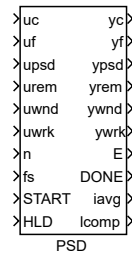
Output

<code>y</code>	Reference to byte array	Reference
----------------	-------------------------	-----------

PSD – Power Spectral Density

Block Symbol

Licence: [ADVANCED](#)



Function Description

The function block **PSD** computes a power spectral density (PSD, periodogram) using the PocketFFT package [19]. The PocketFFT C99 implementation is based on FFTPack (in Fortran) [20], which is based on the chapter in [21].

PSD is most often calculated from segments of length N samples of the input signal. The data of these segments is multiplied by a window function (e.g. Hamming window), the FFT is calculated from them, and the PSD of each segment is calculated from its result. Finally, the **navg** of such results is averaged (the **navg** parameter). The input signal can be divided into segments either sequentially or with a half-overlap (Welch method [22], [23]). The resulting PSD can be calculated in decibels (dB) if the parameter **DB** = **on**.

Input data (a vector or a matrix) of the block are referenced by the **uc** input. If the input **uc** refers to a column vector (number of columns $m = 1$) with the number of elements n , the PSD will be calculated from a single signal. If **uc** refers to a matrix with n rows and m columns, the PSD will be computed m times (for each column). If the input $n > 0$ then the number of samples per segment $N = n$ else N is set to the number of rows of the array referenced by **uc**.

The input **uf** references the internal vector of dimension $\geq N$ for FFT computing. The resulting PSD with **npsd** rows and m columns is referenced by **upsd**. Number of PSD frequencies $npsd = N/2 + 1$ for even N or $npsd = (N + 1)/2$ for odd N . Working array referenced by **uwrk** has the same dimension as the array referenced by **upsd**. If the input data contains more elements than can be processed in one execution of the PSD block, the remaining data is copied into an array referenced by **urem** that has N rows and m columns.

Input data is processed according to the **mode** and **mtrig** parameters (see below). The **mode** parameter specifies the way the PDS is calculated, whether the data in the input vector referenced by **uc** is real or complex, and whether segment overlapped by half will be used. The input data sampled with the frequency connected to the input **fs** may be acquired in a different task (e.g. with a significantly shorter sampling period)

than the one in which the PSD is calculated. The method of synchronizing the start of the calculation is provided by the **mtrig** trigger mode. The calculation can be triggered every PSD block execution period or when a sufficient number of samples are available (at least equal to **N**) or by the rising edge of the **START** input.

A vector of dimension **N** is connected to the **uwnd** input to store the window function selected by the **iwin** parameter. Window functions have (optionally) an integer parameter **lwnd** and/or a real parameter **rwnd**. Some window functions are used in two variants:

Symmetric (for **lwnd=0**). The window of length $L = N$ samples is computed, the first and last elements are equal. This variant is suitable for FIR filter design.

Periodic (for **lwnd=1**). The window of length $L = N + 1$ is computed but only the first N samples is stored. The periodic version is the preferred method for spectral analysis because the discrete Fourier transform assumes periodic extension of the input vector.

Window functions are defined by the following expressions:

iwin=1 Bartlett-Hahn Window

$$w(n) = 0.62 - 0.48 \left| \left(\frac{n}{N-1} - 0.5 \right) \right| + 0.38 \cos \left[2\pi \left(\frac{n}{N-1} - 0.5 \right) \right], \quad 0 \leq n \leq N-1.$$

iwin=2 Bartlett Window

$$w(n) = \begin{cases} \frac{2n}{N}, & 0 \leq n \leq \frac{N}{2}, \\ 2 - \frac{2n}{N}, & \frac{N}{2} \leq n \leq N. \end{cases}$$

iwin=3 Blackman Window

$$w(n) = 0.42 - 0.5 \cos \left(\frac{2\pi n}{L-1} \right) + 0.08 \cos \left(\frac{4\pi n}{L-1} \right), \quad 0 \leq n \leq M-1,$$

where M is $N/2$ when N is even and $(N+1)/2$ when N is odd.

iwin=4 Blackman-Harris Window

$$w(n) = a_0 - a_1 \cos \left(\frac{2\pi n}{L-1} \right) + a_2 \cos \left(\frac{4\pi n}{L-1} \right) - a_3 \cos \left(\frac{6\pi n}{L-1} \right), \quad 0 \leq n \leq N-1$$

where $a_0 = 0.35875$, $a_1 = 0.48829$, $a_2 = 0.14128$ and $a_3 = 0.01168$.

iwin=5 Bohman Window

$$w(x) = (1 - |x|) \cos(\pi |x|) + \frac{1}{\pi} \sin(\pi |x|), \quad -1 \leq x \leq 1$$

iwin=6 Chebyshev Window

Using Chebyshev polynomials of n -th order

$$T_n(x) \triangleq \begin{cases} \cos [n \cos^{-1}(x)] , & |x| \leq 1, \\ \cosh [n \cosh^{-1}(x)] , & |x| > 1, \end{cases}$$

the Fourier transform of the Chebyshev window is

$$W(k) = \frac{T_{N-1}[x_0 \cos(\pi k/N)]}{T_{N-1}(x_0)}$$

where

$$x_0 = \cosh \left[\frac{\cosh^{-1}(10^{r/20})}{N-1} \right].$$

iwin=7 Flat Top Window

$$w(n) = a_0 - a_1 \cos \left(\frac{2\pi n}{L-1} \right) + a_2 \cos \left(\frac{4\pi n}{L-1} \right) - a_3 \cos \left(\frac{6\pi n}{L-1} \right) + a_4 \cos \left(\frac{8\pi n}{L-1} \right),$$

where $0 \leq n \leq N-1$ and $a_0 = 0.21557895$, $a_1 = 0.41663158$, $a_2 = 0.277263158$, $a_3 = 0.083578947$ and $a_4 = 0.006947368$.

iwin=8 Gauss Window

$$w(n) = e^{-\frac{1}{2} \left[\alpha \frac{n}{(N-1)/2} \right]^2} = e^{-n^2/(2\sigma^2)}$$

where $-(N-1)/2 \leq n \leq (N-1)/2$ and $\sigma = (N-1)/(2\alpha)$ is the standard deviation of a Gaussian probability density function.

iwin=9 Hamming Window

$$w(n) = 0.54 - 0.46 \cos \left(2\pi \frac{n}{L-1} \right), \quad 0 \leq n \leq N-1,$$

iwin=10 Hann Window

$$w(n) = 0.5 \left[1 - \cos \left(2\pi \frac{n}{L-1} \right) \right], \quad 0 \leq n \leq N-1,$$

iwin=12 Nuttall's Blackman-Harris Window

$$w(n) = a_0 - a_1 \cos \left(\frac{2\pi n}{L-1} \right) + a_2 \cos \left(\frac{4\pi n}{L-1} \right) - a_3 \cos \left(\frac{6\pi n}{L-1} \right), \quad 0 \leq n \leq N-1$$

where $a_0 = 0.3635819$, $a_1 = 0.4891775$, $a_2 = 0.1365995$ and $a_3 = 0.0106411$.

iwin=13 Parzen Window

$$w(n) = \begin{cases} 1 - 6 \left(\frac{|n|}{N/2} \right)^2 + 6 \left(\frac{|n|}{N/2} \right)^3, & 0 \leq |n| \leq (N-1)/4, \\ 2 \left(1 - \frac{|n|}{N/2} \right)^3, & (N-1)/4 < |n| \leq (N-1)/2. \end{cases}$$

iwin=14 Rectangular Window

$$w(n) = 1, \quad 0 \leq n \leq N-1.$$

Multiplication of all input data elements by 1 is not performed in this block, therefore the **uwnd** input may not be connected in this case.

iwin=16 Triangular Window

For N odd:

$$w(n) = \begin{cases} \frac{2n+2}{N+1}, & 0 \leq n \leq (N-1)/2, \\ 2 - \frac{2n+2}{N+1}, & (N-1)/2 < n \leq (N-1). \end{cases}$$

For N even:

$$w(n) = \begin{cases} \frac{2n+1}{N}, & 0 \leq n \leq N/2 - 1, \\ 2 - \frac{2n+1}{N}, & N/2 \leq n \leq (N-1). \end{cases}$$

iwin=17 Tukey Window

$$w(x) = \begin{cases} \frac{1}{2} \left\{ 1 + \cos \left(\frac{2\pi}{r} [x - r/2] \right) \right\}, & 0 \leq x < \frac{r}{2}, \\ 1, & \frac{r}{2} \leq x < 1 - \frac{r}{2}, \\ \frac{1}{2} \left\{ 1 + \cos \left(\frac{2\pi}{r} [x - 1 + r/2] \right) \right\}, & 1 - \frac{r}{2} \leq x < 1, \end{cases}$$

where r is so called cosine fraction (set in the parameter **rwnd**, default value 0.5).

iwin=18 Exponential Window

$$w(x) = e^{-\frac{|x|}{\tau}}, \quad -1 \leq x \leq 1,$$

where τ is a window parameter (set in the parameter **rwnd**).

iwin=19 Welch Window

$$w(x) = 1 - x^2, \quad -1 \leq x \leq 1.$$

iwin=20 Externally Defined Window

If the user is not satisfied with any of the predefined windows, he/she can set his own window consisting of N samples in the vector referenced by **uwnd** before executing the PSD block.

The HLD input allows the user to temporarily stop the PSD calculation.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uc	Input reference to input data	Reference
uf	Input reference to internal FFT vector	Reference
upsd	Input reference to output PSD vector/matrix	Reference
urem	Input reference to remaining (not processed) data	Reference
uwnd	Input reference to window function vector	Reference
uwrk	Input reference to working vector/matrix	Reference
n	Number of signal samples for FFT segment	Long (I32)
fs	Sampling frequency in [Hz]	Double (F64)
START	Starting signal (rising edge)	Bool
HLD	Hold	Bool

Parameter

mode	PSD computation mode	⊙1 Long (I32)
	1 Real	
	2 Real overlap	
	3 Complex	
	4 Complex overlap	
mtrig	Computation trigger mode	⊙1 Long (I32)
	1 Each period	
	2 Enough samples	
	3 START rising edge	

iwnd	Window function	⊙1	Long (I32)
	1 Bartlett-Hann		
	2 Bartlett		
	3 Blackman		
	4 Blackman-Harris		
	5 Bohman		
	6 Chebyshev		
	7 Flat top		
	8 Gaussian		
	9 Hamming		
	10 Hann		
	11 —		
	12 Nuttall		
	13 Parzen		
	14 Rectangular		
	15 —		
	16 Triangular		
	17 Tukey		
	18 Exponential		
	19 Welch		
	20 External		
lwnd	Parameter lwnd of some window functions	↓1 ↑9 ⊙1	Long (I32)
rwnd	Parameter rwnd of some window functions	↓1 ↑9 ⊙1	Long (I32)
navg	Number of FFT segments for averaging	↓1 ⊙10	Long (I32)
DB	Computed PSD is converted do decibels [dB]		Bool

Output

yc	Output reference to input data	Reference
yf	Output reference to internal FFT vector	Reference
ypsd	Output reference to output PSD vector/matrix	Reference
yrem	Output reference to remaining (not processed) data	Reference
ywnd	Output reference to external window function	Reference
ywrk	Output reference to working vector/matrix	Reference
E	Error indicator	Bool
DONE	Averaging Completion Flag	Bool
iavg	Current index of segment for averaging	Long (I32)
lcomp	Number of successfully computed PSDs	Large (I64)

Chapter 19

MQTTDrv – Communication via MQTT protocol

Contents

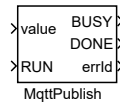
MqttPublish – Publish MQTT message	634
MqttSubscribe – Subscribe to MQTT topic	636

The MQTTDrv library is designed for IoT (Internet of Things) communication using the MQTT (Message Queuing Telemetry Transport) protocol. It consists of two primary blocks: [MqttPublish](#) and [MqttSubscribe](#). The [MqttPublish](#) block is used for sending messages to an MQTT broker, enabling the publication of data to MQTT topics. Conversely, the [MqttSubscribe](#) block is designed for subscribing to topics and receiving messages from a broker. This library facilitates efficient and effective data communication in IoT applications, leveraging the lightweight and widely-used MQTT protocol for message exchange.

MqttPublish – Publish MQTT message

Block Symbol

Licence: [MQTT](#)



Function Description

This function block depends on the MQTT driver. Please read the `MQTTRv` manual [\[24\]](#) before use.

The `MqttPublish` block publishes messages to an MQTT broker through the connection established by the `MQTTRv` 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 [\[25\]](#) 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 [\[25\]](#) 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`.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>value</code>	Input signal	String
<code>RUN</code>	Enable execution	Bool

Parameter

<code>topic</code>	MQTT topic	String
--------------------	------------	--------

QoS	Quality of Service	⊙1	Long (I32)
	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 (I32)

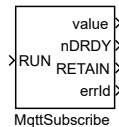
Output

BUSY	Busy flag	Bool
DONE	Indicator of finished transaction	Bool
errId	Error code	Error

MqttSubscribe – Subscribe to MQTT topic

Block Symbol

Licence: [MQTT](#)



Function Description

This function block depends on the MQTT driver. Please read the **MQTTRv** manual [24] 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 **MQTTRv** 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 [25] 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 [25] for more details.

Note that subscribing to topics containing wildcards is not supported.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>RUN</code>	Enable execution	<code>Bool</code>
------------------	------------------	-------------------

Parameter

<code>topic</code>	MQTT topic	<code>String</code>
<code>QoS</code>	Quality of Service	$\odot 1$ <code>Long (I32)</code>
	1 QoS0 (At most once)	
	2 QoS1 (At least once)	
	3 QoS2 (Exactly once)	
<code>type</code>	Expected type of incoming data	$\odot 1$ <code>Long (I32)</code>
	1 string	
	2 double	
	3 long	
	4 bool	
	5 byte vector/blob	
<code>mode</code>	Incoming messages buffering mode	$\odot 1$ <code>Long (I32)</code>
	1 Last value	
	2 Buffered values	
<code>defBuffSize</code>	Default buffer size	$\downarrow 1$ $\odot 2048$ <code>Long (I32)</code>

Output

<code>value</code>	Output signal	<code>Any</code>
<code>nDRDY</code>	Number of received messages	$\downarrow 0 \uparrow 10$ <code>Long (I32)</code>
<code>RETAIN</code>	Retain last message	$\odot on$ <code>Bool</code>
<code>errId</code>	Error code	<code>Error</code>

Chapter 20

MC_SINGLE – Motion control - single axis blocks

Contents

MCP_AccelerationProfile – * Acceleration profile	641
MCP_Halt – * Stopping a movement (interruptible)	643
MCP_HaltSuperimposed – * Stopping a movement (superimposed and interruptible)	644
MCP_Home – * Homing	645
MCP_MoveAbsolute – * Move to position (absolute coordinate) . .	647
MCP_MoveAdditive – * Move to position (relative to previous motion)	649
MCP_MoveContinuousAbsolute – * Move to position (absolute coor- dinate)	651
MCP_MoveContinuousRelative – * Move to position (relative to pre- vious motion)	653
MCP_MoveRelative – * Move to position (relative to execution point)	655
MCP_MoveSuperimposed – * Superimposed move	657
MCP_MoveVelocity – * Move with constant velocity	658
MCP_PositionProfile – * Position profile	660
MCP_SetOverride – * Set override factors	662
MCP_Stop – * Stopping a movement	663
MCP_TorqueControl – * Torque/force control	664
MCP_VelocityProfile – * Velocity profile	666
MC_AccelerationProfile, MCP_AccelerationProfile – Acceleration profile	668
MC_Halt, MCP_Halt – Stopping a movement (interruptible)	672
MC_HaltSuperimposed, MCP_HaltSuperimposed – Stopping a move- ment (superimposed and interruptible)	673
MC_Home, MCP_Home – Homing	674

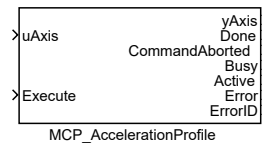
MC_MoveAbsolute, MCP_MoveAbsolute – Move to position (absolute coordinate)	676
MC_MoveAdditive, MCP_MoveAdditive – Move to position (relative to previous motion)	680
MC_MoveContinuousAbsolute, MCP_MoveContinuousAbsolute – Move to position (absolute coordinate)	683
MC_MoveContinuousRelative, MCP_MoveContinuousRelative – Move to position (relative to previous motion)	686
MC_MoveRelative, MCP_MoveRelative – Move to position (relative to execution point)	690
MC_MoveSuperimposed, MCP_MoveSuperimposed – Superimposed move	693
MC_MoveVelocity, MCP_MoveVelocity – Move with constant velocity	696
MC_PositionProfile, MCP_PositionProfile – Position profile	700
MC_Power – Axis activation (power on/off)	704
MC_ReadActualPosition – Read actual position	705
MC_ReadAxisError – Read axis error	706
MC_ReadBoolParameter – Read axis parameter (bool)	707
MC_ReadParameter – Read axis parameter	708
MC_ReadStatus – Read axis status	710
MC_Reset – Reset axis errors	712
MC_SetOverride, MCP_SetOverride – Set override factors	713
MC_Stop, MCP_Stop – Stopping a movement	715
MC_TorqueControl, MCP_TorqueControl – Torque/force control . . .	717
MC_VelocityProfile, MCP_VelocityProfile – Velocity profile	720
MC_WriteBoolParameter – Write axis parameter (bool)	724
MC_WriteParameter – Write axis parameter	725
RM_Axis – Motion control axis	726
RM_AxisOut – Axis output	733
RM_AxisSpline – Commanded values interpolation	734
RM_HomeOffset – * Homing by setting offset	739
RM_Track – Tracking and inching	740

The MC_SINGLE library is designed for motion control in single-axis systems. It features blocks like `MC_MoveAbsolute`, `MC_MoveRelative`, and `MC_MoveVelocity` for precise positioning and speed control. The library includes `MC_Home` for homing operations, and `MC_Power` for controlling the power state of the axis. Advanced functionalities are provided by `MC_AccelerationProfile`, `MC_PositionProfile`, and `MC_VelocityProfile` for customizing motion profiles. It also offers monitoring and parameter adjustment capabilities through `MC_ReadActualPosition`, `MC_ReadAxisError`, `MC_ReadParameter`, and `MC_WriteParameter`. Additionally, the library contains blocks like `MC_Halt`, `MC_Reset`, and `MC_Stop` for emergency and control operations. This library is essential for applications requiring precise and controlled motion in single-axis configurations.

MCP_AccelerationProfile — * Acceleration profile

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

Parameter

alg	Algorithm for interpolation	⊙1	Long (I32)
	1 Sequence of time/value pairs		
	2 Sequence of equidistant values		
	3 Spline		
	4 Equidistant spline		
	5 Sequence of time/value pairs (+border)		
	6 Sequence of equidistant values (+border)		
	7 cubic aproximation (B-spline)		
	8 quintic aproximation (B-spline)		
	9 all linear		
nmax	Number of profile segments	⊙3	Long (I32)
TimeScale	Overall scale factor in time	⊙1.0	Double (F64)
AccelerationScale	Overall scale factor in value	⊙1.0	Double (F64)
Offset	Overall profile offset in value		Double (F64)

BufferMode	Buffering mode	⊙1	Long (I32)
	1 Aborting		
	2 Buffered		
	3 Blending low		
	4 Blending high		
	5 Blending previous		
	6 Blending next		
times	Times when segments are switched	⊙[0 30]	Double (F64)
values	Values or interpolating polynomial coefficients (a0, a1, a2, ...)	⊙[0 100 100 50]	Double (F64)

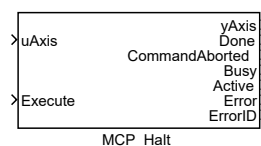
Output

yAxis	Axis reference (only RM_Axis.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 last operation	Error
	i REXYGEN error code	

MCP_Halt — * Stopping a movement (interruptible)

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

Parameter

Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	Double (F64)
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	

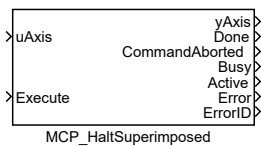
Output

yAxis	Axis reference (only RM_Axis.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 last operation	Error
	i REXYGEN error code	

MCP_HaltSuperimposed – * **Stopping a movement (superimposed and interruptible)**

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

Parameter

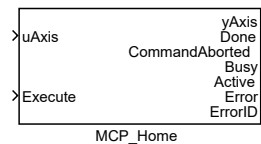
Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	Double (F64)
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	

Output

yAxis	Axis reference (only RM_Axis.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 last operation	Error
	i REXYGEN error code	

MCP_Home — * **Homing**

Block Symbol

Licence: [MOTION CONTROL](#)

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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

Parameter

Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]	Double (F64)
TorqueLimit	Maximal allowed torque/force	Double (F64)
TimeLimit	Maximal allowed time for the whole algorithm [s]	Double (F64)
DistanceLimit	Maximal allowed distance for the whole algorithm [unit]	Double (F64)
LagLimit	Maximal allowed lag for the whole algorithm [unit]	Double (F64)
Direction	Direction of movement (cyclic axis or special case only)	⊙3 Long (I32)
	1 Positive	
	2 Shortest	
	3 Negative	
	4 Current	
Position_	Requested target position (absolute) [unit]	Double (F64)

HomingMode	Homing mode algorithm	⊙1	Long (I32)
	1 Absolute switch		
	2 Limit switch		
	3 Reference pulse		
	4 Direct (user reference)		
	5 Absolute encoder		
	6 Block		
	7 reserved1		
	8 reserved2		

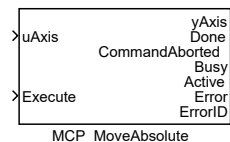
Output

yAxis	Axis reference (only RM_Axis.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 last operation	Error
	i REXYGEN error code	

MCP_MoveAbsolute – * Move to position (absolute coordinate)

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

Parameter

Position_	Requested target position (absolute) [unit]	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	↓0.0 Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]	↓0.0 Double (F64)
Deceleration	Maximal allowed deceleration [unit/s^2]	↓0.0 Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	↓0.0 Double (F64)
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	
	3 Blending low	
	4 Blending high	
	5 Blending previous	
	6 Blending next	
Direction	Direction of movement (cyclic axis or special case only)	⊙1 Long (I32)
	1 Positive	
	2 Shortest	
	3 Negative	
	4 Current	

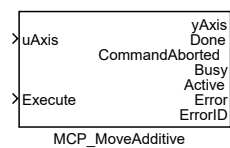
Output

yAxis	Axis reference (only RM_Axis.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 last operation	Error
	i REXYGEN error code	

MCP_MoveAdditive – * Move to position (relative to previous motion)

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

Parameter

Distance	Requested target distance (relative to start point) [unit]	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	↓0.0 Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]	↓0.0 Double (F64)
Deceleration	Maximal allowed deceleration [unit/s^2]	↓0.0 Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	↓0.0 Double (F64)
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	
	3 Blending low	
	4 Blending high	
	5 Blending previous	
	6 Blending next	

Output

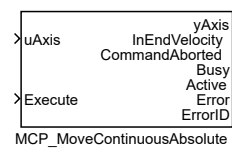
yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Done	Algorithm finished	Bool

<code>CommandAborted</code>	Algorithm was aborted	<code>Bool</code>
<code>Busy</code>	Algorithm not finished yet	<code>Bool</code>
<code>Active</code>	The block is controlling the axis	<code>Bool</code>
<code>Error</code>	Error occurred	<code>Bool</code>
<code>ErrorID</code>	Result of last operation	<code>Error</code>
	<i>i</i> REXYGEN error code	

MCP_MoveContinuousAbsolute — * Move to position (absolute coordinate)

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

Parameter

Position_	Requested target position (absolute) [unit]	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	↓0.0 Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]	↓0.0 Double (F64)
Deceleration	Maximal allowed deceleration [unit/s^2]	↓0.0 Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	↓0.0 Double (F64)
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	
	3 Blending low	
	4 Blending high	
	5 Blending previous	
	6 Blending next	
Direction	Direction of movement (cyclic axis or special case only)	⊙1 Long (I32)
	1 Positive	
	2 Shortest	
	3 Negative	
	4 Current	
EndVelocity	End velocity	Double (F64)

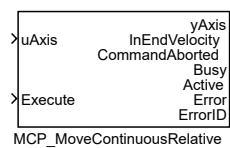
Output

yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
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 last operation	Error
	i REXYGEN error code	

MCP_MoveContinuousRelative – * Move to position (relative to previous motion)

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

Parameter

Distance	Requested target distance (relative to execution point) [unit]	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	↓0.0 Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]	↓0.0 Double (F64)
Deceleration	Maximal allowed deceleration [unit/s^2]	↓0.0 Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	↓0.0 Double (F64)
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	
	3 Blending low	
	4 Blending high	
	5 Blending previous	
	6 Blending next	
EndVelocity	End velocity	Double (F64)

Output

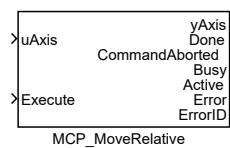
yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
--------------	--	------------------

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 last operation	Error
	i REXYGEN error code	

MCP_MoveRelative – * Move to position (relative to execution point)

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

Parameter

Distance	Requested target distance (relative to execution point) [unit]	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	↓0.0 Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]	↓0.0 Double (F64)
Deceleration	Maximal allowed deceleration [unit/s^2]	↓0.0 Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	↓0.0 Double (F64)
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	
	3 Blending low	
	4 Blending high	
	5 Blending previous	
	6 Blending next	

Output

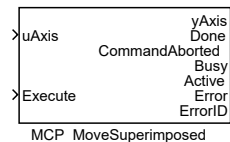
yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Done	Algorithm finished	Bool

<code>CommandAborted</code>	Algorithm was aborted	<code>Bool</code>
<code>Busy</code>	Algorithm not finished yet	<code>Bool</code>
<code>Active</code>	The block is controlling the axis	<code>Bool</code>
<code>Error</code>	Error occurred	<code>Bool</code>
<code>ErrorID</code>	Result of last operation	<code>Error</code>
	<i>i</i> REXYGEN error code	

MCP_MoveSuperimposed – * Superimposed move

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

Parameter

Distance	Requested target distance (relative to execution point) [unit]	Double (F64)
VelocityDiff	Maximal allowed velocity [unit/s]	↓0.0 Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]	↓0.0 Double (F64)
Deceleration	Maximal allowed deceleration [unit/s^2]	↓0.0 Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	↓0.0 Double (F64)

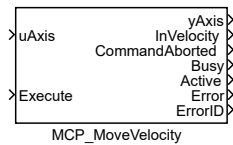
Output

yAxis	Axis reference (only RM_Axis.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 last operation	Error
i REXYGEN error code		

MCP_MoveVelocity – * **Move with constant velocity**

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

Parameter

Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s^2]	Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	Double (F64)
Direction	Direction of movement (cyclic axis or special case only)	⊙1 Long (I32)
	1 Positive	
	2 Shortest	
	3 Negative	
	4 Current	
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	
	3 Blending low	
	4 Blending high	
	5 Blending previous	
	6 Blending next	

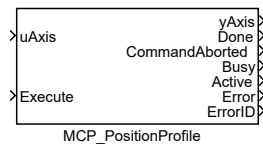
Output

yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
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 last operation	Error
	i REXYGEN error code	

MCP_PositionProfile – * **Position profile**

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

Parameter

alg	Algorithm for interpolation	⊙2	Long (I32)
	1 Sequence of time/value pairs		
	2 Sequence of equidistant values		
	3 Spline		
	4 Equidistant spline		
	5 Sequence of time/value pairs (+border)		
	6 Sequence of equidistant values (+border)		
	7 cubic aproximation (B-spline)		
	8 quintic aproximation (B-spline)		
	9 all linear		
nmax	Number of profile segments	⊙3	Long (I32)
TimeScale	Overall scale factor in time	⊙1.0	Double (F64)
PositionScale	Overall scale factor in value	⊙1.0	Double (F64)
Offset	Overall profile offset in value		Double (F64)

BufferMode	Buffering mode	⊙1	Long (I32)
	1 Aborting		
	2 Buffered		
	3 Blending low		
	4 Blending high		
	5 Blending previous		
	6 Blending next		
BeginVelocity	trajectory begin velocity		Double (F64)
EndVelocity	trajectory end velocity		Double (F64)
times	Times when segments are switched	⊙[0 30]	Double (F64)
values	Values or interpolating polynomial coefficients (a0, a1, a2, ...)	⊙[0 100 100 50]	Double (F64)

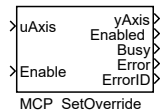
Output

yAxis	Axis reference (only RM_Axis.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 last operation	Error
	i REXYGEN error code	

MCP_SetOverride – * Set override factors

Block Symbol

Licence: MOTION CONTROL



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.

This block does not propagates the signal quality. More information can be found in the 1.4 section.

Input

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Enable	Block function is enabled	Bool

Parameter

diff	Minimal allowed difference of override factor	↓0.0 ↑1.0	Double (F64)
VelFactor	Velocity multiplication factor	⊙1.0	Double (F64)
AccFactor	Acceleration/deceleration multiplication factor	⊙1.0	Double (F64)
JerkFactor	Jerk multiplication factor	⊙1.0	Double (F64)

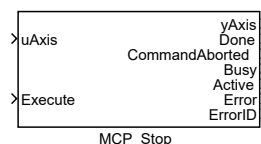
Output

yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Enabled	Block function is enabled	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of last operation i REXYGEN error code	Error

MCP_Stop – * Stopping a movement

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

Parameter

Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	Double (F64)

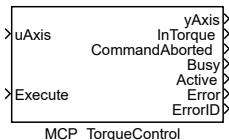
Output

yAxis	Axis reference (only RM_Axis.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 last operation	Error
i REXYGEN error code		

MCP_TorqueControl – * **Torque/force control**

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

Parameter

kma	Torque/force to acceleration ratio	Double (F64)
Torque	Maximal allowed torque/force	Double (F64)
TorqueRamp	Maximal allowed torque/force ramp	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s^2]	Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	Double (F64)
Direction	Direction of movement (cyclic axis or special case only)	⊙1 Long (I32)
	1 Positive	
	2 Shortest	
	3 Negative	
	4 Current	
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	
	3 Blending low	
	4 Blending high	
	5 Blending previous	
	6 Blending next	

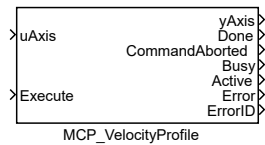
Output

yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
InTorque	Requested torque/force is 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 last operation	Error
	i REXYGEN error code	

MCP_VelocityProfile – * **Velocity profile**

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

Parameter

alg	Algorithm for interpolation	⊙1	Long (I32)
	1 Sequence of time/value pairs		
	2 Sequence of equidistant values		
	3 Spline		
	4 Equidistant spline		
	5 Sequence of time/value pairs (+border)		
	6 Sequence of equidistant values (+border)		
	7 cubic aproximation (B-spline)		
	8 quintic aproximation (B-spline)		
	9 all linear		
nmax	Number of profile segments	⊙3	Long (I32)
TimeScale	Overall scale factor in time	⊙1.0	Double (F64)
VelocityScale	Overall scale factor in value	⊙1.0	Double (F64)
Offset	Overall profile offset in value		Double (F64)

BufferMode	Buffering mode	⊙1	Long (I32)
	1 Aborting		
	2 Buffered		
	3 Blending low		
	4 Blending high		
	5 Blending previous		
	6 Blending next		
BeginAcceleration	trajectory begin aceleration		Double (F64)
EndAcceleration	trajectory end acceleration		Double (F64)
times	Times when segments are switched	⊙[0 15 25 30]	Double (F64)
values	Values or interpolating polynomial coefficients (a0, a1, a2, ...)	⊙[0 100 100 50]	Double (F64)

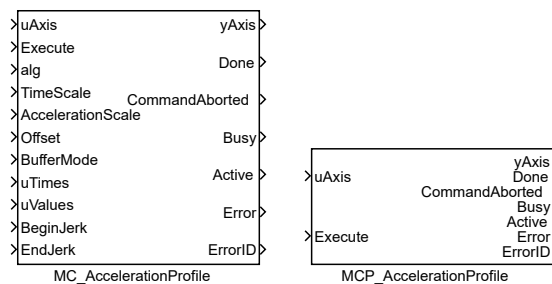
Output

yAxis	Axis reference (only RM_Axis.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 last operation	Error
	i REXYGEN error code	

MC_AccelerationProfile, MCP_AccelerationProfile – Acceleration profile

Block Symbols

Licence: [MOTION CONTROL](#)



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 fails with error -707 (step). It is recommended to use `BufferMode=BlendingNext` to eliminate the problem with start velocity.

Inputs

<code>uAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	Reference
<code>Execute</code>	The block is activated on rising edge	Bool
<code>TimeScale</code>	Overall scale factor in time	Double (F64)
<code>AccelerationScale</code>	Overall scale factor in value	Double (F64)
<code>Offset</code>	Overall profile offset in value	Double (F64)
<code>BufferMode</code>	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)	

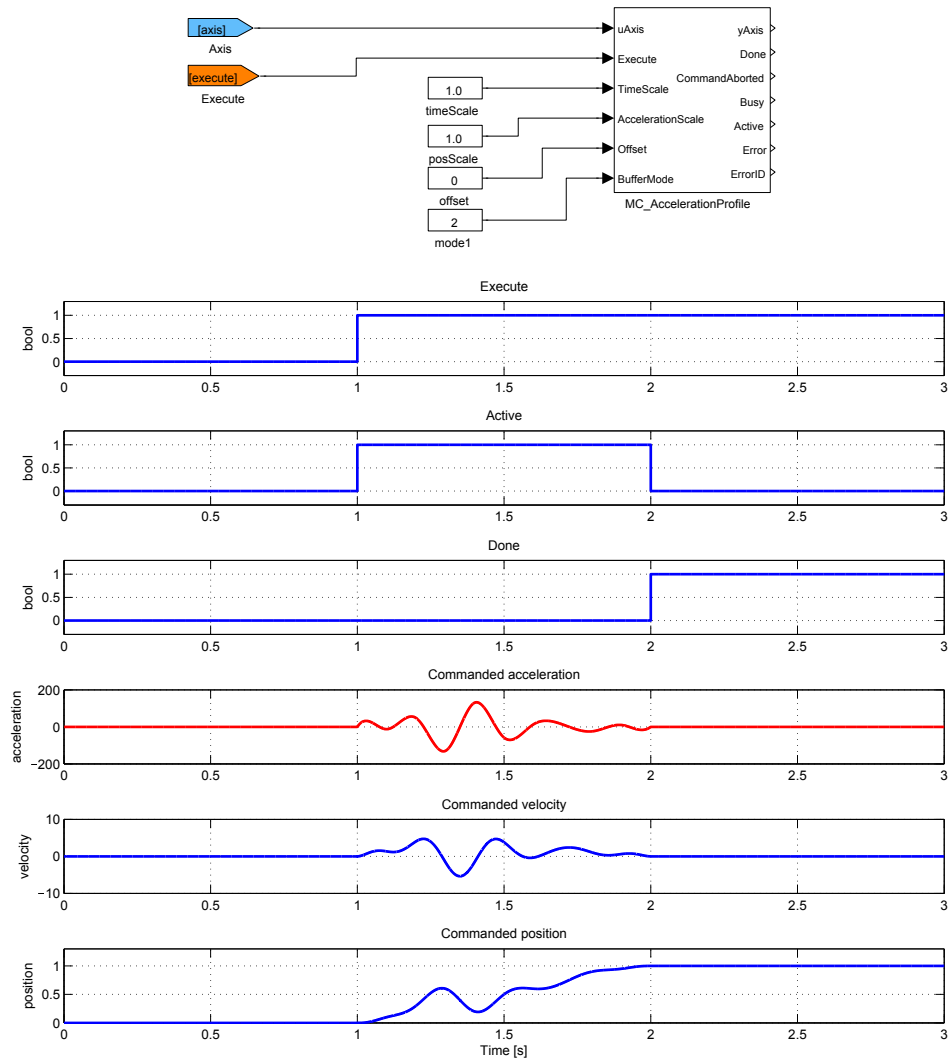
Outputs

<code>yAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	Reference
<code>Done</code>	Algorithm finished	Bool
<code>CommandAborted</code>	Algorithm was aborted	Bool
<code>Busy</code>	Algorithm not finished yet	Bool
<code>Active</code>	The block is controlling the axis	Bool
<code>Error</code>	Error occurred	Bool
<code>ErrorID</code>	Result of the last operation	Error
	i REXYGEN general error	

Parameters

alg	Algorithm for interpolation	⊙2	Long (I32)
	1 Sequence of time/value pairs		
	2 Sequence of equidistant values		
	3 Spline		
	4 Equidistant spline		
nmax	Number of profile segments	⊙3	Long (I32)
times	Times when segments are switched		Reference
values	Values or interpolating polynomial coefficients (a0, a1, a2, ...)		Reference

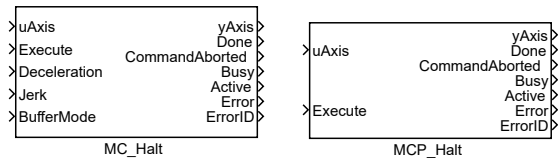
Example



MC_Halt, MCP_Halt – Stopping a movement (interruptible)

Block Symbols

Licence: [MOTION CONTROL](#)



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

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	Double (F64)

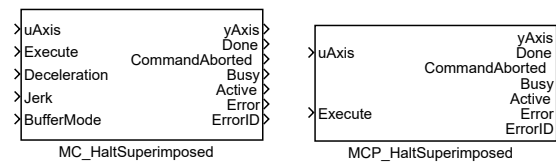
Outputs

yAxis	Axis reference (only RM_Axis.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 i REXYGEN general error	Error

MC_HaltSuperimposed, MCP_HaltSuperimposed – Stopping a movement (superimposed and interruptible)

Block Symbols

Licence: [MOTION CONTROL](#)



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

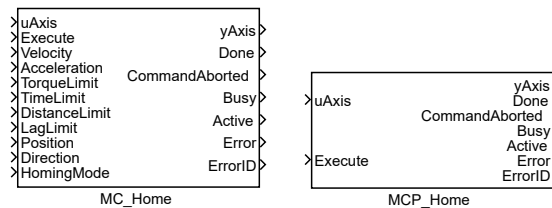
uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	Double (F64)

Outputs

yAxis	Axis reference (only RM_Axis.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	

MC_Home, MCP_Home – **Homing**

Block Symbols

Licence: [MOTION CONTROL](#)

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 Block 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 **LimZ**).

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

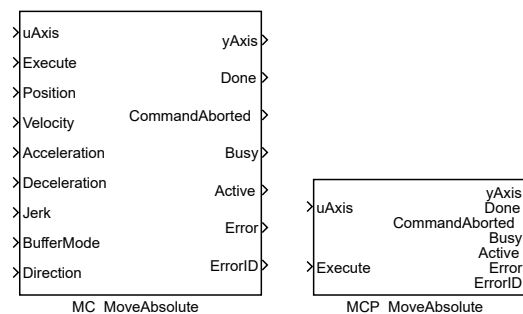
uAxis	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s ²]	Double (F64)
TorqueLimit	Maximal allowed torque/force	Double (F64)
TimeLimit	Maximal allowed time for the whole algorithm [s]	Double (F64)
DistanceLimit	Maximal allowed distance for the whole algorithm [unit]	Double (F64)
Position	Requested target position (absolute) [unit]	Double (F64)
Direction	Direction of movement (cyclic axis or special case only)	Long (I32)
	1 Positive	
	2 Shortest	
	3 Negative	
	4 Current	
HomingMode	Homing mode algorithm	Long (I32)
	1 Absolute switch	
	2 Limit switch	
	3 Reference pulse	
	4 Direct (user reference)	
	5 Absolute encoder	
	6 Block	

Outputs

yAxis	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> 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	

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.

Inputs

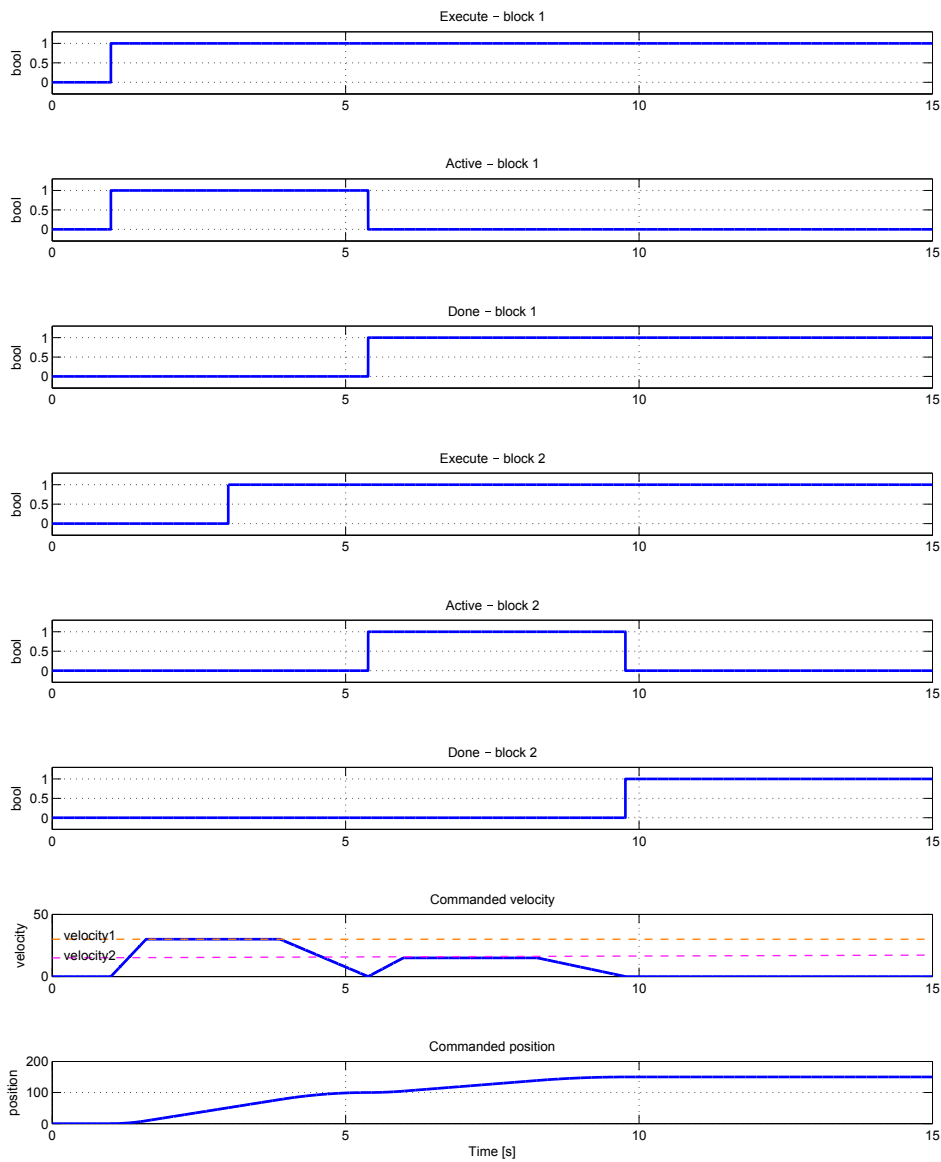
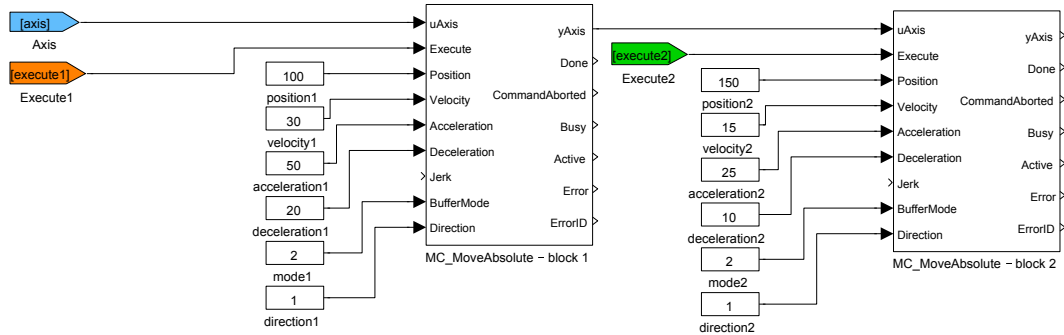
uAxis	AAxis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Position	Requested target position (absolute) [unit]	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s ²]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	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)	
Direction	Direction of movement (cyclic axis or special case only)	Long (I32)
	1 Positive	
	2 Shortest	
	3 Negative	
	4 Current	

Outputs

yAxis	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> 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	

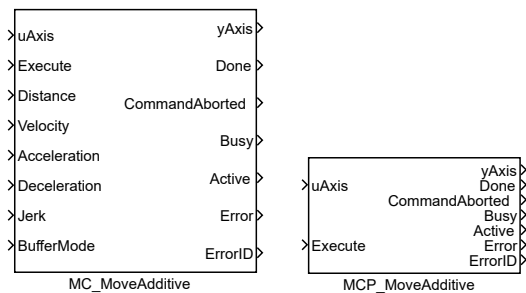
Example



MC_MoveAdditive, MCP_MoveAdditive – Move to position (relative to previous motion)

Block Symbols

Licence: [MOTION CONTROL](#)



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

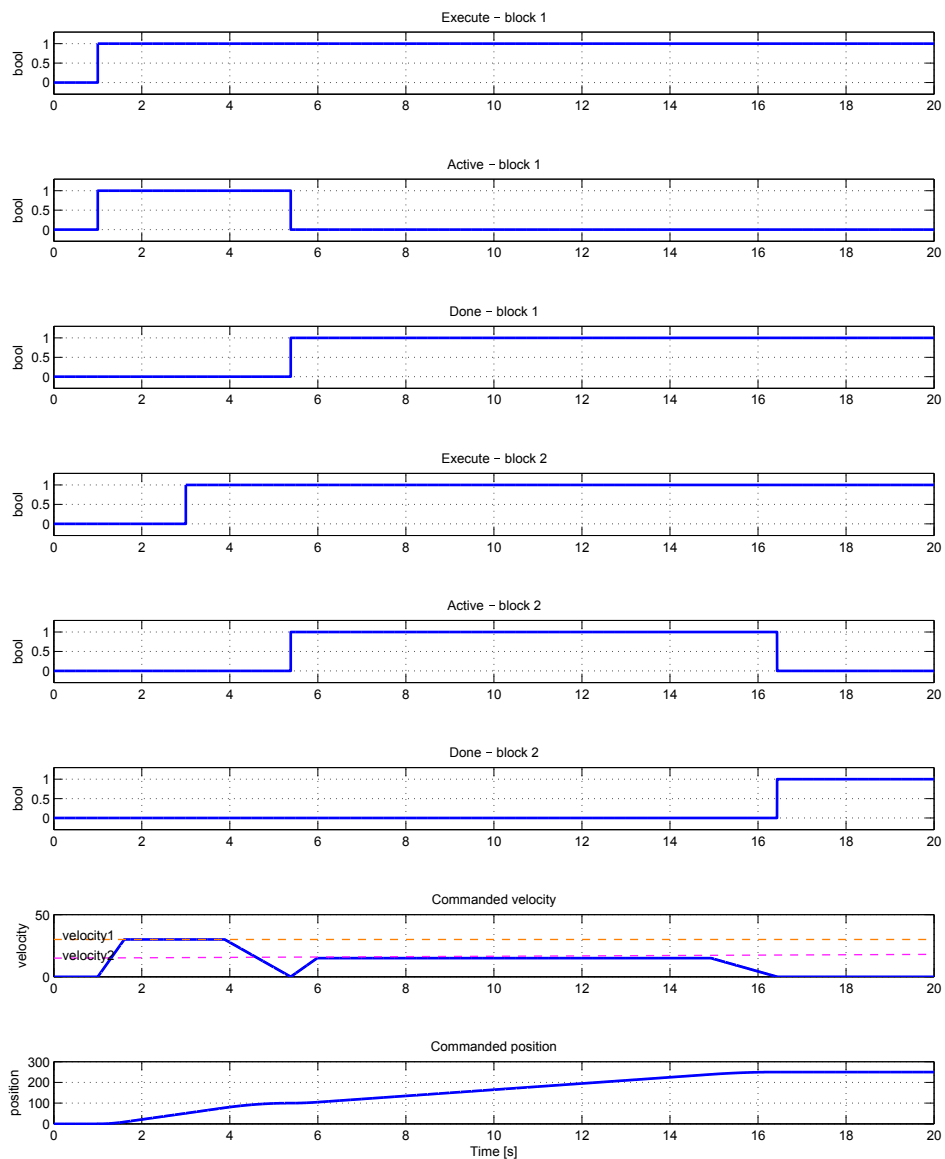
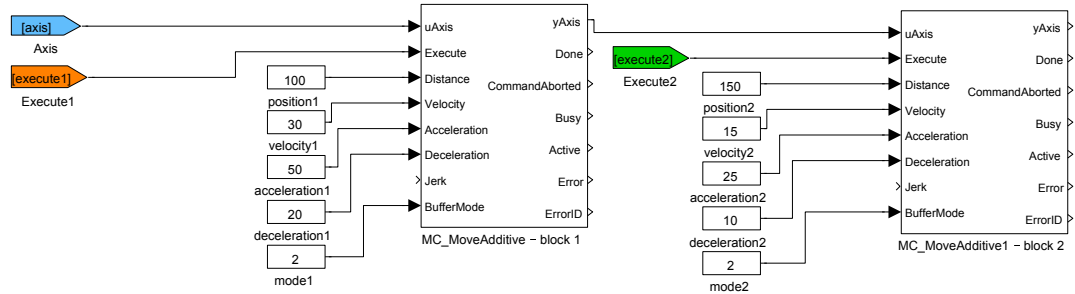
uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Distance	Requested target distance (relative to start point) [unit]	Double (F64)

Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s ²]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	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-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	

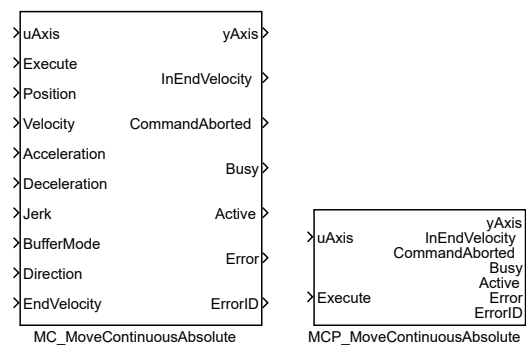
Example



MC_MoveContinuousAbsolute, MCP_MoveContinuousAbsolute – Move to position (absolute coordinate)

Block Symbols

Licence: [MOTION CONTROL](#)



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](#).

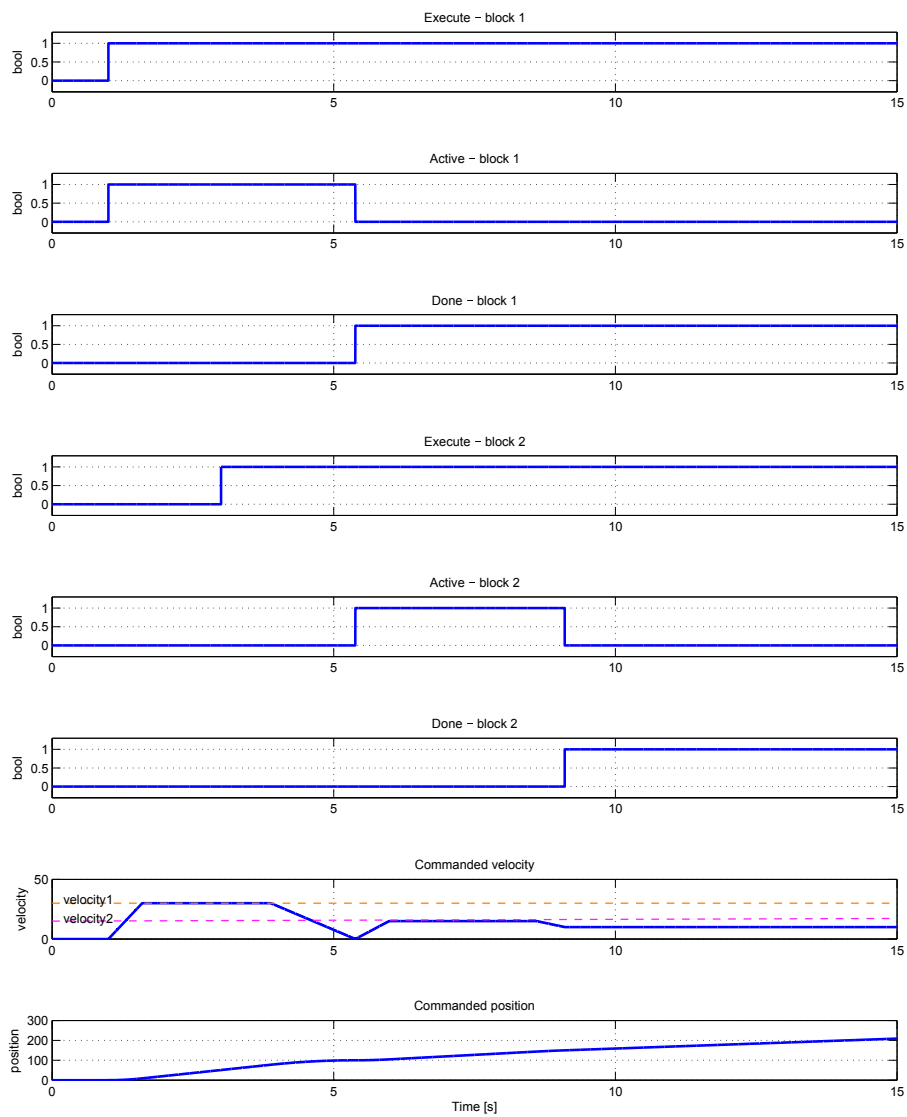
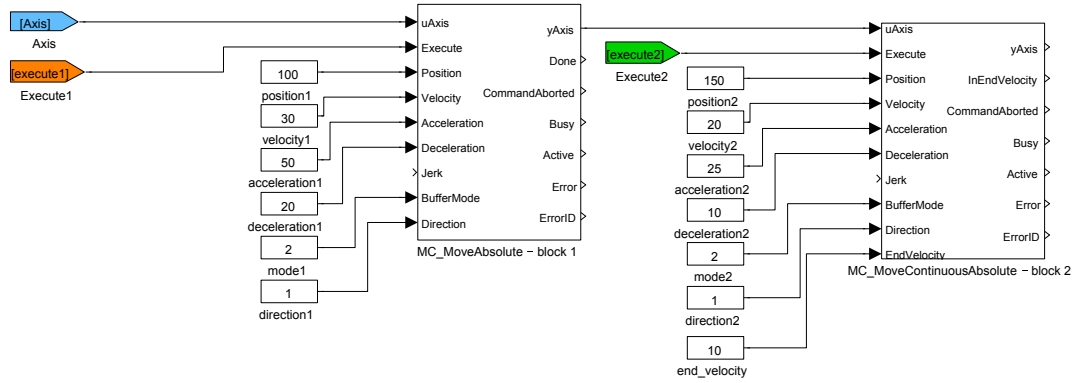
Inputs

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
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/s ²]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	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)	
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)	Reference
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	

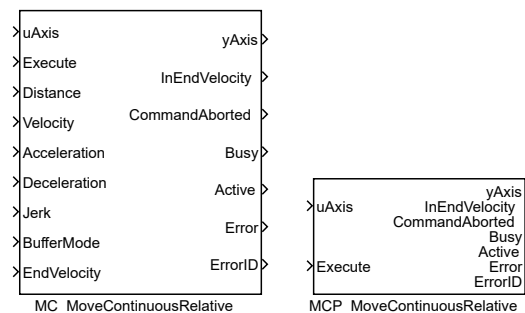
Example



MC_MoveContinuousRelative, MCP_MoveContinuousRelative – Move to position (relative to previous motion)

Block Symbols

Licence: [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

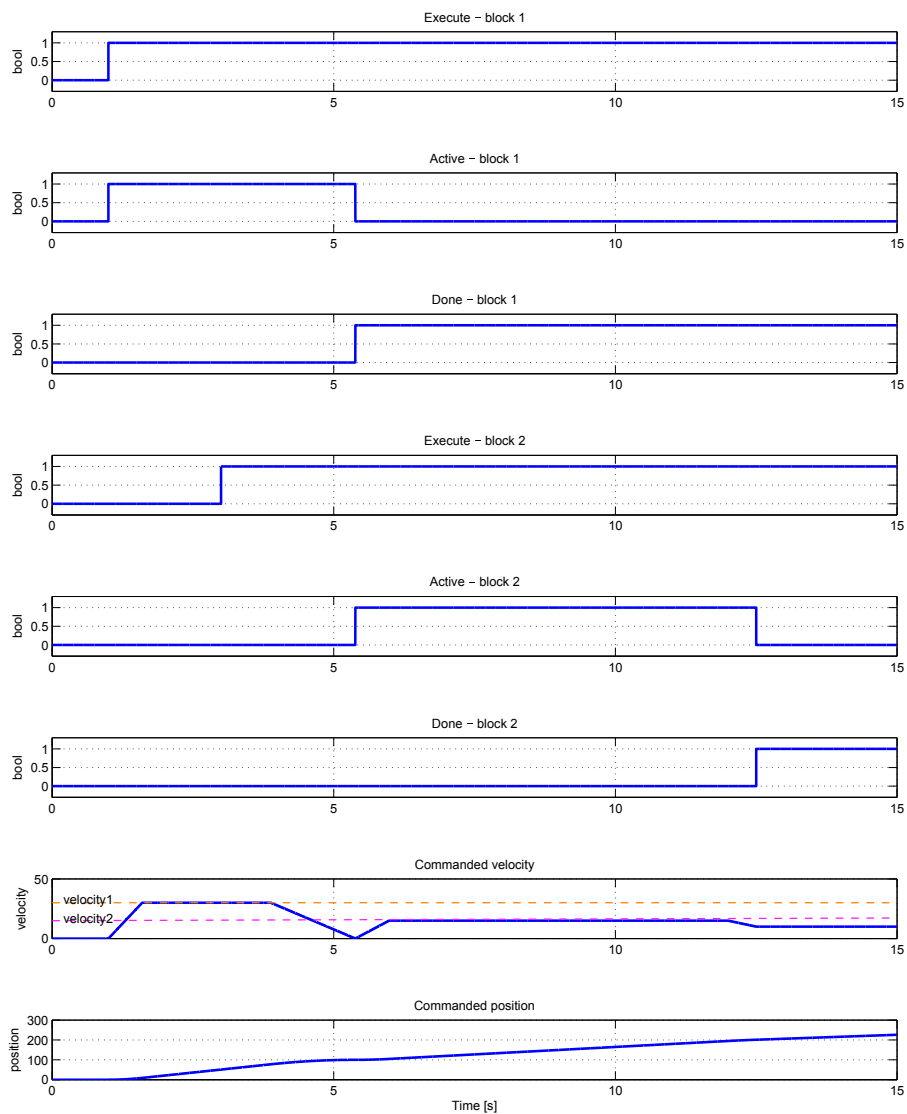
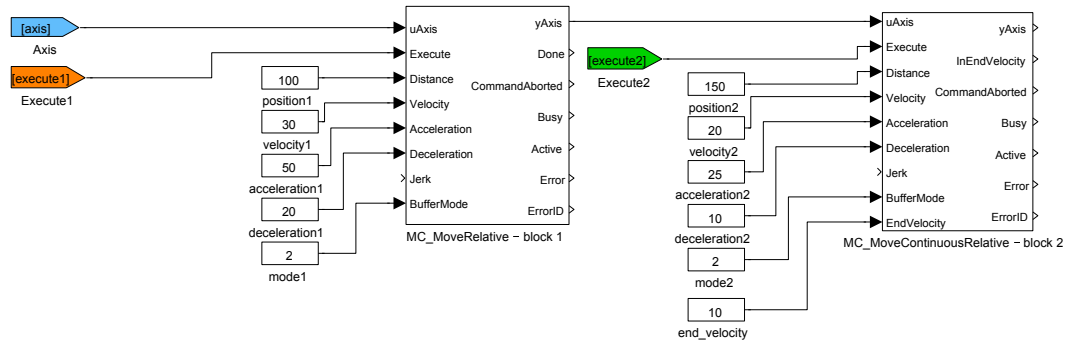
uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Distance	Requested target distance (relative to execution point) [unit]	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s ²]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	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)	
EndVelocity	End velocity	Long (I32)

Outputs

yAxis	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	Reference
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	

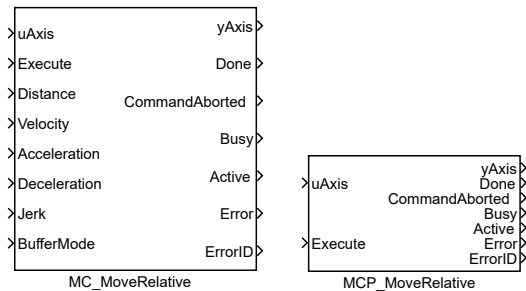
Example



MC_MoveRelative, MCP_MoveRelative – Move to position (relative to execution point)

Block Symbols

Licence: [MOTION CONTROL](#)



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

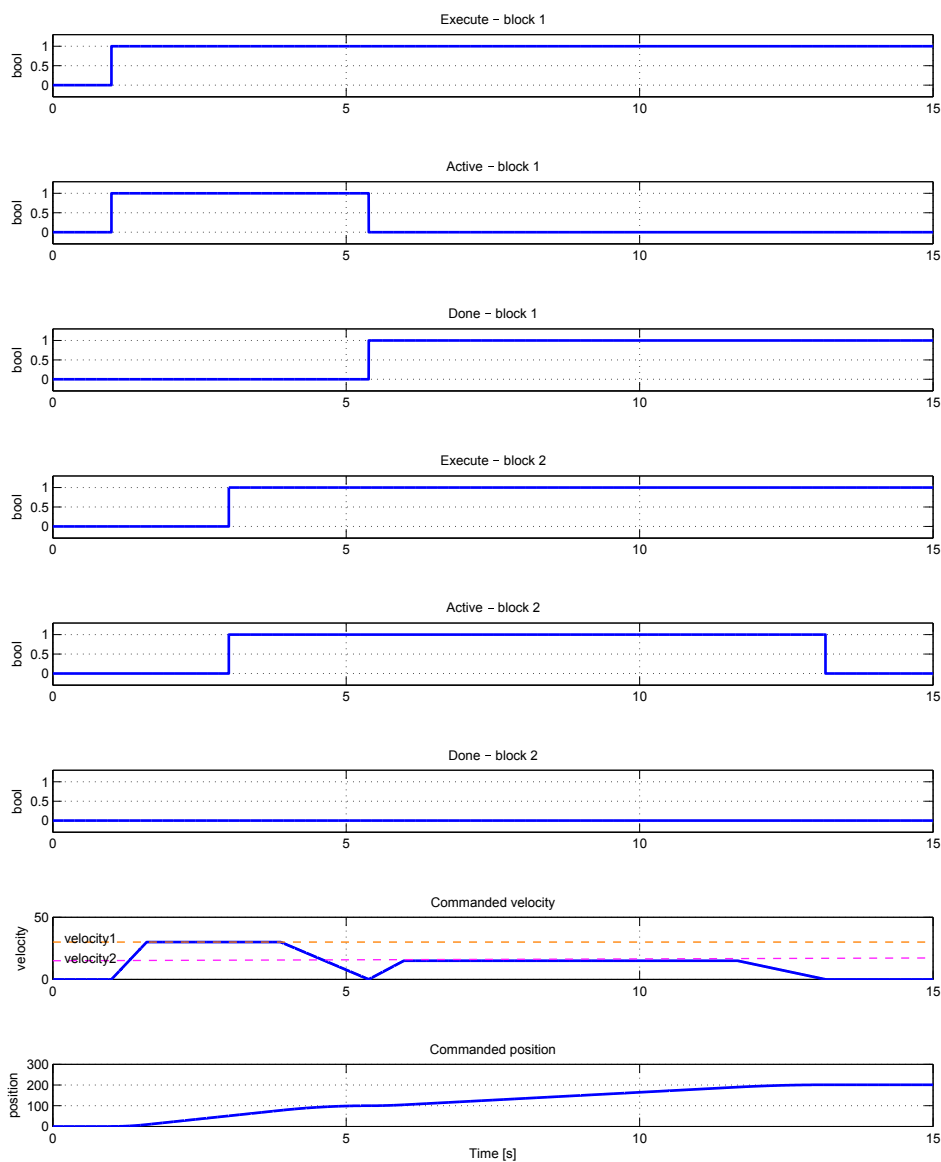
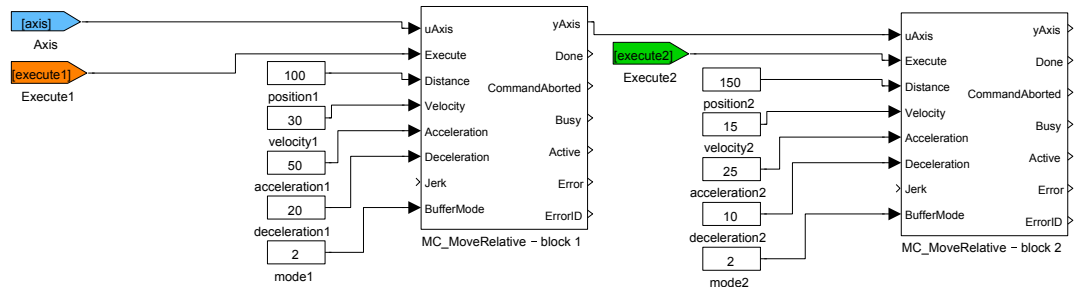
uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Distance	Requested target distance (relative to execution point) [unit]	Double (F64)

Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [[unit/s ²]	Double (F64)
Deceleration	Maximal allowed deceleration [[unit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [[unit/s ³]	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-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	

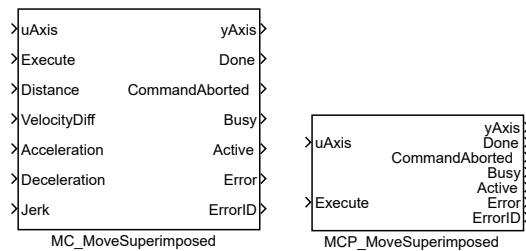
Example



MC_MoveSuperimposed, MCP_MoveSuperimposed – Superimposed move

Block Symbols

Licence: [MOTION CONTROL](#)



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).

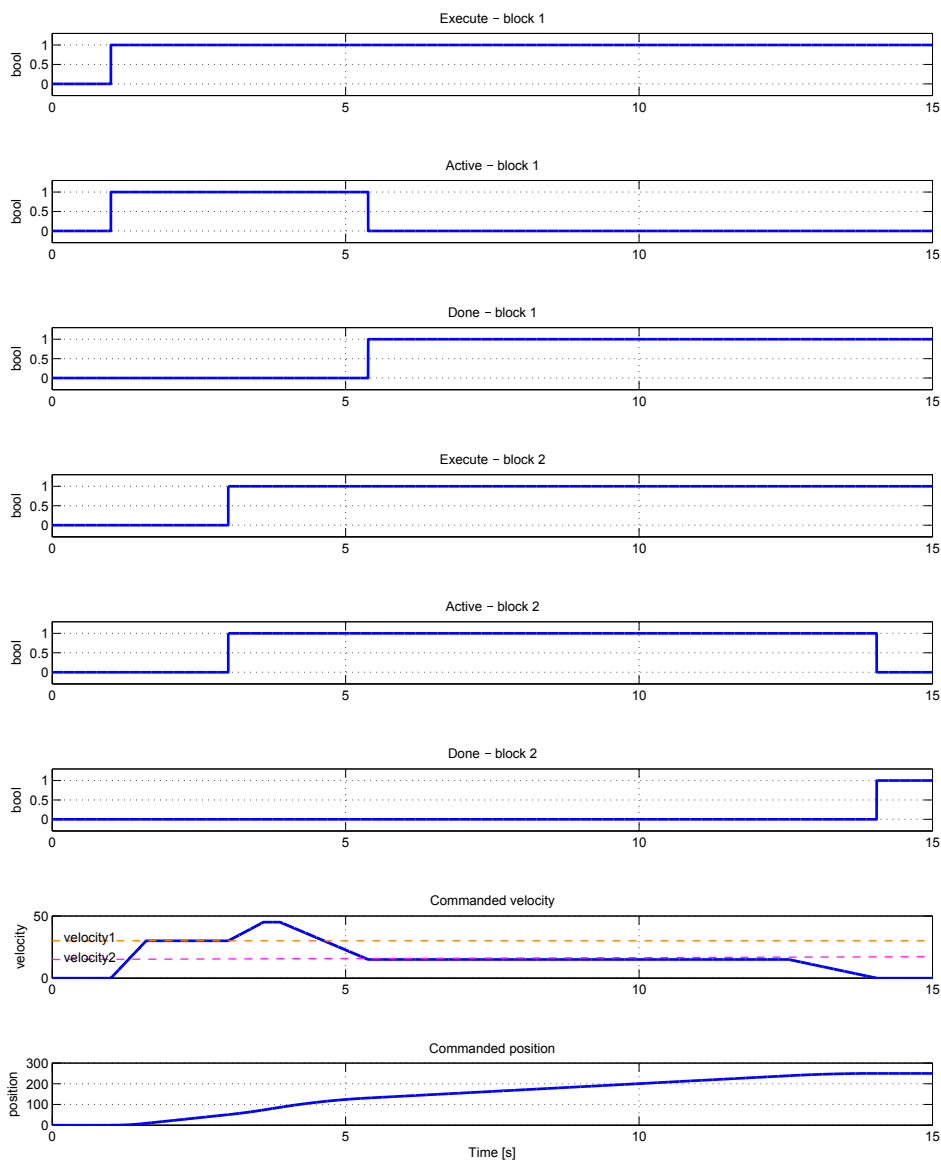
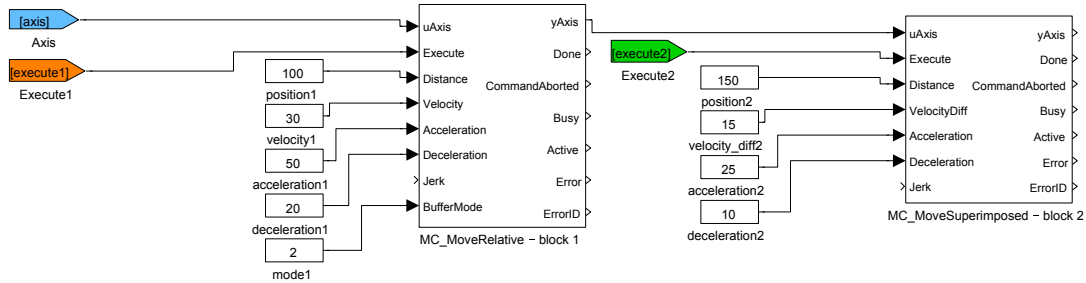
Inputs

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Distance	Requested target distance (relative to execution point) [unit]	Double (F64)
VelocityDiff	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s ²]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	Double (F64)

Outputs

yAxis	Axis reference (only RM_Axis.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	

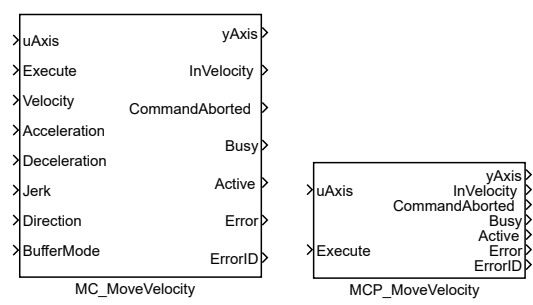
Example



MC_MoveVelocity, MCP_MoveVelocity – Move with constant velocity

Block Symbols

Licence: [MOTION CONTROL](#)



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

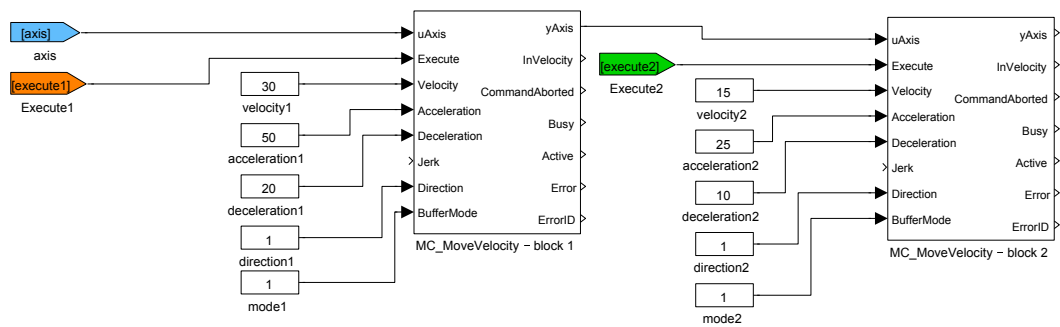
<code>uAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	Reference
<code>Execute</code>	The block is activated on rising edge	Bool
<code>Velocity</code>	Maximal allowed velocity [unit/s]	Double (F64)
<code>Acceleration</code>	Maximal allowed acceleration [unit/s ²]	Double (F64)
<code>Deceleration</code>	Maximal allowed deceleration [unit/s ²]	Double (F64)
<code>Jerk</code>	Maximal allowed jerk [unit/s ³]	Double (F64)
<code>Direction</code>	Direction of movement (cyclic axis or special case only)	Long (I32)
	1 Positive	
	2 Shortest	
	3 Negative	
	4 Current	

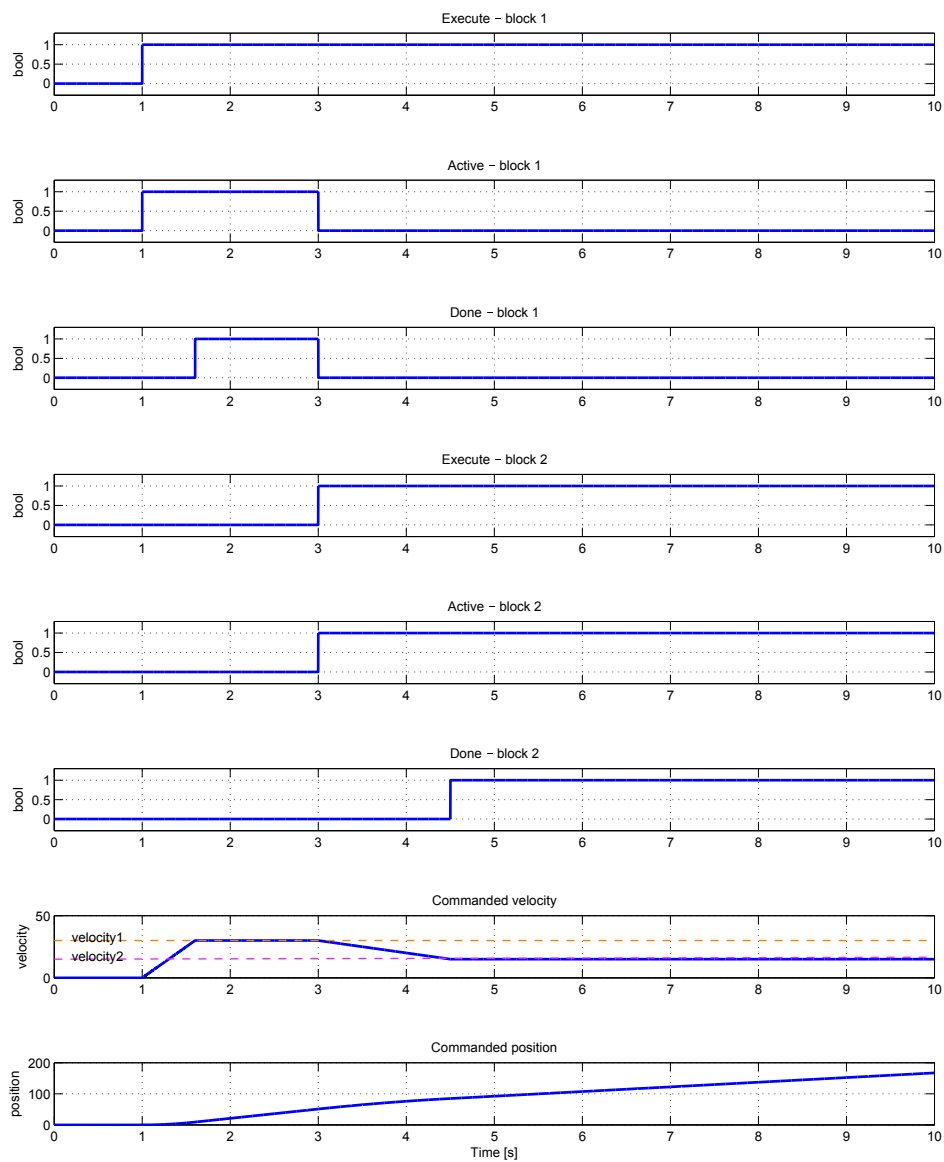
BufferMode	Buffering mode	Long (I32)
1 Aborting (start immediately)	
2 Buffered (start after finish of previous motion)	
3 Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)	
4 Blending high (start after finishing the previous 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 connections are allowed)	Reference
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	

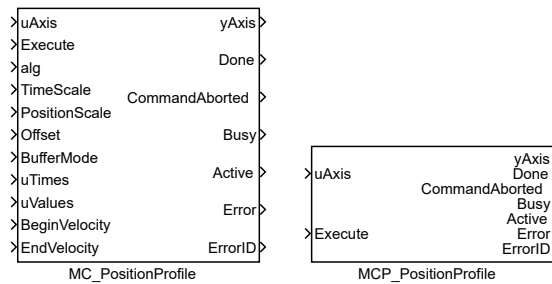
Example





MC_PositionProfile, MCP_PositionProfile – Position profile

Block Symbols

Licence: [MOTION CONTROL](#)

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 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_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

<code>uAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	Reference
<code>Execute</code>	The block is activated on rising edge	Bool
<code>TimeScale</code>	Overall scale factor in time	Double (F64)
<code>PositionScale</code>	Overall scale factor in value	Double (F64)
<code>Offset</code>	Overall profile offset in value	Double (F64)
<code>BufferMode</code>	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)	

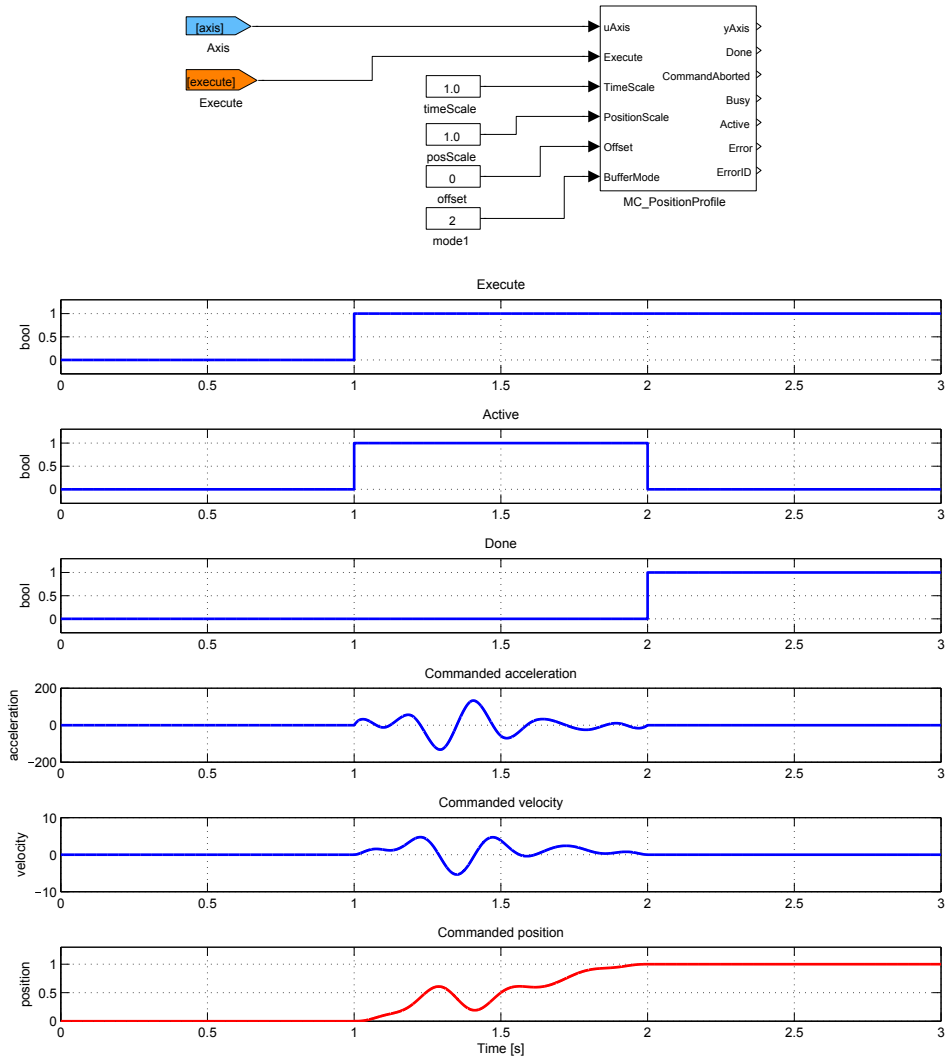
Outputs

<code>yAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	Reference
<code>Done</code>	Algorithm finished	Bool
<code>CommandAborted</code>	Algorithm was aborted	Bool
<code>Busy</code>	Algorithm not finished yet	Bool
<code>Active</code>	The block is controlling the axis	Bool
<code>Error</code>	Error occurred	Bool
<code>ErrorID</code>	Result of the last operation	Error
	i REXYGEN general error	

Parameters

alg	Algorithm for interpolation	⊙2	Long (I32)
	1 Sequence of time/value pairs		
	2 Sequence of equidistant values		
	3 Spline		
	4 Equidistant spline		
nmax	Number of profile segments	⊙3	Long (I32)
times	Times when segments are switched		Reference
values	Values or interpolating polynomial coefficients (a0, a1, a2, ...)		Reference

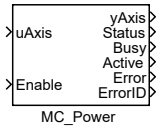
Example



MC_Power – Axis activation (power on/off)

Block Symbol

Licence: MOTION CONTROL



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 stopping sequence is activated and the status is switched to off/diabled when the motion stops (commanded velocity reaches zero value).

Inputs

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Enable	Block function is enabled	Bool

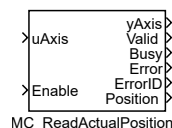
Outputs

yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Status	Effective state of the power stage	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 i REXYGEN general error	Error

MC_ReadActualPosition – Read actual position

Block Symbol

Licence: [MOTION CONTROL](#)



Function Description

The block `MC_ReadActualPosition` displays actual value of 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 value which is entered into all of the motion blocks as position input. In case that no absolute position encoder is used or the internal position is set in other way (e.g. via [MC_Home](#) block), the `CommandedPosition` output of the corresponding [RM_Axis](#) may display different value.

Inputs

<code>uAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	Reference
<code>Enable</code>	Block function is enabled	Bool

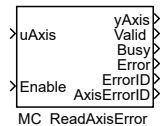
Outputs

<code>yAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	Reference
<code>Valid</code>	Output value is valid	Bool
<code>Busy</code>	Algorithm not finished yet	Bool
<code>Error</code>	Error occurred	Bool
<code>ErrorID</code>	Result of the last operation i REXYGEN general error	Error
<code>Position</code>	Actual absolute position	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

<code>uAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	Reference
<code>Enable</code>	Block function is enabled	Bool

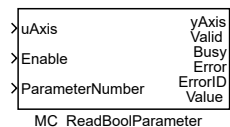
Outputs

<code>yAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	Reference
<code>Valid</code>	Output value is valid	Bool
<code>Busy</code>	Algorithm not finished yet	Bool
<code>Error</code>	Error occurred	Bool
<code>ErrorID</code>	Result of the last operation i REXYGEN general error	Error
<code>AxisErrorID</code>	Result of the last operation read from axis i REXYGEN general error	Error

MC_ReadBoolParameter – Read axis parameter (bool)

Block Symbol

Licence: [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

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Enable	Block function is enabled	Bool
ParameterNumber	Parameter ID	Long (I32)
	4 Enable sw positive limit	
	5 Enable sw negative limit	
	6 Enable position lag monitoring	

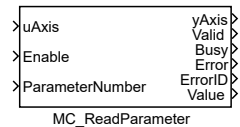
Outputs

yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Valid	Output value is valid	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
	i REXYGEN general error	
Value	Parameter value	Bool

MC_ReadParameter – Read axis parameter

Block Symbol

Licence: MOTION CONTROL



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

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Enable	Block function is enabled	Bool
ParameterNumber	Parameter ID	Long (I32)
	1 Commanded position	
	2 Positive sw limit switch	
	3 Negative sw limit switch	
	7 Maximal position lag	
	8 Maximal velocity (system)	
	9 Maximal velocity (appl)	
	10 Actual velocity	
	11 Commanded velocity	
	12 Maximal acceleration (system)	
	13 Maximal acceleration (appl.)	
	14 Maximal deceleration (system)	
	15 Maximal deceleration (appl.)	
	16 Maximal jerk	
	1000 .. Actual position	
	1001 .. Maximal torque/force	
	1003 .. Actual torque/force	
	1004 .. Commanded torque/force	

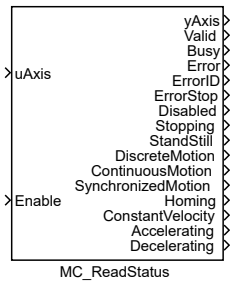
Outputs

yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Valid	Output value is valid	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i REXYGEN general error	Error
Value	Parameter value	Double (F64)

MC_ReadStatus – Read axis status

Block Symbol

Licence: MOTION CONTROL



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

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Enable	Block function is enabled	Bool

Outputs

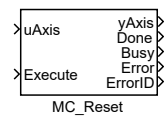
yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Valid	Output value is valid	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i REXYGEN general error	Error
ErrorStop	Axis is in the ErrorStop state	Bool
Disabled	Axis is in the Disabled state	Bool
Stopping	Axis is in the Stopping state	Bool
StandStill	Axis is in the StandStill state	Bool
DiscreteMotion	Axis is in the DiscreteMotion state	Bool
ContinuousMotion	Axis is in the ContinuousMotion state	Bool
SynchronizedMotion	Axis is in the SynchronizedMotion state	Bool
Homing	Axis is in the Homing state	Bool

<code>ConstantVelocity</code>	Axis is moving with constant velocity	<code>Bool</code>
<code>Accelerating</code>	Axis is accelerating	<code>Bool</code>
<code>Decelerating</code>	Axis is decelerating	<code>Bool</code>

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

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

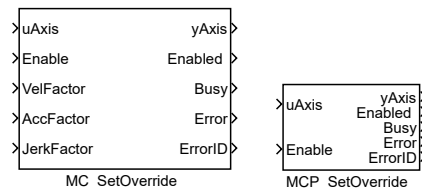
Outputs

yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
	i REXYGEN general error	

MC_SetOverride, MCP_SetOverride – Set override factors

Block Symbols

Licence: [MOTION CONTROL](#)



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

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	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)

Outputs

yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Enabled	Block function is enabled	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i REXYGEN general error	Error

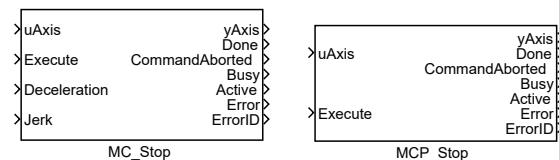
Parameter

diff	Deadband (difference for recalculation)	↓0.0 ↑1.0 ⊙0.1	Double (F64)
------	---	----------------	--------------

MC_Stop, MCP_Stop – Stopping a movement

Block Symbols

Licence: [MOTION CONTROL](#)



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 parameteres from [RM_Axis](#) or generates error-stop-sequence).

Inputs

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	Double (F64)

Outputs

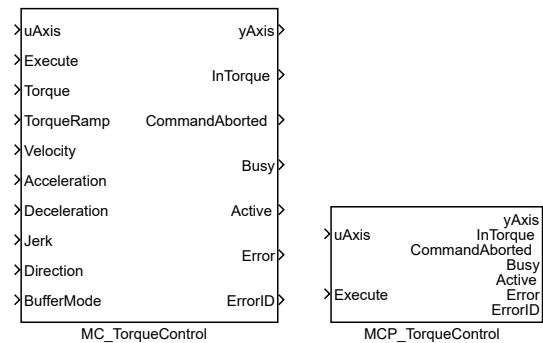
yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Done	Algorithm finished	Bool
CommandAborted	Algorithm was aborted	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool
Error	Error occurred	Bool

ErrorID	Result of the last operation	Error
i	REXYGEN general error	

MC_TorqueControl, MCP_TorqueControl – Torque/force control

Block Symbols

Licence: [MOTION CONTROL](#)



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

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Torque	Maximal allowed torque/force	Double (F64)
TorqueRamp	Maximal allowed torque/force ramp	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s ²]	Double (F64)
Deceleration	Maximal allowed deceleration [uunit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	Double (F64)
Direction	Direction of movement (cyclic axis or special case only)	Long (I32)
	1 Positive	
	2 Shortest	
	3 Negative	
	4 Current	

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)	

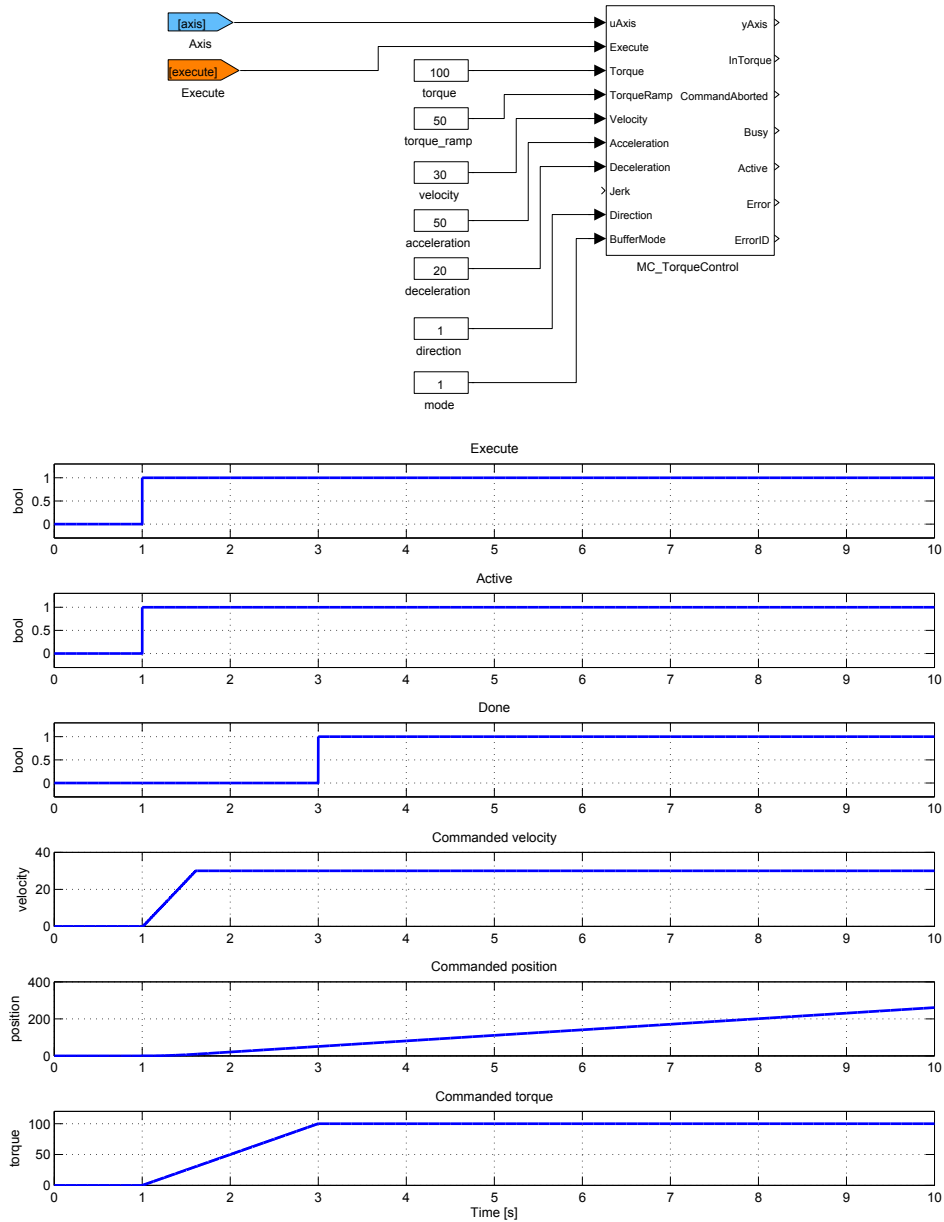
Outputs

yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
InTorque	Requested torque/force is 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	

Parameter

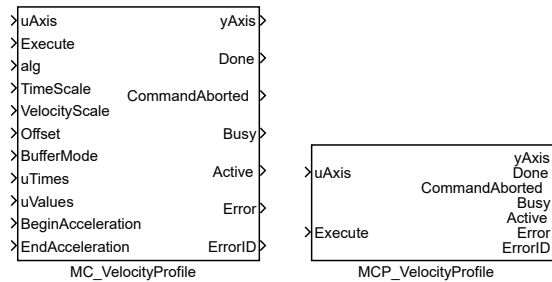
kma	Torque/force to acceleration ratio	Double (F64)
------------	------------------------------------	---------------------

Example



MC_VelocityProfile, MCP_VelocityProfile – Velocity profile

Block Symbols

Licence: [MOTION CONTROL](#)

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).

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

<code>uAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	Reference
<code>Execute</code>	The block is activated on rising edge	Bool
<code>TimeScale</code>	Overall scale factor in time	Double (F64)
<code>VelocityScale</code>	Overall scale factor in value	Double (F64)
<code>Offset</code>	Overall profile offset in value	Double (F64)
<code>BufferMode</code>	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)	

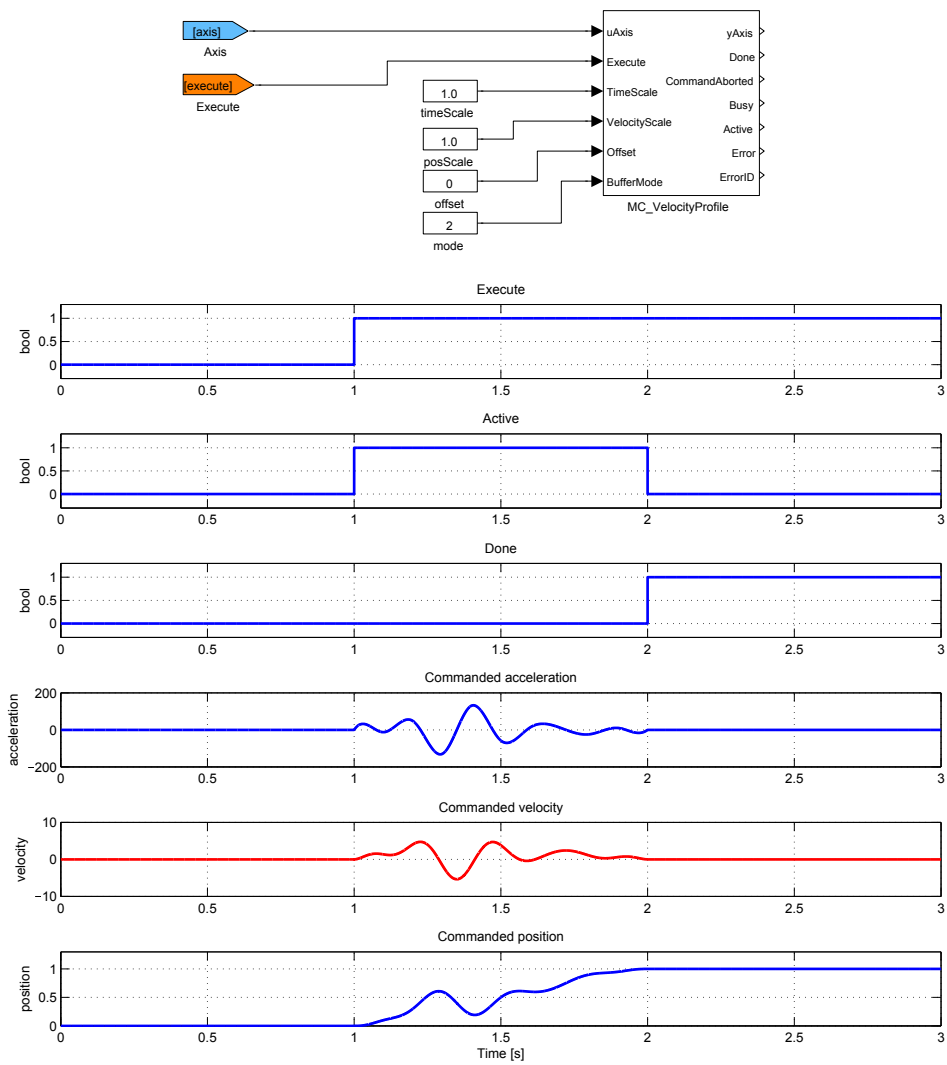
Outputs

<code>yAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	Reference
<code>Done</code>	Algorithm finished	Bool
<code>CommandAborted</code>	Algorithm was aborted	Bool
<code>Busy</code>	Algorithm not finished yet	Bool
<code>Active</code>	The block is controlling the axis	Bool
<code>Error</code>	Error occurred	Bool
<code>ErrorID</code>	Result of the last operation	Error
	i REXYGEN general error	

Parameters

alg	Algorithm for interpolation	⊙2	Long (I32)
	1 Sequence of time/value pairs		
	2 Sequence of equidistant values		
	3 Spline		
	4 Equidistant spline		
nmax	Number of profile segments	⊙3	Long (I32)
times	Times when segments are switched		Reference
values	Values or interpolating polynomial coefficients (a0, a1, a2, ...)		Reference

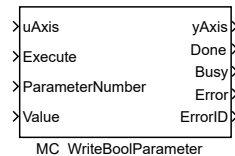
Example



MC_WriteBoolParameter – Write axis parameter (bool)

Block Symbol

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

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
ParameterNumber	Parameter ID	Long (I32)
	4 Enable sw positive limit	
	5 Enable sw negative limit	
	6 Enable position lag monitoring	
Value	Parameter value	Bool

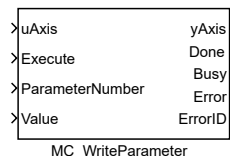
Outputs

yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
	i REXYGEN general error	

MC_WriteParameter – Write axis parameter

Block Symbol

Licence: [MOTION CONTROL](#)



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)	Reference
Execute	The block is activated on rising edge	Bool
ParameterNumber	Parameter ID	Long (I32)
	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)

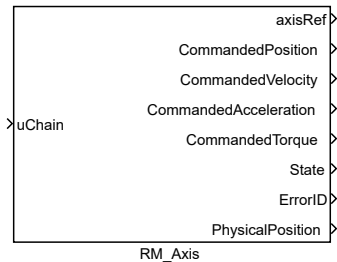
Outputs

yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
	i REXYGEN general error	

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 feedback signals are connected through the [RM_AxisSpline](#) block.

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 -703 error code (invalid state).

The parameters for limit velocity, acceleration and deceleration are twofold. One for application, e.g. limit value which could be set into motion blocks. This value could be exceeded in some cases. Second limit is for system. The system limits must be higher than application limits and it is never exceeded. If some motion block generate path, that exceed system limit, error stop sequence is activated.

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.

Inputs

uChain	Input is not used by the block. User can connect any signal to define order of block's execution	Long (I32)
--------	--	------------

Outputs

axisRef	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
CommandedPosition	Requested (commanded) position of the axis. The value is logical position that is put into the motion blocks. The position is different from PhysicalPosition if the axis is circular or homed.	Double (F64)
CommandedVelocity	Requested (commanded) velocity of the axis	Double (F64)
CommandedAcceleration	Requested (commanded) acceleration of the axis	Double (F64)
CommandedTorque	Requested (commanded) torque in the axis	Double (F64)
State	State of the axis	Long (I32)
	0 Disabled	
	1 Stand still	
	2 Homing	
	3 Discrete motion	
	4 Continuous motion	
	5 Synchronized motion	
	6 Coordinated motion	
	7 Stopping	
	8 Error stop	
	9 Drive error(simillar to Error stop, but fault is caused by external signal)	
ErrorID	Result of the last operation	Error
	i REXYGEN general error	
PhysicalPosition	Requested (commanded) position of the axis. The value is physical position that is put into the feedback controller. The position is different from CommandedPosition if the axis is circular or homed.	Double (F64)

Parameters

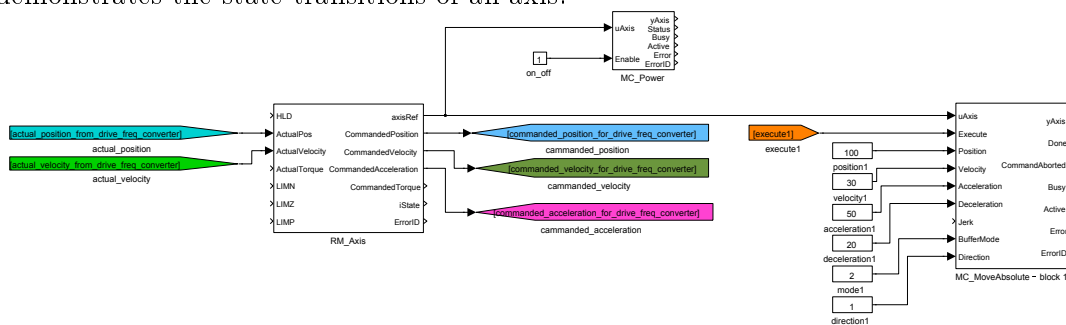
AxisType	Type of the axis	⊙1 Long (I32)
	1 Linear axis	
	2 Cyclic axis with cyclic position sensor	
	3 Cyclic axis with linear position sensor	
	4 Linear axis in testing mode. In this mode, position limits and limit switches are disabled. This type is intended for testing and commissioning, sometimes it can also be suitable for continuous operation on cyclic axes.	
EnableLimitPos	Enable positive position limit checking (e.g. if checked, MaxPosAppl is valid)	Bool
MaxPosAppl	Positive position limit for application (MC blocks). The value should be smaller then (before the) MaxPosSystem for linear axis. The value limit cyclic axis with linear sensor for few revolution (useful for robotic application) and must be bigger then (beyond the) MaxPosSystem .	Double (F64)

MaxPosSystem	Positive position limit for system. The value is never exceeded for linear axis. The value is end of revolution for cyclic axis.	Double (F64)
EnableLimitNeg	Enable negative position limit checking (e.g. if checked, MinPosAppl is valid)	Bool
MinPosAppl	Negative position limit for application (MC blocks) The value should be bigger then (before the) MinPosSystem for linear axis. The value limit cyclic axis with linear sensor for few revolution (useful for robotic application) and must be smaller then (beyond the) MinPosSystem.	Double (F64)
MinPosSystem	Negative position limit for system. The value is never exceeded for linear axis. The value is begin of revolution for cyclic axis.	Double (F64)
EnablePosLagMonitor	Enable monitoring of position lag (e.g. if checked, MaxPositionLag is valid)	Bool
MaxPositionLag	Maximal position lag. Any moving is stopped and the axis is switched into error stop state if different between PhysicalPosition and ActualPosition exceed this value.	Double (F64)
MaxVelocitySystem	Maximal allowed velocity for system	Double (F64)
MaxVelocityAppl	Maximal allowed velocity for application (MC blocks)	Double (F64)
MaxAccelerationSystem	Maximal allowed acceleration for system	Double (F64)
MaxAccelerationAppl	Maximal allowed acceleration for application (MC blocks)	Double (F64)
MaxDecelerationSystem	Maximal allowed deceleration for system	Double (F64)
MaxDecelerationAppl	Maximal allowed deceleration for application (MC blocks)	Double (F64)
DefaultJerk	Maximal recommended jerk [unit/s ³]. Real jerk is not checked and could overcome this value.	Double (F64)
MaxTorque	Maximal motor torque/force (0=not used)	Double (F64)
TorqueRatio	Torque-Acceleration ratio. The requested torque value is useful for feedback controller. The most block don't generate it. The requested torque value is computed as requested acceleration multiplied by this parameter.	Double (F64)
LoopDelay	delay between commanded and actual values[s] The actual position value is delayed from commanded value due communication with feedback controller, feedback loop, value interpolation and sampling period. The delay could be set into this parameter and then position lag is computed more precisely. (not yet implemented)	Double (F64)
StartMode	Some options when axis is enabled 1 start stopped 2 start tracking	⊙1 Long (I32)
HomingRequired	Homing is required before any move	Bool

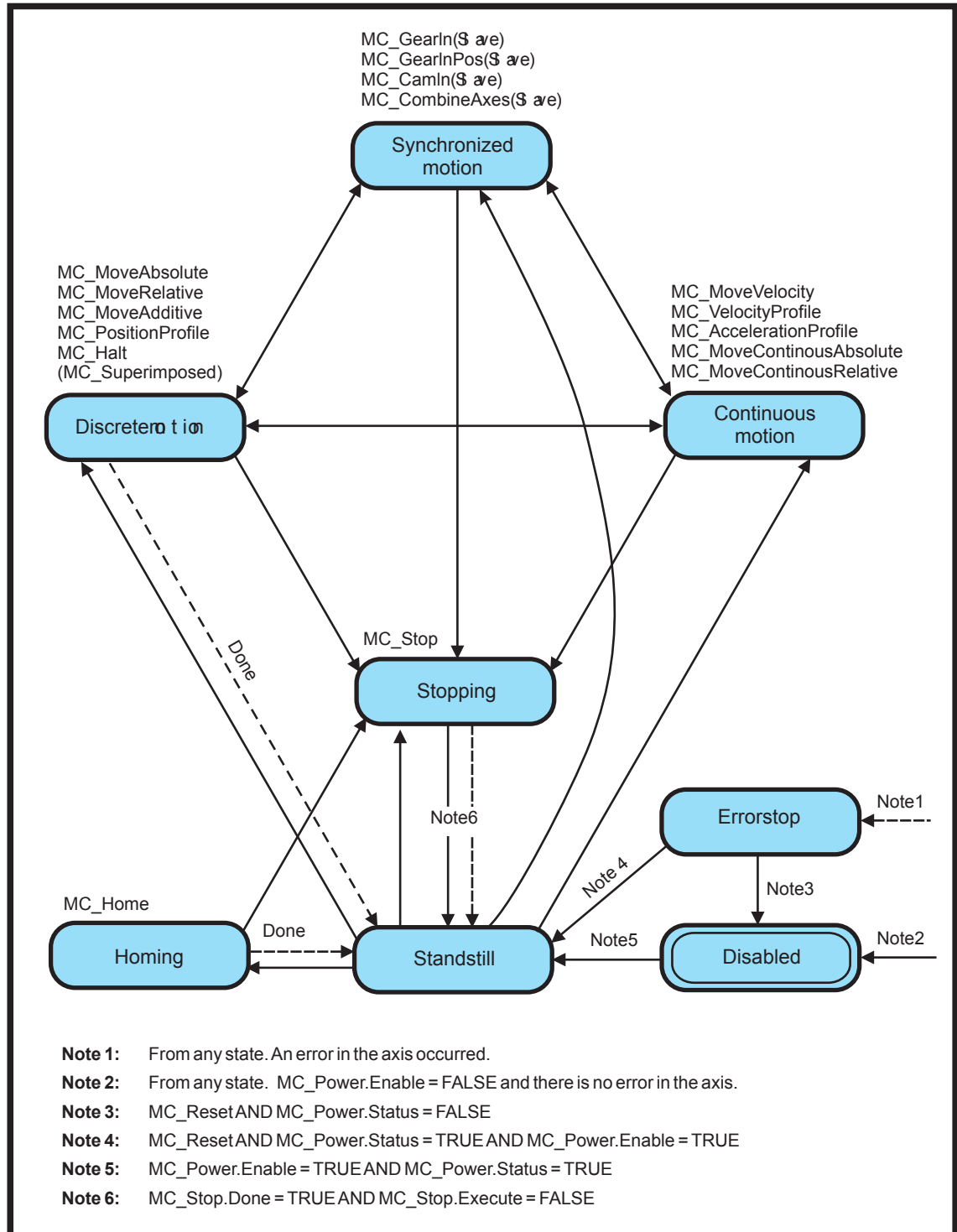
Example

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 **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. **MC_MoveAbsolute**). The trajectory of motion in form of desired position, velocity and acceleration is generated in output signals of the **RM_Axis** block. The reference values are provided to an actuator control loop which is implemented locally in **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 **MC_Reset** block prior to any subsequent motion command. The following state diagram demonstrates the state transitions of an axis.



Axis state transition diagram

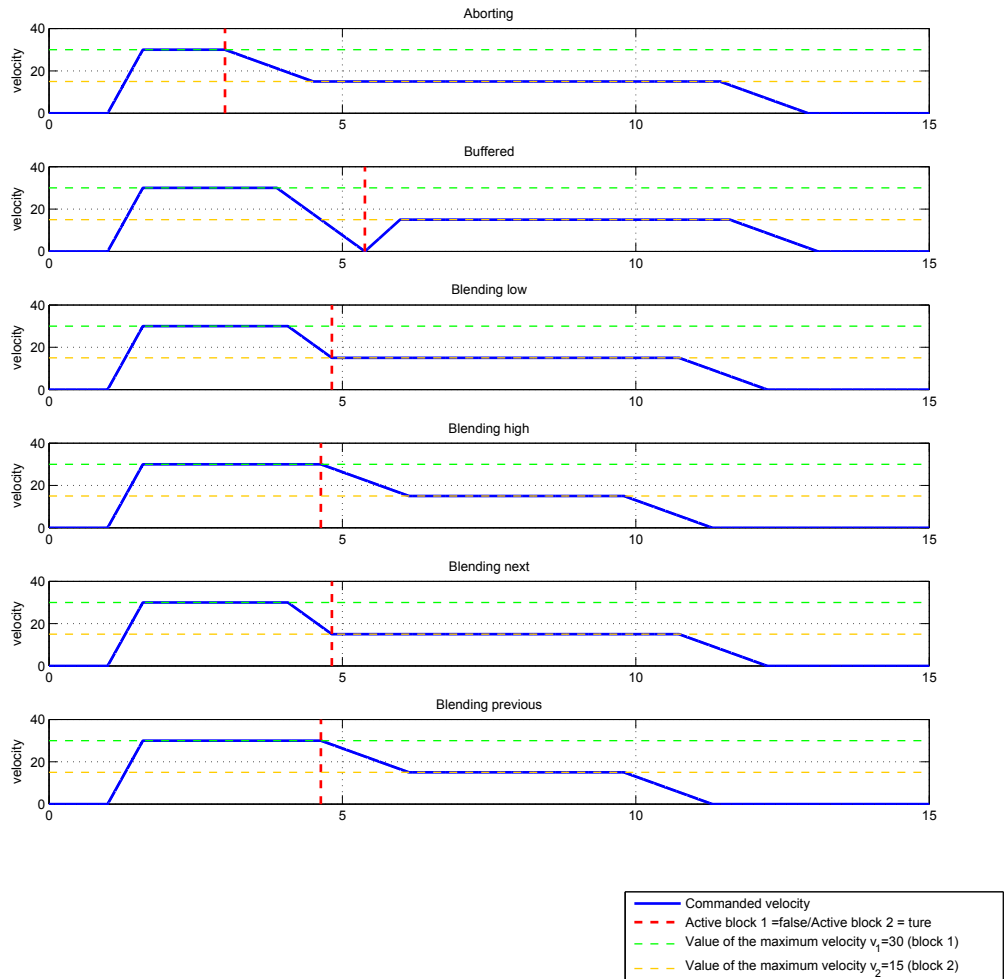


Motion blending

According to PLCOpen 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 PLCOpen specification.

Aborting	The new motion is executed immediately
Buffered	the new motion is executed immediately after finishing the previous one, there is no blending
Blending low	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
Blending high	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
Blending previous	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
Blending next	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

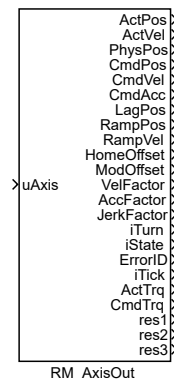
Illustration of blending modes



RM_AxisOut – Axis output

Block Symbol

Licence: [MOTION CONTROL](#)



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: 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.

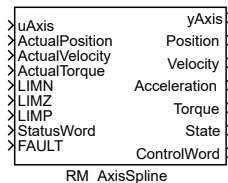
Inputs

<code>uAxis</code>	axis reference that must be connected to <code>axisRef</code> of the <code>RM_Axis</code> block (direct or indirect throw output <code>yAxis</code> of some other block)	Reference
--------------------	--	-----------

RM_AxisSpline – Commanded values interpolation

Block Symbol

Licence: [MOTION CONTROL](#)



Function Description

The purpose of the block is to connect a virtual axis (represented by the **RM_Axis** block) to the motor or rather the servo drive and transform virtual axis into physical one. It includes some independent functions that are covered by this block.

The block has commanded and actual (feedback) signals to connect feedback controller. It includes inputs **ActualPosition**, **ActualVelocity**, **ActualTorque** and outputs **Position**, **Velocity**, **Acceleration**, **Torque**.

The feedback controller or servo drive usually works with different units (position unit is usually in encoder's tick that is transformed by gear ratio). The **RM_AxisSpline** block transforms drive unit into axis logical unit. The function is controlled by the **DriveUnits** and **AxisUnits** parameters.

The servo drive often uses integer types for compute or communicate position, velocity and torque. Position can overflow range of integer value when motor is running one direction long time. The **RM_AxisSpline** block expects this situation and set correct position if feedback signal overflow from maximum integer value to minimal integer value. This feature is controlled by the **DriveBits** and must be also supported by the servo drive to work correctly.

The servo drive has different working state and operation mode and require some sequence to switch into operation mode where motor follow requested position. The most common standard for the mode and sequencing is CiA402. The **RM_AxisSpline** block support the CiA402 standard by the **StatusWord** input, the **ControlWord** output and the **DriveMode**, **DriveTimeout** parameters. The servo drive must be set to Cyclic Synchronous Position Mode (or mode with similar functionality). There is also possible to use Velocity Mode, but position loop regulator must be realized in control system (typically by a PIDU block).

There are a lot of motion control blocks which implement complicated algorithms so they require bigger sampling period (typical update rate is from 10 to 200 ms). 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 another task than other

motion control blocks with highest possible sampling period. It interpolates commanded position, velocity, acceleration and torque and generates smooth curve which is more suited for motor driver controllers.

There are many possibilities how to compute position (and velocity, acceleration, torque) between sampled points by slower task. This could be chosen by the `InterpolationMode` parameter, but torque is interpolated always by linear function. The supported methods include:

- 1: **linear**: Position is interpolated linearly, velocity as the derivative of position, acceleration is 0 (i.e., velocity is a piecewise constant function with jumps).
- 2: **cubic spline**: Position is a 3rd order polynomial calculated based on the position and velocity at the beginning and end of the interval; velocity is the derivative of position, acceleration is the derivative of velocity.
- 3: **quintic spline**: Position is a 5th order polynomial calculated based on the position, velocity, and acceleration at the beginning and end of the interval; velocity is the derivative of position, acceleration is the derivative of velocity.
- 4: **cubic approximation (B-spline)**: Position is a 3rd order polynomial calculated based on two positions before and two positions after the current interval; the interpolated function may not exactly pass through the given points; velocity is the derivative of position, acceleration is the derivative of velocity.
- 5: **quintic approximation (B-spline)**: Position is a 5th order polynomial calculated based on three positions before and three positions after the current interval; the interpolated function may not exactly pass through the given points; velocity is the derivative of position, acceleration is the derivative of velocity.
- 6: **all linear**: Position, velocity, and acceleration are independently interpolated linearly, i.e., velocity does not precisely correspond to the derivative of position, and acceleration does not precisely correspond to the derivative of velocity.
- 7: **all cubic**: Both position and velocity are interpolated by a 3rd order polynomial independently, i.e., velocity does not exactly correspond to the derivative of position.
- 8: *reserved for future use.*
- 9: *reserved for future use.*

Most simple is linear interpolation, but it leads to steps in velocity. Better possibility is higher order polynomial (e.g. 3th or 5th order). It generates a smooth curve, but leads to a huge acceleration if the original trajectory isn't the same polynomial. Drawback of polynomial interpolation could be solved by Bspline approximation, but it requires more samples and therefore bigger delay. Some original position values can be changed with this method.

Note 1: Because the execution time 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 signaled by states `Run1`, `Run2`, `Run3`.

Note 2: The interpolation functionality requires to put the block into different (faster) task than `RM_Axis`. For this reason, the block `RM_AxisSpline` has an internally safe solution for connecting axis references by the block `Inport` and `Outport` between different tasks.

Input

<code>uAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	Reference
<code>ActualPosition</code>	Current position of the axis (feedback) [drive unit]	Double (F64)
<code>ActualVelocity</code>	Current velocity of the axis (feedback) [drive unit/s]	Double (F64)
<code>ActualTorque</code>	Current torque in the axis (feedback)	Double (F64)
<code>LIMN</code>	Limit switch in negative direction	Bool
<code>LIMZ</code>	Absolute switch or reference pulse for homing	Bool
<code>LIMP</code>	Limit switch in positive direction	Bool
<code>StatusWord</code>	Status register for drive control according CiA402 specification	Long (I32)
<code>FAULT</code>	External fault signal	Bool

Outputs

<code>yAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	Reference
<code>Position</code>	Commanded interpolated position [drive unit]	Double (F64)
<code>Velocity</code>	Commanded interpolated velocity [drive unit/s]	Double (F64)
<code>Acceleration</code>	Commanded interpolated acceleration [drive unit/s/s]	Double (F64)
<code>Torque</code>	Commanded interpolated torque/force	Double (F64)

State	Interpolator state/error	Long (I32)
0	Off (interpolator is disabled, actual data put on output)	
1	Wait (not enough data in buffer, waiting)	
2	Run1 (interpolator running, data from first buffered interval)	
3	Run2 (interpolator running, data from second buffered interval)	
4	Run3 (interpolator running, data from third buffered interval)	
-1	Overflow (interpolation buffer overflow, the interpolation restarts automatically, but a bump in output values may occur)	
-2	Underflow (interpolation buffer underflow, the interpolation restarts automatically, but a bump in output values may occur)	
-3	Busy (data from RM_Axis cannot be read consistently, it usually indicates, that some task is overloaded)	
-4	Slow (the task with RM_AxisSpline has longer period then a task with RM_Axis)	
ControlWord	Control register for drive control according CiA402 specification	Long (I32)

Parameters

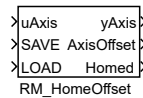
InterpolationMode	Algorithm for interpolation	⊙9 Long (I32)
1	linear	
2	cubic spline	
3	quintic spline	
4	cubic aproximation (B-spline)	
5	quintic aproximation (B-spline)	
6	all linear	
7	all cubic	
8	—	
9	—	
ReverseLimit	Invert meaning of LIMN, LIMZ and LIMP inputs	Bool
InterpolationMode	Drive control mode	⊙9 Long (I32)
1	Simplified CiA402 (only basic check of StatusWord , e.g. fault bit only, and direct switching of ControlWord , e.g. without sequencing)	
2	Strict CiA402 (full check of StatusWord in each state and full sequencing of ControlWord)	
DriveTimeout	Drive control response timeout [s] (for Strict CiA402 mode only)	Double (F64)
DriveBits	number of valid bits (negative value means signed number) in the Position output and the ActualPosition input	Long (I32)
	↓-64 ↑63 ⊙-32	

DriveUnits	Distance in drive units for position transformation (value correspond to AxisUnits)	Double (F64)
AxisUnits	Distance in axis units for position transformation (value correspond to DriveUnits)	Double (F64)
VelocityCalculate	if checked, the input ActualVelocity is ignored and velocity is calculated by actual position difference	Bool

RM_HomeOffset – * Homing by setting offset

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
SAVE	Set HomeOffset from the axis to the block parameter	Bool
LOAD	Set HomeOffset from the block parameter to the axis	Bool

Parameter

SavedOffset	Homing offset value	Double (F64)
--------------------	---------------------	---------------------

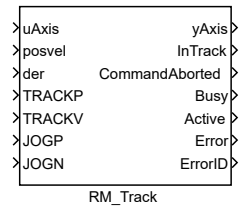
Output

yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
AxisOffset	Current home offset in axis	Bool
Homed	Axis homed flag	Bool

RM_Track – Tracking and inching

Block Symbol

Licence: [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 though the [MC_PositionProfile](#) block is better if whole path is known.

If the input **JOGP** 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 **JOGP** 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 **JOGP**, 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

uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
posvel	Requested target position or velocity [unit]	Double (F64)
TRACKP	Position tracking mode	Bool
TRACKV	Velocity tracking mode	Bool
JOGP	Moving positive direction mode	Bool
JOGN	Moving negative direction mode	Bool

Parameters

<code>pv</code>	Maximal allowed velocity [unit/s]	Double (F64)
<code>pa</code>	Maximal allowed acceleration [unit/s ²]	Double (F64)
<code>pd</code>	Maximal allowed deceleration [unit/s ²]	Double (F64)
<code>pj</code>	Maximal allowed jerk [unit/s ³]	Double (F64)
<code>iLen</code>	Length of buffer for estimation	⊙10 Long (I32)

Outputs

<code>yAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	Reference
<code>InTrack</code>	Requested position is reached	Bool
<code>CommandAborted</code>	Algorithm was aborted	Bool
<code>Busy</code>	Algorithm not finished yet	Bool
<code>Active</code>	The block is controlling the axis	Bool
<code>Error</code>	Error occurred	Bool
<code>ErrorID</code>	Result of the last operation	Error
	<code>i</code> REXYGEN general error	

Chapter 21

MC_MULTI – Motion control - multi axis blocks

Contents

MCP_CamIn – * Engage the cam	745
MCP_CamTableSelect – Cam definition	747
MCP_CombineAxes – * Combine the motion of 2 axes into a third axis	749
MCP_GearIn – * Engage the master/slave velocity ratio	751
MCP_GearInPos – * Engage the master/slave velocity ratio in defined position	753
MCP_PhasingAbsolute – * Create phase shift (absolute coordinate)	755
MCP_PhasingRelative – * Create phase shift (relative to previous motion)	757
MC_CamIn, MCP_CamIn – Engage the cam	759
MC_CamOut – Disengage the cam	763
MC_CombineAxes, MCP_CombineAxes – Combine the motion of 2 axes into a third axis	765
MC_GearIn, MCP_GearIn – Engage the master/slave velocity ratio	768
MC_GearInPos, MCP_GearInPos – Engage the master/slave velocity ratio in defined position	771
MC_GearOut – Disengage the master/slave velocity ratio	776
MC_PhasingAbsolute, MCP_PhasingAbsolute – Phase shift in synchronized motion (absolute coordinates)	778
MC_PhasingRelative, MCP_PhasingRelative – Phase shift in synchronized motion (relative coordinates)	781

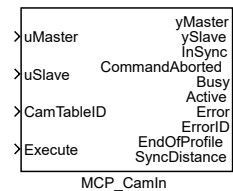
The MC_MULTI library is specialized for multi-axis motion control. It includes blocks like [MC_CombineAxes](#) for synchronizing multiple axes, [MC_GearIn](#) and [MC_GearOut](#) for gearing operations, and [MC_PhasingAbsolute](#), [MC_PhasingRelative](#) for precise axis

phasing. The library offers `MC_CamIn` and `MC_CamOut` for camming functionalities, allowing complex motion profiles to be followed. Additionally, `MCP_CamTableSelect` provides flexibility in selecting cam tables, and `MC_GearInPos` enables position-based gearing. This library is essential for advanced applications requiring coordinated motion control across multiple axes.

MCP_CamIn – * Engage the cam

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uMaster	Master axis reference	Reference
uSlave	Slave axis reference	Reference
CamTableID	Cam table reference (connect to MCP_CamTableSelect.CamTableID)	Reference
Execute	The block is activated on rising edge	Bool

Parameter

MasterOffset	Offset in cam table on master side [unit]	Double (F64)
SlaveOffset	Offset in cam table on slave side [unit]	Double (F64)
MasterScaling	Overall scaling factor in cam table on master side	⊙1.0 Double (F64)
SlaveScaling	Overall scaling factor in cam table on slave side	⊙1.0 Double (F64)
StartMode	Select relative or absolute cam table	⊙4 Long (I32)
	1 Master relative	
	2 Slave relative	
	3 Both relative	
	4 Both absolute	
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	
	3 Blending low	
	4 Blending high	
	5 Blending previous	
	6 Blending next	

RampIn	RampIn factor (0 = RampIn mode not used)	Double (F64)
---------------	--	---------------------

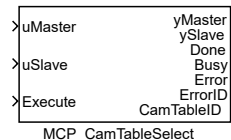
Output

yMaster	Master axis reference	Reference
ySlave	Slave axis reference	Reference
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 last operation	Error
	i REXYGEN error code	
EndOfProfile	Indicate end of cam profile (not periodic cam only)	Bool
SyncDistance	Position deviation of the slave axis from synchronized position	Double (F64)

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	Reference
ySlave	Slave axis reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
	i REXYGEN general error	
CamTableID	Cam table reference (connect to MC_CamIn .CamTableID)	Reference

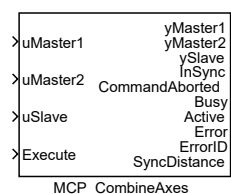
Parameters

alg	Algorithm for interpolation	⊙2	Long (I32)
	1 Sequence of time/value pairs		
	2 Sequence of equidistant values		
	3 Spline		
	4 Equidistant spline		
nmax	Number of profile segments	⊙3	Long (I32)
Periodic	Indicate periodic cam profile	⊙on	Bool
camname	Filename of special editor data file (filename is generated by system if parameter is empty)		String
mvalues	Master positions where segments are switched	⊙[0 30]	Double (F64)
sValues	Slave positions or interpolating polynomial coefficients (a0, a1, a2, ...)	⊙[0 100 100 0]	Double (F64)

MCP_CombineAxes – * Combine the motion of 2 axes into a third axis

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

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

Parameter

GearRatioNumeratorM1	Numerator for the gear factor for master axis 1	Long (I32)
		⊙1
GearRatioDenominatorM1	Denominator for the gear factor for master axis 1	Long (I32)
		⊙1
GearRatioNumeratorM2	Numerator for the gear factor for master axis 2	Long (I32)
		⊙1
GearRatioDenominatorM2	Denominator for the gear factor for master axis 2	Long (I32)
		⊙1
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	
	3 Blending low	
	4 Blending high	
	5 Blending previous	
	6 Blending next	

CombineMode	axis combination mode	⊙1	Long (I32)
	1 addition		
	2 subtraction		
RampIn	RampIn factor (0 = RampIn mode not used)		Double (F64)

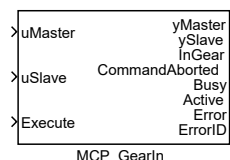
Output

yMaster1	First master axis reference		Reference
yMaster2	Second master axis reference		Reference
ySlave	Slave axis reference		Reference
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 last operation		Error
	i REXYGEN error code		
SyncDistance	Position deviation of the slave axis from synchronized position		Double (F64)

MCP_GearIn — * Engage the master/slave velocity ratio

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uMaster	Master axis reference	Reference
uSlave	Slave axis reference	Reference
Execute	The block is activated on rising edge	Bool

Parameter

RatioNumerator	Gear ratio Numerator	⊙1	Long (I32)
RatioDenominator	Gear ratio Denominator	⊙1	Long (I32)
Acceleration	Maximal allowed acceleration [unit/s ²]		Double (F64)
Deceleration	Maximal allowed deceleration [unit/s ²]		Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]		Double (F64)
BufferMode	Buffering mode	⊙1	Long (I32)
	1 Aborting		
	2 Buffered		
	3 Blending low		
	4 Blending high		
	5 Blending previous		
	6 Blending next		

Output

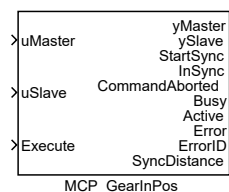
yMaster	Master axis reference	Reference
ySlave	Slave axis reference	Reference
InGear	Slave axis reached gearing ratio	Bool

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 last operation	Error
	i REXYGEN error code	

MCP_GearInPos – * Engage the master/slave velocity ratio in defined position

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uMaster	Master axis reference	Reference
uSlave	Slave axis reference	Reference
Execute	The block is activated on rising edge	Bool

Parameter

RatioNumerator	Gear ratio Numerator	⊙1	Long (I32)
RatioDenominator	Gear ratio Denominator	⊙1	Long (I32)
MasterSyncPosition	Master position for synchronization		Double (F64)
SlaveSyncPosition	Slave position for synchronization		Double (F64)
MasterStartDistance	Master distance for starting gear in procedure		Double (F64)
Velocity	Maximal allowed velocity [unit/s]		Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]		Double (F64)
Deceleration	Maximal allowed deceleration [unit/s^2]		Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]		Double (F64)
BufferMode	Buffering mode	⊙1	Long (I32)
	1 Aborting		
	2 Buffered		
	3 Blending low		
	4 Blending high		
	5 Blending previous		
	6 Blending next		

SyncMode	Synchronization mode (cyclic axes only)	⊙2	Long (I32)
	1 CatchUp		
	2 Shortest		
	3 SlowDown		

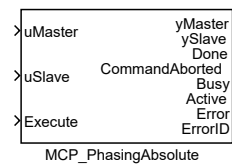
Output

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 last operation	Error
	i REXYGEN error code	
SyncDistance	Position deviation of the slave axis from synchronized position	Double (F64)

MCP_PhasingAbsolute — * Create phase shift (absolute coordinate)

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uMaster	Master axis reference	Reference
uSlave	Slave axis reference	Reference
Execute	The block is activated on rising edge	Bool

Parameter

PhaseShift	Requested phase shift (distance on master axis) for cam	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s^2]	Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	Double (F64)
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	

Output

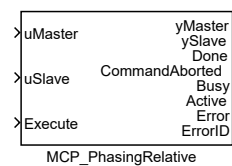
yMaster	Master axis reference	Reference
ySlave	Slave axis reference	Reference
Done	Algorithm finished	Bool
CommandAborted	Algorithm was aborted	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool

Error	Error occurred	Bool
ErrorID	Result of last operation	Error
	i REXYGEN error code	

MCP_PhasingRelative – * Create phase shift (relative to previous motion)

Block Symbol

Licence: [MOTION CONTROL](#)



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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uMaster	Master axis reference	Reference
uSlave	Slave axis reference	Reference
Execute	The block is activated on rising edge	Bool

Parameter

PhaseShift	Requested phase shift (distance on master axis) for cam	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s^2]	Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	Double (F64)
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	

Output

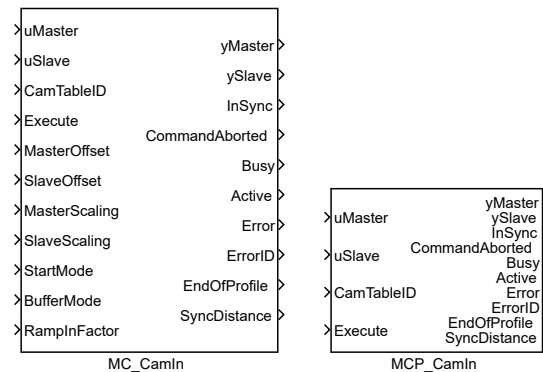
yMaster	Master axis reference	Reference
ySlave	Slave axis reference	Reference
Done	Algorithm finished	Bool
CommandAborted	Algorithm was aborted	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool

Error	Error occurred	Bool
ErrorID	Result of last operation	Error
	i REXYGEN error code	

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 $Cam(x)$, master axis position $PosM$ and slave axis position $PosS$, we obtain (for absolute relationship, without phasing): $PosS = Cam((PosM - MasterOffset)/MasterScaling) * SlaveScaling + SlaveOffset$. 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

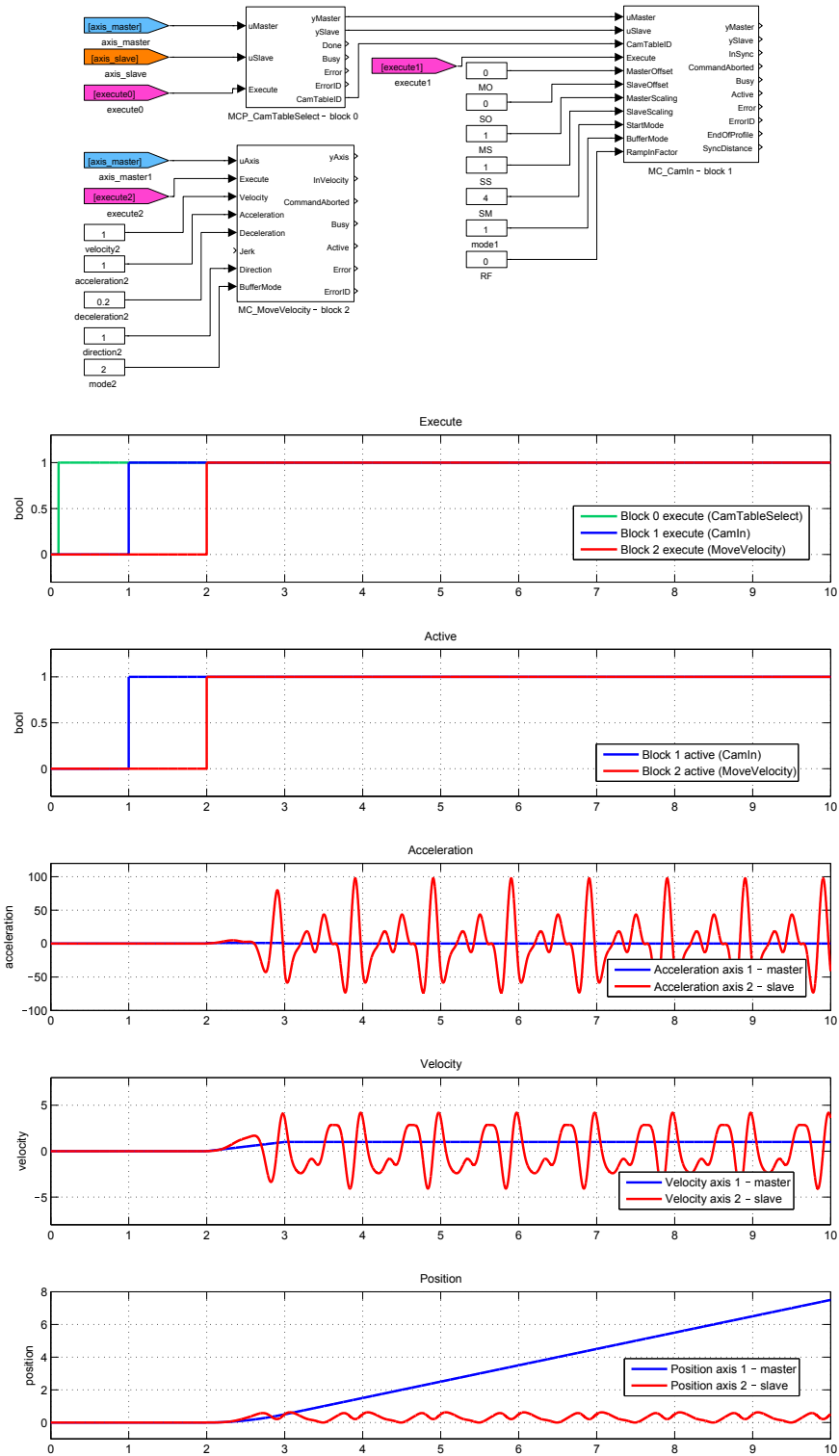
uMaster	Master axis reference	Reference
uSlave	Slave axis reference	Reference
CamTableID	Cam table reference (connect to <code>MCP_CamTableSelect.CamTableID</code>)	Reference
Execute	The block is activated on rising edge	Bool
MasterOffset	Offset in cam table on master side [unit]	Double (F64)
SlaveOffset	Offset in cam table on slave side [unit]	Double (F64)
MasterScaling	Overall scaling factor in cam table on master side	Double (F64)
SlaveScaling	Overall scaling factor in cam table on slave side	Double (F64)
StartMode	Select relative or absolute cam table	Long (I32)
	1 Master relative	
	2 Slave relative	
	3 Both relative	
	4 Both absolute	
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)	
RampIn	RampIn factor (0 = RampIn mode not used); average additive velocity (absolute value) during ramp-in process	Double (F64)

Outputs

yMaster	Master axis reference	Reference
ySlave	Slave axis reference	Reference
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	
EndOfProfile	Indicate end of cam profile (not periodic cam only)	Bool

SyncDistance Position deviation of the slave axis from synchronized position **Double** (F64)

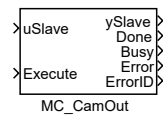
Example



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).

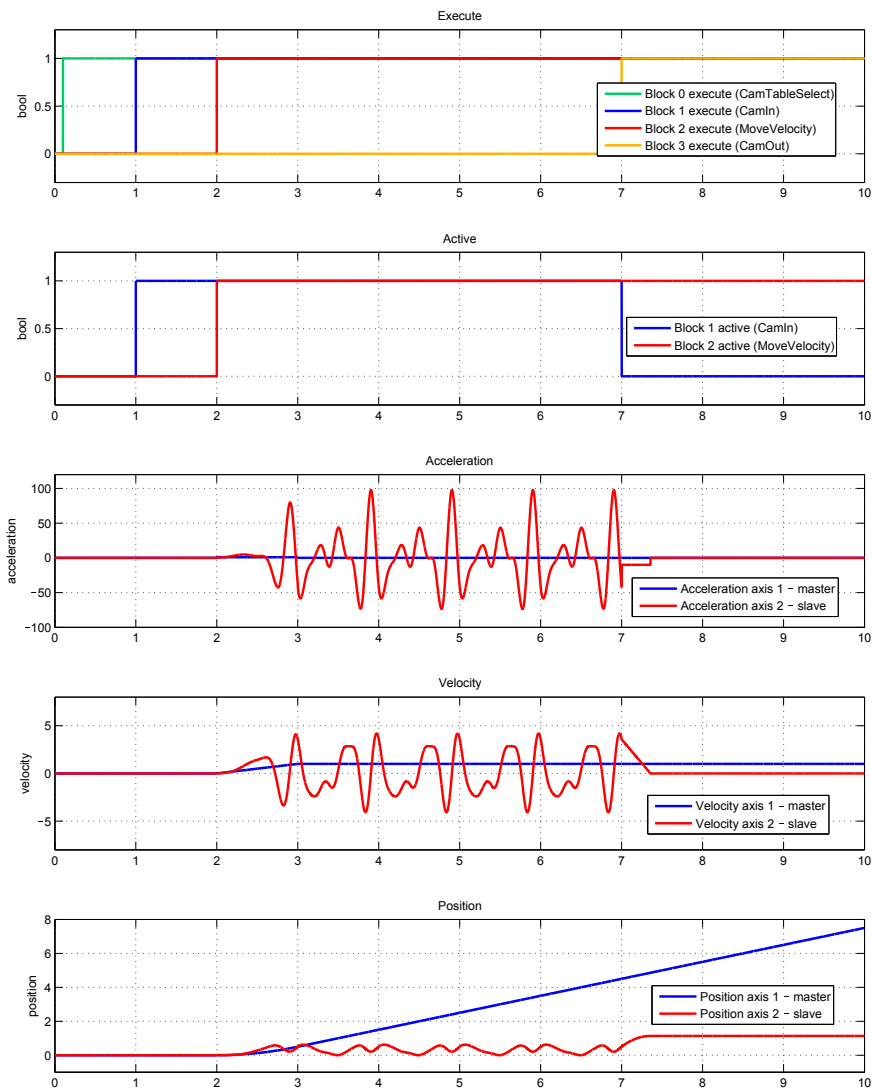
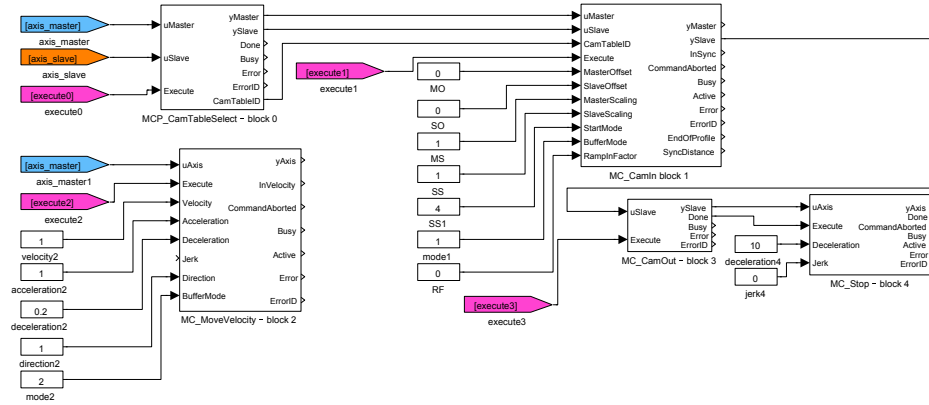
Inputs

<code>uSlave</code>	Slave axis reference	Reference
<code>Execute</code>	The block is activated on rising edge	Bool

Outputs

<code>ySlave</code>	Slave axis reference	Reference
<code>Done</code>	Algorithm finished	Bool
<code>Busy</code>	Algorithm not finished yet	Bool
<code>Error</code>	Error occurred	Bool
<code>ErrorID</code>	Result of the last operation	Error
	i REXYGEN general error	

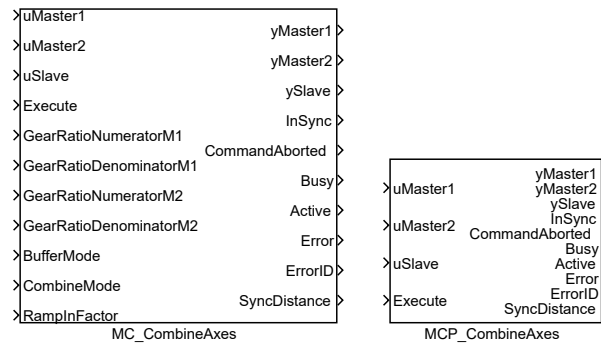
Example



MC_CombineAxes, MCP_CombineAxes – Combine the motion of 2 axes into a third axis

Block Symbols

Licence: [MOTION CONTROL](#)



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:

$$\begin{aligned} \text{SlavePosition} = & \text{Master1Position} \cdot \frac{\text{GearRatioNumeratorM1}}{\text{GearRatioDenominatorM1}} + \\ & + \text{Master2Position} \cdot \frac{\text{GearRatioNumeratorM2}}{\text{GearRatioDenominatorM2}} \end{aligned}$$

Negative number can be set in **GearRatio...** parameter to obtain the resulting slave movement in form of difference of master axes positions.

Inputs

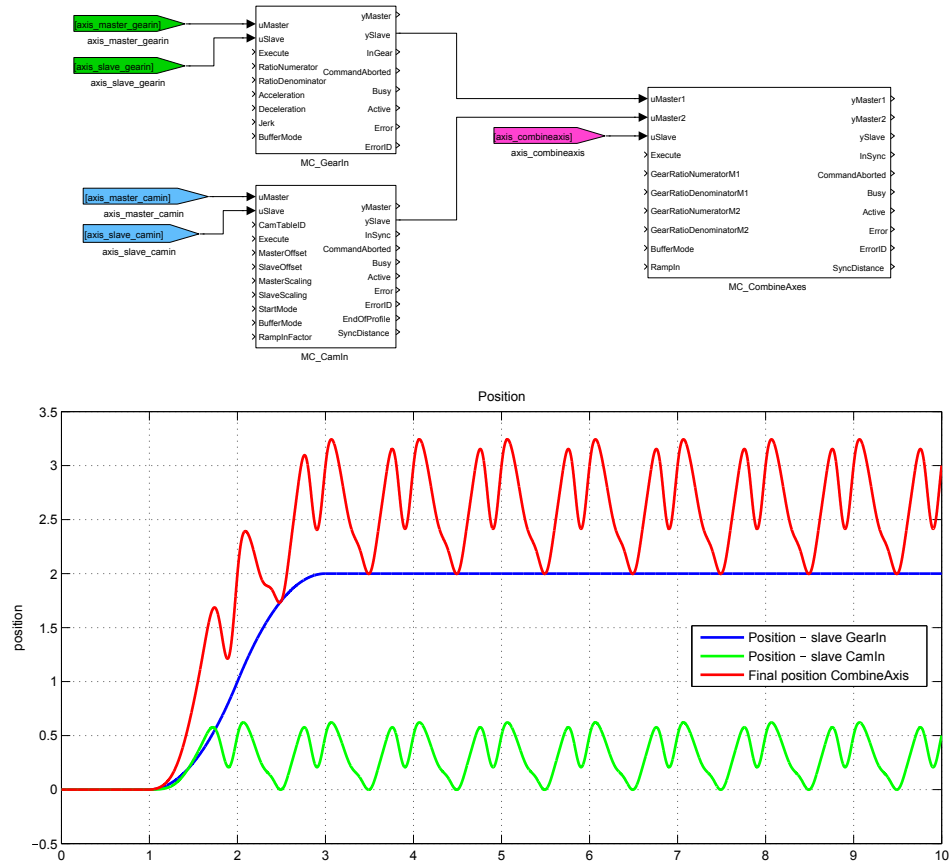
uMaster1	First master axis reference	Reference
uMaster2	Second master axis reference	Reference
uSlave	Slave axis reference	Reference
Execute	The block is activated on rising edge	Bool
GearRatioNumeratorM1	Numerator for the gear factor for master axis 1	Long (I32)
GearRatioDenominatorM1	Denominator for 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	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)	
RampIn	RampIn factor (0 = RampIn mode not used)	Double (F64)

Outputs

yMaster1	First master axis reference	Reference
yMaster2	Second master axis reference	Reference
ySlave	Slave axis reference	Reference
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)

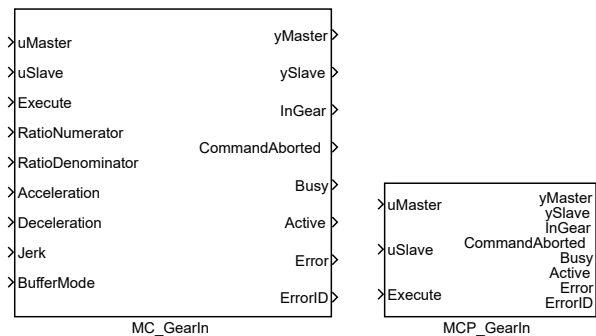
Example



MC_GearIn, MCP_GearIn – Engange the master/slave velocity ratio

Block Symbols

Licence: [MOTION CONTROL](#)



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 pre-set ratio between master and slave velocities is maintained. Considering the velocity of master axis $VelM$ and velocity of slave axis $VelS$, following relation holds (without phasing): $VelS = VelM * RatioNumerator / RatioDenominator$. 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

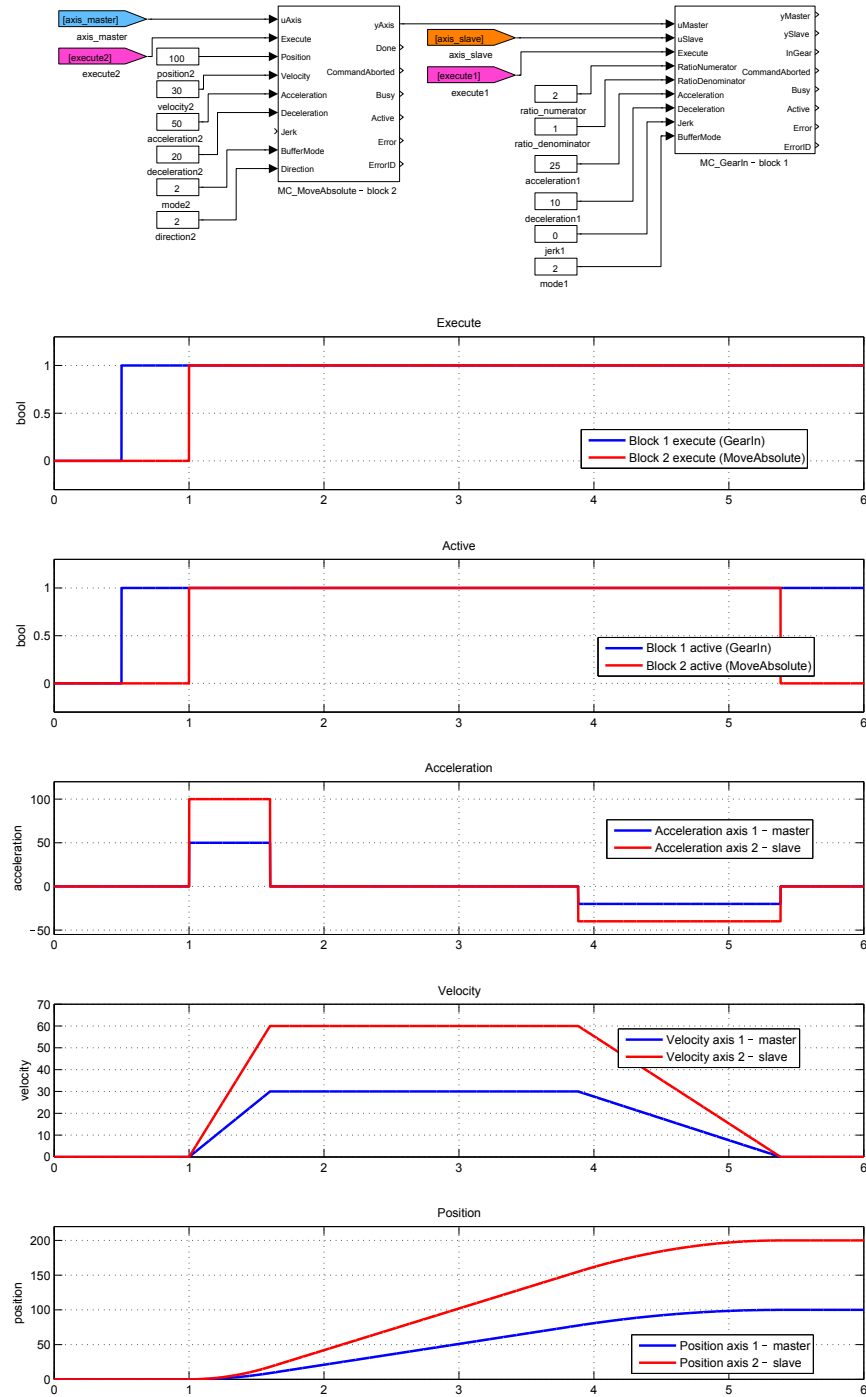
uMaster	Master axis reference	Reference
uSlave	Slave axis reference	Reference
Execute	The block is activated on rising edge	Bool
RatioNumerator	Gear ratio Numerator	Long (I32)
RatioDenominator	Gear ratio Denominator	Long (I32)
Acceleration	Maximal allowed acceleration [unit/s ²]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)

Jerk	Maximal allowed jerk [unit/s ³]	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

yMaster	Master axis reference	Reference
ySlave	Slave axis reference	Reference
InGear	Slave axis reached gearing ratio	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	

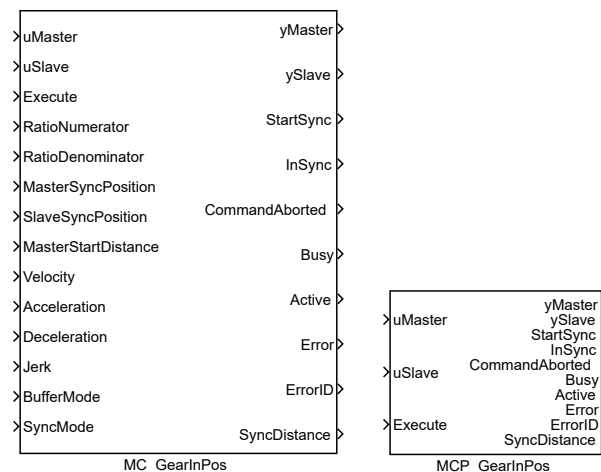
Example



MC_GearInPos, MCP_GearInPos – Engage the master/slave velocity ratio in defined position

Block Symbols

Licence: [MOTION CONTROL](#)



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:

$$\frac{SlavePosition - SlaveSyncPosition}{MasterPosition - MasterSyncPosition} = \frac{RatioNumerator}{RatioDenominator}.$$

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 `MasterStartDistance=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 `MasterSyncPosition ± MasterStartDistance`.

Notes:

1. The synchronization procedure uses two algorithms: I. The algorithm implemented in `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 `SlaveSyncPosition` approximately at the moment when the master position enters the range for synchronization given by `MasterSyncPosition` and `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.

Inputs

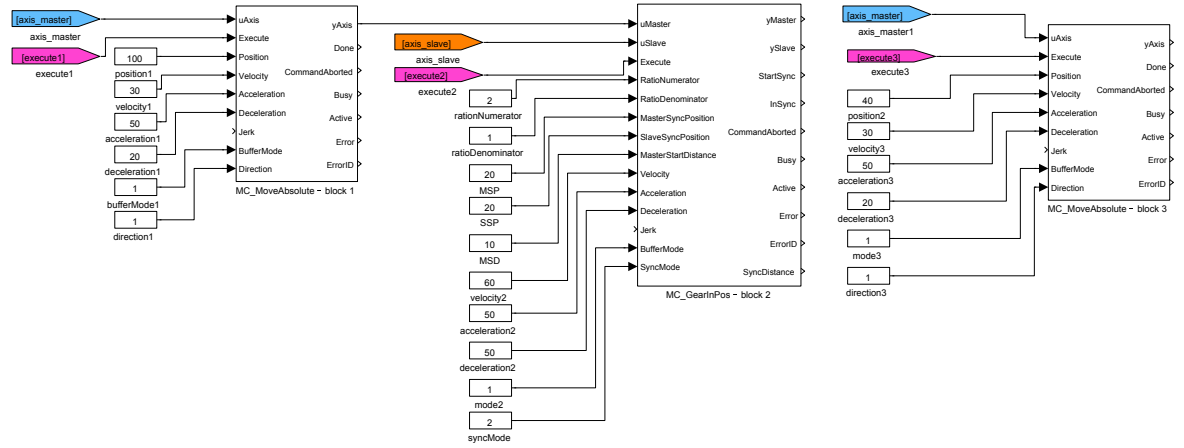
<code>uMaster</code>	Master axis reference	Reference
<code>uSlave</code>	Slave axis reference	Reference
<code>Execute</code>	The block is activated on rising edge	Bool
<code>RatioNumerator</code>	Gear ratio Numerator	Long (I32)
<code>RatioDenominator</code>	Gear ratio Denominator	Long (I32)
<code>MasterSyncPosition</code>	Master position for synchronization	Double (F64)
<code>SlaveSyncPosition</code>	Slave position for synchronization	Double (F64)
<code>MasterStartDistance</code>	Master distance for starting gear in procedure	Double (F64)
<code>Velocity</code>	Maximal allowed velocity [unit/s]	Double (F64)
<code>Acceleration</code>	Maximal allowed acceleration [unit/s ²]	Double (F64)
<code>Deceleration</code>	Maximal allowed deceleration [unit/s ²]	Double (F64)
<code>Jerk</code>	Maximal allowed jerk [unit/s ³]	Double (F64)

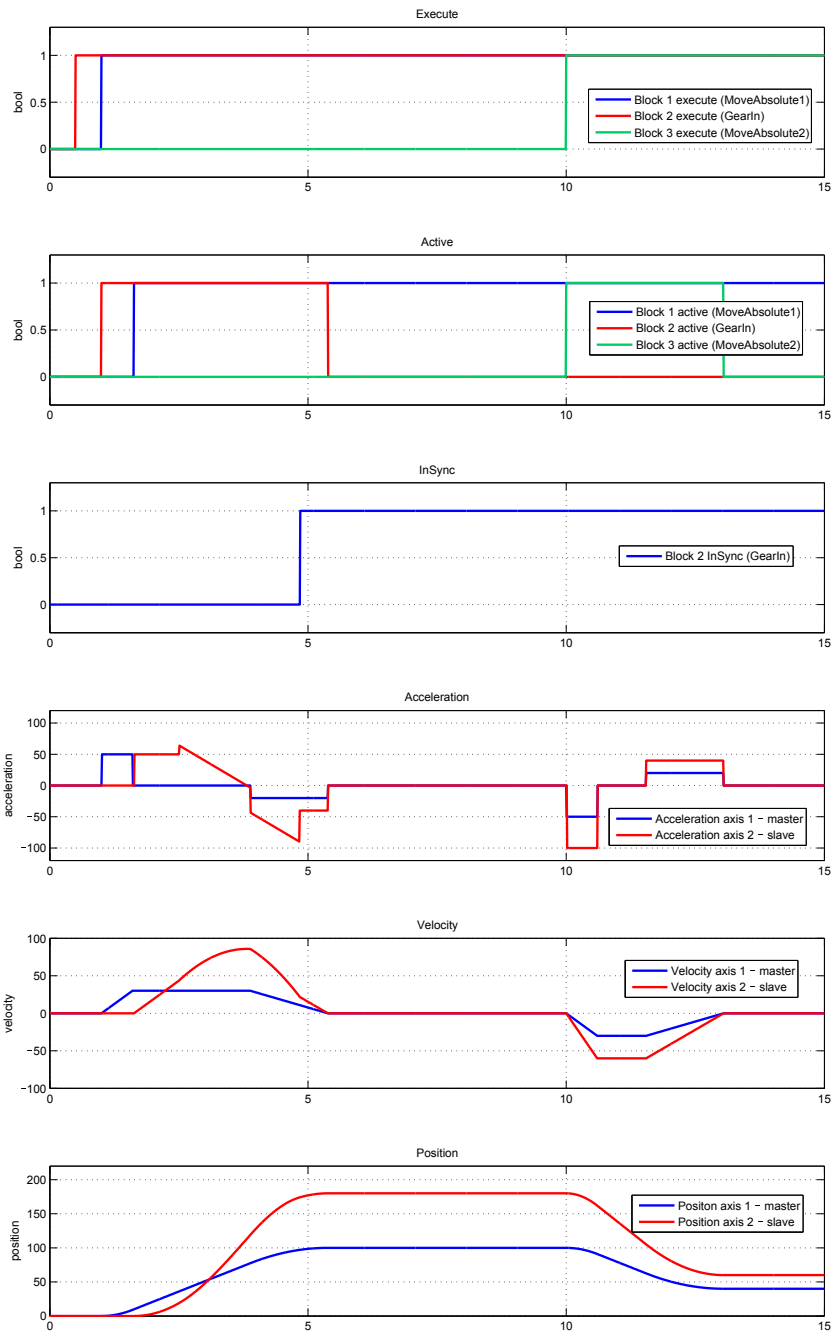
BufferMode	Buffering mode	Long (I32)
1	Aborting (start immediately)	
2	Buffered (start after finish of previous motion)	
3	Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)	
4	Blending high (start after finishing the previous 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)	
SyncMode	Synchronization mode (cyclic axes only)	Long (I32)
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)

Example

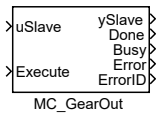




MC_GearOut – Disengage the master/slave velocity ratio

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).

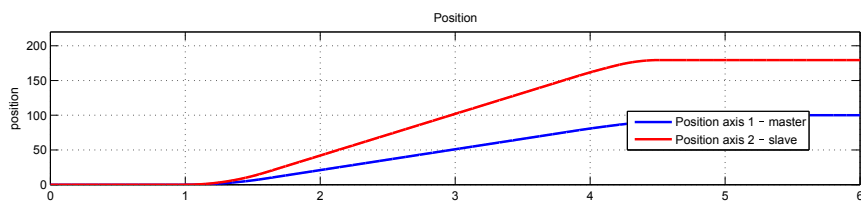
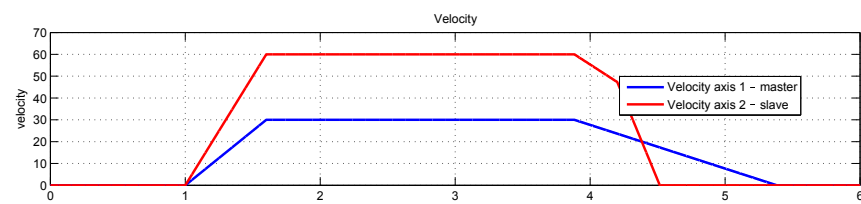
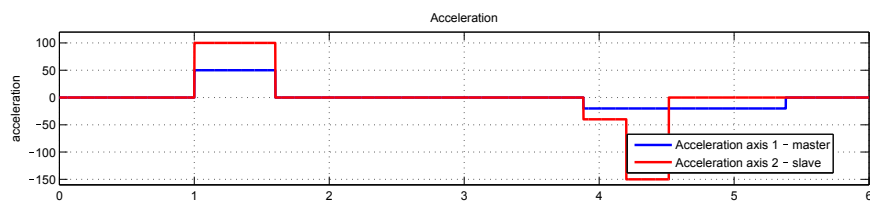
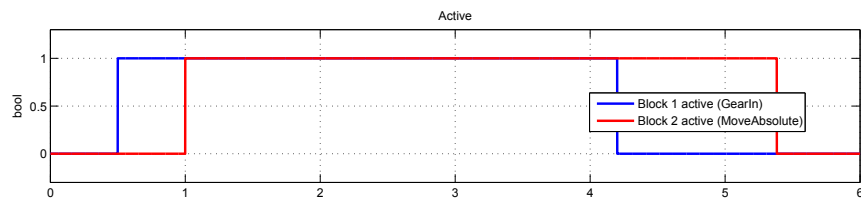
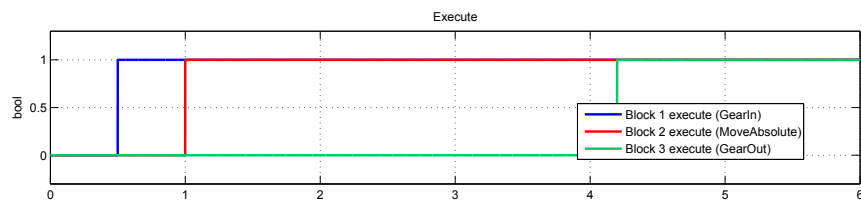
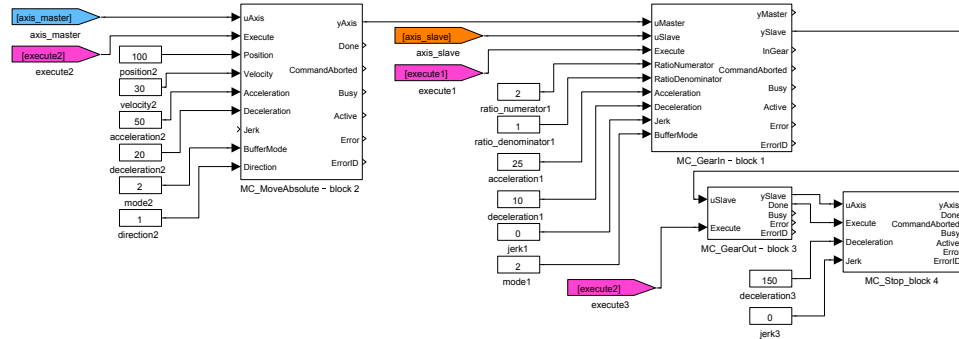
Inputs

<code>uSlave</code>	Slave axis reference	Reference
<code>Execute</code>	The block is activated on rising edge	Bool

Outputs

<code>ySlave</code>	Slave axis reference	Reference
<code>Done</code>	Algorithm finished	Bool
<code>Busy</code>	Algorithm not finished yet	Bool
<code>Error</code>	Error occurred	Bool
<code>ErrorID</code>	Result of the last operation	Error
	i REXYGEN general error	

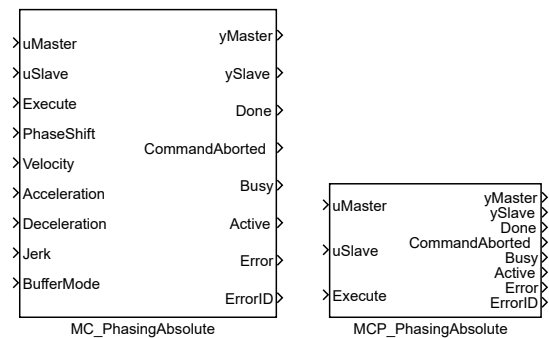
Example



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

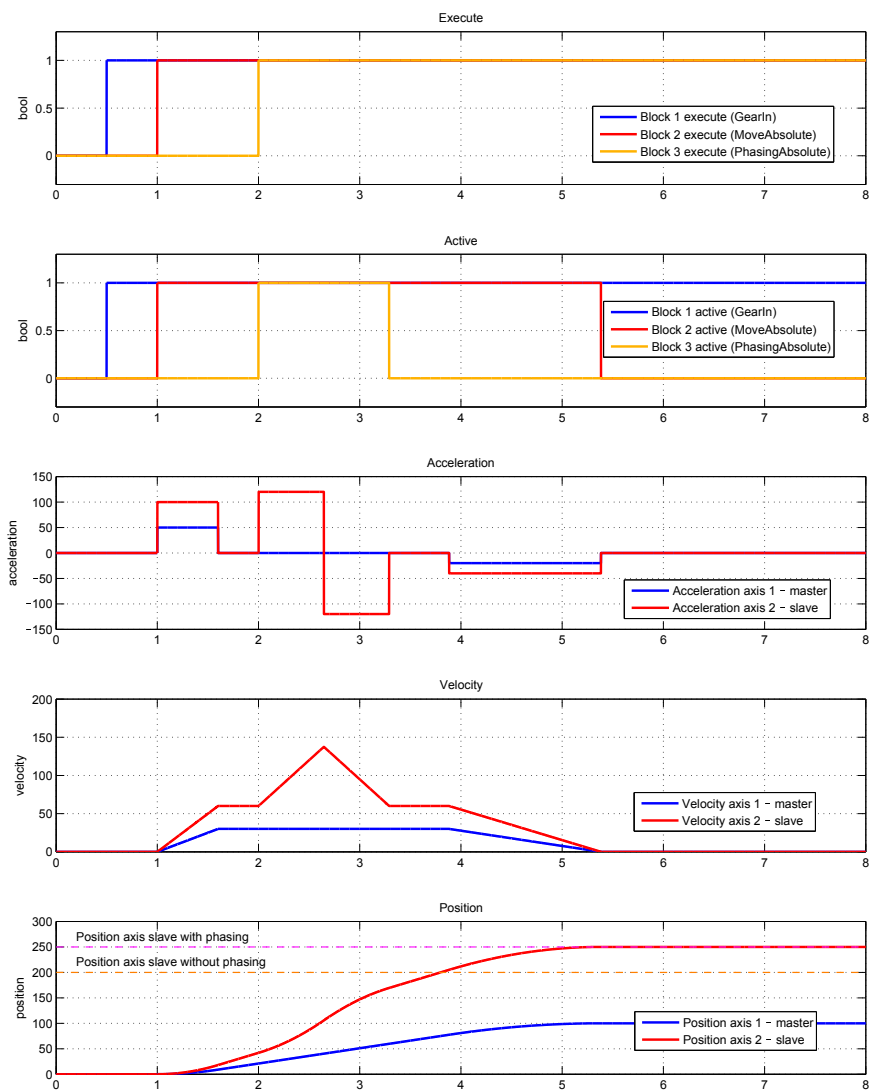
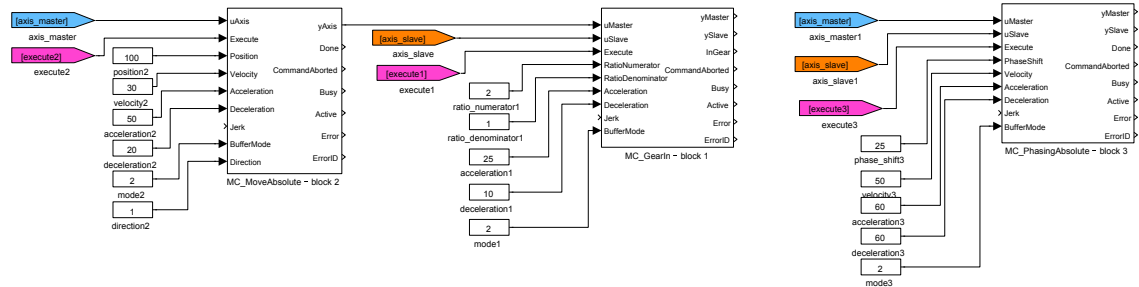
uMaster	Master axis reference	Reference
uSlave	Slave axis reference	Reference
Execute	The block is activated on rising edge	Bool
PhaseShift	Requested phase shift (distance on master axis) for cam	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s ²]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)

Jerk	Maximal allowed jerk [unit/s ³]	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	

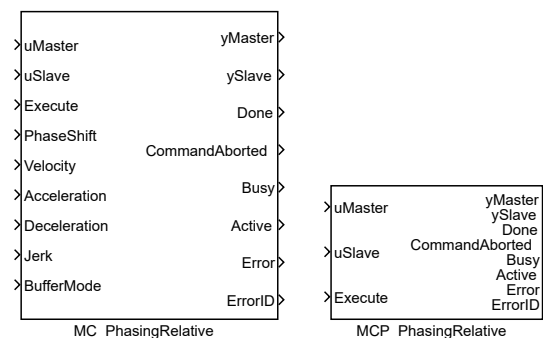
Example



MC_PhasingRelative, MCP_PhasingRelative – Phase shift in synchronized motion (relative coordinates)

Block Symbols

Licence: [MOTION CONTROL](#)



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

uMaster	Master axis reference	Reference
uSlave	Slave axis reference	Reference
Execute	The block is activated on rising edge	Bool
PhaseShift	Requested phase shift (distance on master axis) for cam	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s ²]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)

Jerk	Maximal allowed jerk [unit/s ³]	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	

Chapter 22

MC_COORD – Motion control - coordinated movement blocks

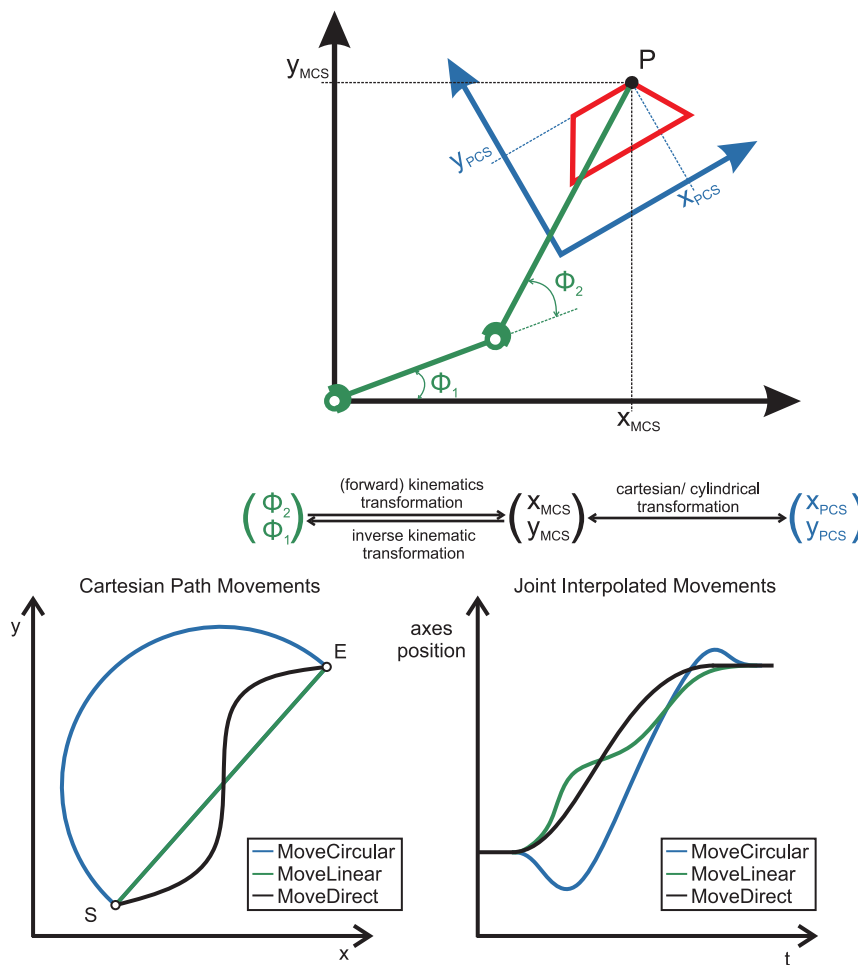
Contents

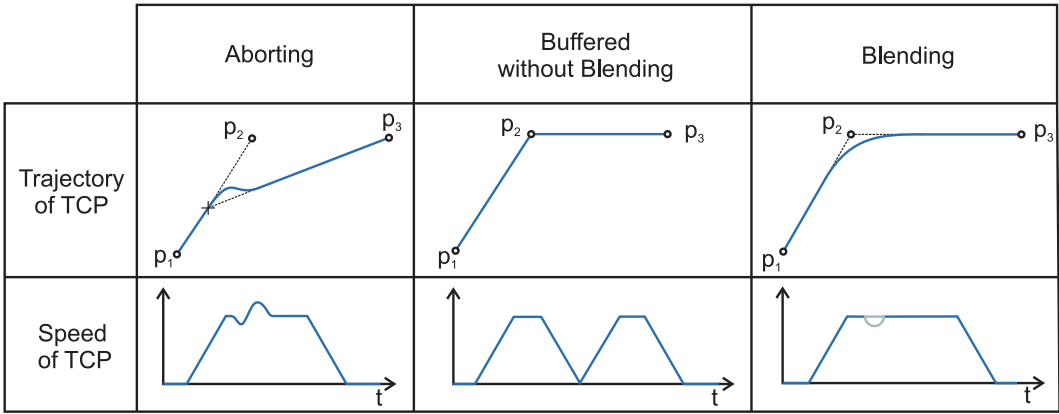
MCP_GroupHalt – * Stopping a group movement (interruptible) .	787
MCP_GroupInterrupt – * Read a group interrupt	788
MCP_GroupSetOverride – * Set group override factors	789
MCP_GroupSetPosition – * Sets the position of all axes in a group	790
MCP_GroupStop – * Stopping a group movement	791
MCP_MoveCircularAbsolute – * Circular move to position (absolute coordinates)	792
MCP_MoveCircularRelative – * Circular move to position (relative to execution point)	794
MCP_MoveDirectAbsolute – * Direct move to position (absolute coordinates)	796
MCP_MoveDirectRelative – * Direct move to position (relative to execution point)	798
MCP_MoveLinearAbsolute – * Linear move to position (absolute coordinates)	800
MCP_MoveLinearRelative – * Linear move to position (relative to execution point)	802
MCP_MovePath – * General spatial trajectory generation	804
MCP_MovePath_PH – * General spatial trajectory generation PH . .	806
MCP_SetCartesianTransform – * Sets Cartesian transformation . .	808
MCP_SetKinTransform_Arm – * Kinematic transformation robot ARM	810
MCP_SetKinTransform_UR – * Kinematic transformation for UR robot	812
MC_AddAxisToGroup – Adds one axis to a group	814
MC_GroupContinue – Continuation of interrupted movement . . .	815
MC_GroupDisable – Changes the state of a group to GroupDisabled	816
MC_GroupEnable – Changes the state of a group to GroupEnable	817

MC_GroupHalt – Stopping a group movement (interruptible) . . .	818
MC_GroupInterrupt, MCP_GroupInterrupt – Read a group interrupt	823
MC_GroupReadActualAcceleration – Read actual acceleration in the selected coordinate system	824
MC_GroupReadActualPosition – Read actual position in the selected coordinate system	825
MC_GroupReadActualVelocity – Read actual velocity in the selected coordinate system	826
MC_GroupReadError – Read a group error	827
MC_GroupReadStatus – Read a group status	828
MC_GroupReset – Reset axes errors	829
MC_GroupSetOverride – Set group override factors	830
MC_GroupSetPosition, MCP_GroupSetPosition – Sets the position of all axes in a group	831
MC_GroupStop – Stopping a group movement	833
MC_MoveCircularAbsolute – Circular move to position (absolute coordinates)	836
MC_MoveCircularRelative – Circular move to position (relative to execution point)	840
MC_MoveDirectAbsolute – Direct move to position (absolute coordinates)	844
MC_MoveDirectRelative – Direct move to position (relative to execution point)	847
MC_MoveLinearAbsolute – Linear move to position (absolute coordinates)	850
MC_MoveLinearRelative – Linear move to position (relative to execution point)	854
MC_MovePath – General spatial trajectory generation	858
MC_MovePath_PH – * General spatial trajectory generation PH . .	860
MC_ReadCartesianTransform – Reads the parameter of the cartesian transformation	862
MC_SetCartesianTransform – Sets Cartesian transformation	863
MC_UngroupAllAxes – Removes all axes from the group	865
RM_AxesGroup – Axes group for coordinated motion control . . .	866
RM_Feed – * MC Feeder ???	869
RM_Gcode – * CNC motion control	870
RM_GroupTrack – T	872

The MC_COORD library is specifically designed for the coordination of multi-axis motion control within complex systems. It encompasses a variety of blocks, including `MC_MoveLinearAbsolute` for executing precise linear movements, complemented by `MC_MoveLinearRelative` for relative linear motion. For the execution of circular motion,

the library incorporates `MC_MoveCircularAbsolute` alongside `MC_MoveCircularRelative`, ensuring detailed circular trajectories. In the context of managing group axis control, this library introduces `MC_AddAxisToGroup`, which is further supported by functionalities such as `MC_GroupEnable` for activation, `MC_GroupDisable` for deactivation, and `MC_GroupHalt` for immediate stopping of grouped axes. Furthermore, the library provides `MC_MoveDirectAbsolute` and `MC_MoveDirectRelative`, enabling direct control over axis movements. For navigating through complex paths, `MC_MovePath` is made available. Essential monitoring and control features are facilitated by `MC_GroupReadActualPosition` for positional data, `MC_GroupReadActualVelocity` for velocity insights, `MC_GroupReadError` for error detection, and `MC_GroupReadStatus` for status updates. Additionally, the library integrates `MC_ReadCartesianTransform` and `MC_SetCartesianTransform`, which are vital for Cartesian transformation processes. This collection of functionalities underscores the library's significance in applications that demand the synchronized control of multiple axes, particularly in the realms of robotics and automation systems.

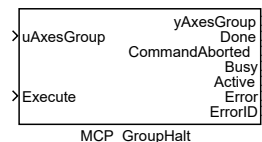




MCP_GroupHalt — * Stopping a group movement (interruptible)

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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

Parameter

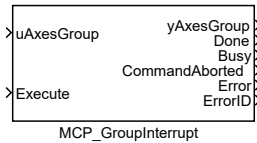
Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	Double (F64)
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	
LimitMode	Velocity/Acceleration/Jerk limits meaning	⊙1 Long (I32)
	1 Relative [part of default]	
	2 Absolute[unit/s unit/s ² ...]	
Superimposed	start as superimposed motion flag	Bool

Output

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 last operation	Error
	i REXYGEN error code	

MCP_GroupInterrupt – * **Read a group interrupt**

Block SymbolLicence: [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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

Parameter

Deceleration	Maximal allowed deceleration [unit/s^2]	Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	Double (F64)
LimitMode	Velocity/Acceleration/Jerk limits meaning	⊙1 Long (I32)
	1 Relative [part of default]	
	2 Absolute[unit/s unit/s^2 ...]	

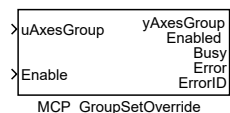
Output

yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
CommandAborted	Algorithm was aborted	Bool
Error	Error occurred	Bool
ErrorID	Result of last operation	Error
	i REXYGEN error code	

MCP_GroupSetOverride — * Set group override factors

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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxesGroup	Axes group reference	Reference
Enable	Block function is enabled	Bool

Parameter

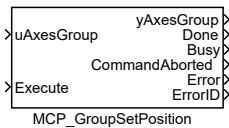
diff	Deadband (difference for recalculation)	⊙0.05	Double (F64)
VelFactor	Velocity multiplication factor	⊙1.0	Double (F64)
AccFactor	Acceleration/deceleration multiplication factor	⊙1.0	Double (F64)
JerkFactor	Jerk multiplication factor	⊙1.0	Double (F64)

Output

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 last operation	Error
i REXYGEN error code		

MCP_GroupSetPosition – * Sets the position of all axes in a group

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.

This block does not propagates the signal quality. More information can be found in the 1.4 section.

Input

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

Parameter

Relative	Mode of position inputs	Bool
CoordSystem	Reference to the coordinate system used	⊙3 Long (I32)
	1 ACS	
	2 MCS	
	3 PCS	
	4 TCS	
Position_	Array of coordinates (positions and orientations)	Double (F64)
	⊙[0.0 0.0 0.0 0.0 0.0 0.0]	

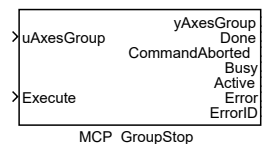
Output

yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
CommandAborted	Algorithm was aborted	Bool
Error	Error occurred	Bool
ErrorID	Result of last operation	Error
	i REXYGEN error code	

MCP_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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

Parameter

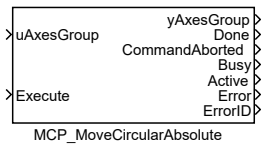
Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	Double (F64)

Output

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 last operation	Error
i REXYGEN error code		

MCP_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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

<code>uAxesGroup</code>	Axes group reference	Reference
<code>Execute</code>	The block is activated on rising edge	Bool

Parameter

<code>CircMode</code>	Specifies the meaning of the input signals <code>AuxPoint</code> and <code>CircDirection</code>	⊙1	Long (I32)
	1 BORDER		
	2 CENTER		
	3 RADIUS		
<code>PathChoice</code>	Choice of path	⊙1	Long (I32)
	1 Clockwise		
	2 CounterClockwise		
<code>Velocity</code>	Maximal allowed velocity [unit/s]		Double (F64)
<code>Acceleration</code>	Maximal allowed acceleration [unit/s^2]		Double (F64)
<code>Jerk</code>	Maximal allowed jerk [unit/s^3]		Double (F64)
<code>CoordSystem</code>	Reference to the coordinate system used	⊙1	Long (I32)
	1 ACS		
	2 MCS		
	3 PCS		
	4 TCS		

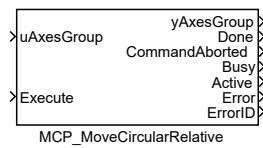
BufferMode	Buffering mode	⊙1	Long (I32)
	1 Aborting		
	2 Buffered		
	3 Blending low		
	4 Blending high		
	5 Blending previous		
	6 Blending next		
LimitMode	Velocity/Acceleration/Jerk limits meaning	⊙1	Long (I32)
	1 Relative [part of default]		
	2 Absolute[unit/s unit/s^2 ...]		
TransitionMode	Transition mode in blending mode	⊙1	Long (I32)
	1 TMNone		
	2 TMStartVelocity		
	3 TMConstantVelocity		
	4 TMCornerDistance		
	5 TMMaxCornerDeviation		
	11 Smooth		
TransitionParameter	Parametr for transition (depends on transition mode)		Double (F64)
Superimposed	start as superimposed motion flag		Bool
AuxPoint	Next coordinates to define circle (depend on CircMode)		Double (F64)
	⊙[0.0 0.0 0.0 0.0 0.0 0.0]		
EndPoint	Target axes coordinates position		Double (F64)
	⊙[0.0 0.0 0.0 0.0 0.0 0.0]		

Output

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 last operation	Error
	i REXYGEN error code	

MCP_MoveCircularRelative – * **Circular 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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

<code>uAxesGroup</code>	Axes group reference	Reference
<code>Execute</code>	The block is activated on rising edge	Bool

Parameter

<code>CircMode</code>	Specifies the meaning of the input signals <code>AuxPoint</code> and <code>CircDirection</code>	⊙1	Long (I32)
	1 BORDER		
	2 CENTER		
	3 RADIUS		
<code>PathChoice</code>	Choice of path	⊙1	Long (I32)
	1 Clockwise		
	2 CounterClockwise		
<code>Velocity</code>	Maximal allowed velocity [unit/s]		Double (F64)
<code>Acceleration</code>	Maximal allowed acceleration [unit/s^2]		Double (F64)
<code>Jerk</code>	Maximal allowed jerk [unit/s^3]		Double (F64)
<code>CoordSystem</code>	Reference to the coordinate system used	⊙1	Long (I32)
	1 ACS		
	2 MCS		
	3 PCS		
	4 TCS		

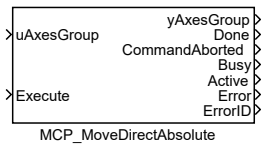
BufferMode	Buffering mode	⊙1	Long (I32)
	1 Aborting		
	2 Buffered		
	3 Blending low		
	4 Blending high		
	5 Blending previous		
	6 Blending next		
LimitMode	Velocity/Acceleration/Jerk limits meaning	⊙1	Long (I32)
	1 Relative [part of default]		
	2 Absolute[unit/s unit/s^2 ...]		
TransitionMode	Transition mode in blending mode	⊙1	Long (I32)
	1 TMNone		
	2 TMStartVelocity		
	3 TMConstantVelocity		
	4 TMCornerDistance		
	5 TMMaxCornerDeviation		
	11 Smooth		
TransitionParameter	Parametr for transition (depends on transition mode)		Double (F64)
Superimposed	start as superimposed motion flag		Bool
AuxPoint	Next coordinates to define circle (depend on CircMode)		Double (F64)
	⊙[0.0 0.0 0.0 0.0 0.0 0.0]		
EndPoint	Target axes coordinates position		Double (F64)
	⊙[0.0 0.0 0.0 0.0 0.0 0.0]		

Output

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 last operation	Error
	i REXYGEN error code	

MCP_MoveDirectAbsolute – * **Direct move to position (absolute coordinates)**

Block SymbolLicence: [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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

Parameter

CoordSystem	Reference to the coordinate system used	⊙1	Long (I32)
	1 ACS		
	2 MCS		
	3 PCS		
	4 TCS		
BufferMode	Buffering mode	⊙1	Long (I32)
	1 Aborting		
	2 Buffered		
	3 Blending low		
	4 Blending high		
	5 Blending previous		
	6 Blending next		
TransitionMode	Transition mode in blending mode	⊙1	Long (I32)
	1 TMNone		
	2 TMStartVelocity		
	3 TMConstantVelocity		
	4 TMCornerDistance		
	5 TMMaxCornerDeviation		
	11 Smooth		

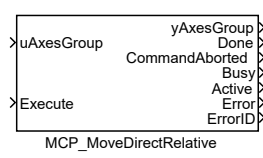
TransitionParameter	Parametr for transition (depends on transition mode)	Double (F64)
Superimposed	start as superimposed motion flag	Bool
Position_	Array of coordinates (positions and orientations)	Double (F64)
	⊙[0.0 0.0 0.0 0.0 0.0 0.0]	

Output

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 last operation	Error
	i REXYGEN error code	

MCP_MoveDirectRelative – * **Direct move to position (relative to execution point)**

Block SymbolLicence: [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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

Parameter

CoordSystem	Reference to the coordinate system used	⊙1	Long (I32)
	1 ACS		
	2 MCS		
	3 PCS		
	4 TCS		
BufferMode	Buffering mode	⊙1	Long (I32)
	1 Aborting		
	2 Buffered		
	3 Blending low		
	4 Blending high		
	5 Blending previous		
	6 Blending next		
TransitionMode	Transition mode in blending mode	⊙1	Long (I32)
	1 TMNone		
	2 TMStartVelocity		
	3 TMConstantVelocity		
	4 TMCornerDistance		
	5 TMMaxCornerDeviation		
	11 Smooth		

TransitionParameter	Parametr for transition (depends on transition mode)	Double (F64)
Superimposed	start as superimposed motion flag	Bool
Distance	Array of coordinates (relative distances and orientations) ⊙[0.0 0.0 0.0 0.0 0.0 0.0]	Double (F64)

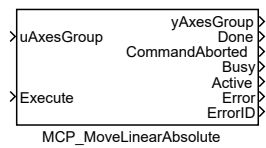
Output

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 last operation i REXYGEN error code	Error

MCP_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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

Parameter

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 Long (I32)
	1 ACS	
	2 MCS	
	3 PCS	
	4 TCS	
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	
	3 Blending low	
	4 Blending high	
	5 Blending previous	
	6 Blending next	
LimitMode	Velocity/Acceleration/Jerk limits meaning	⊙1 Long (I32)
	1 Relative [part of default]	
	2 Absolute[unit/s unit/s^2 ...]	

TransitionMode	Transition mode in blending mode	⊙1	Long (I32)
1 TMNone		
2 TMStartVelocity		
3 TMConstantVelocity		
4 TMCornerDistance		
5 TMMaxCornerDeviation		
11 Smooth		
TransitionParameter	Parametr for transition (depends on transition mode)		Double (F64)
Superimposed	start as superimposed motion flag		Bool
Position_	Array of coordinates (positions and orientations)		Double (F64)
	⊙[0.0 0.0 0.0 0.0 0.0 0.0]		

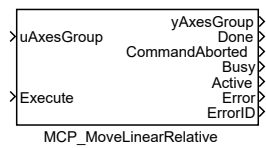
Output

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 last operation	Error
i REXYGEN error code	

MCP_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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

Parameter

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 Long (I32)
	1 ACS	
	2 MCS	
	3 PCS	
	4 TCS	
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	
	3 Blending low	
	4 Blending high	
	5 Blending previous	
	6 Blending next	
LimitMode	Velocity/Acceleration/Jerk limits meaning	⊙1 Long (I32)
	1 Relative [part of default]	
	2 Absolute[unit/s unit/s^2 ...]	

TransitionMode	Transition mode in blending mode	⊙1	Long (I32)
1 TMNone		
2 TMStartVelocity		
3 TMConstantVelocity		
4 TMCornerDistance		
5 TMMaxCornerDeviation		
11 Smooth		
TransitionParameter	Parametr for transition (depends on transition mode)		Double (F64)
Superimposed	start as superimposed motion flag		Bool
Distance	Array of coordinates (relative distances and orientations)		Double (F64)
	⊙[0.0 0.0 0.0 0.0 0.0 0.0]		

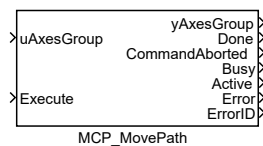
Output

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 last operation	Error
i REXYGEN error code	

MCP_MovePath – * **General spatial trajectory generation**

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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

Parameter

TotalTime	Time [s] for whole move	Double (F64)
RampTime	Time [s] for acceleration/deceleration	Double (F64)
CoordSystem	Reference to the coordinate system used	⊙2 Long (I32)
	1 ACS	
	2 MCS	
	3 PCS	
	4 TCS	
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	
	3 Blending low	
	4 Blending high	
	5 Blending previous	
	6 Blending next	
TransitionMode	Transition mode in blending mode	⊙1 Long (I32)
	1 TMNone	
	2 TMStartVelocity	
	3 TMConstantVelocity	
	4 TMCornerDistance	
	5 TMMaxCornerDeviation	
	11 Smooth	

TransitionParameter	Parametr for transition (depends on transition mode)	Double (F64)
RampIn	RampIn factor (0 = RampIn mode not used)	Double (F64)
Superimposed	start as superimposed motion flag	Bool
pc	Control-points matrix $\odot[0.0 \ 1.0 \ 2.0; \ 0.0 \ 1.0 \ 1.0; \ 0.0 \ 1.0 \ 0.0]$	Double (F64)
pk	Knot-points vector $\odot[0.0 \ 0.0 \ 0.0 \ 0.0 \ 0.5 \ 1.0 \ 1.0]$	Double (F64)
pw	Weighting vector $\odot[1.0 \ 1.0 \ 1.0]$	Double (F64)
pv	Polynoms for feedrate definition $\odot[0.0 \ 0.05 \ 0.95; \ 0.0 \ 0.1 \ 0.1; \ 0.0 \ 0.0 \ 0.0; \ 0.1 \ 0.0 \ -0.1; \ -0.05 \ 0.0 \ 0.05; \ 0.0 \ 0.0 \ 0.0]$	Double (F64)
pt	Knot-points (time [s]) for feedrate $\odot[0.0 \ 1.0 \ 10.0 \ 11.0]$	Double (F64)
user	Only for special edit $\odot[0.0 \ 1.0 \ 2.0 \ 3.0]$	Double (F64)

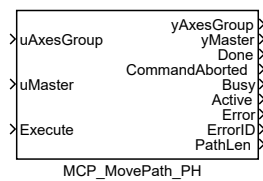
Output

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 last operation i REXYGEN error code	Error

MCP_MovePath_PH – * **General spatial trajectory generation PH**

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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxesGroup	Axes group reference	Reference
uMaster	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

Parameter

CoordSystem	Reference to the coordinate system used	⊙2	Long (I32)
	1 ACS		
	2 MCS		
	3 PCS		
	4 TCS		
BufferMode	Buffering mode	⊙1	Long (I32)
	1 Aborting		
	2 Buffered		
	3 Blending low		
	4 Blending high		
	5 Blending previous		
	6 Blending next		
Cyclic	Profile is cyclic flag		Bool
RampIn	RampIn factor (0 = RampIn mode not used)		Double (F64)
Superimposed	start as superimposed motion flag		Bool
UZV1		⊙[]	Double (F64)
Points		⊙[]	Double (F64)

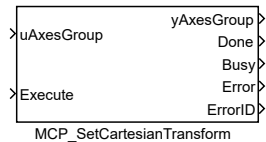
Output

yAxesGroup	Axes group reference	Reference
yMaster	Axis reference (only RM_Axis.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 last operation	Error
PathLen		Double (F64)

MCP_SetCartesianTransform – * Sets Cartesian transformation

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.

This block does not propagates the signal quality. More information can be found in the 1.4 section.

Input

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

Parameter

SelTrans	Coordinate transformation to set/get	⊙1	Long (I32)
	1 PCS offset		
	2 Tool offset		
	3 Machine base offset		
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)
Relative	Mode of position inputs		Bool
SSF	Simple shift flag		Bool

Output

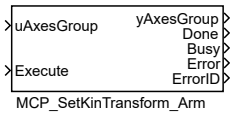
yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool

ErrorID	Result of last operation	Error
i REXYGEN error code	

MCP_SetKinTransform_Arm – * **Kinematic transformation robot ARM**

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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

Parameter

arot	Angle for one revolute (ACS)	⊙6.28318531	Double (F64)
mrot	Angle for one revolute (MCS)	⊙6.28318531	Double (F64)
irt	Rotation format	⊙1	Long (I32)
	1 ZYX angles		
	2 Quaternion		
a1		⊙400.0	Double (F64)
a2		⊙300.0	Double (F64)
a3			Double (F64)
d1			Double (F64)
d23			Double (F64)
d4		⊙200.0	Double (F64)
d6		⊙100.0	Double (F64)
xe			Double (F64)
ye			Double (F64)
ze			Double (F64)
gamaE			Double (F64)
betaE			Double (F64)
alphaE			Double (F64)

Output

		Reference
yAxesGroup	Axes group reference	
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of last operation	Error
	i REXYGEN error code	

MCP_SetKinTransform_UR – * **Kinematic transformation for UR robot**

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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

Parameter

arot	Angle for one revolute (ACS)	⊙6.28318531	Double (F64)
mrot	Angle for one revolute (MCS)	⊙6.28318531	Double (F64)
irt	Rotation format	⊙1	Long (I32)
	1 ZYX angles		
	2 Quaternion		
11			Double (F64)
12			Double (F64)
13			Double (F64)
14		⊙400.0	Double (F64)
15		⊙400.0	Double (F64)
16		⊙400.0	Double (F64)
17		⊙400.0	Double (F64)
18		⊙400.0	Double (F64)

Output

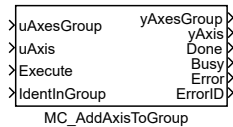
yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool

Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of last operation	Error
	i REXYGEN error code	

MC_AddAxisToGroup – Adds one axis to a group

Block Symbol

Licence: COORDINATED MOTION



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	Reference
uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
IdentInGroup	The order of axes in the group (0 = first unassigned)	Long (I32)

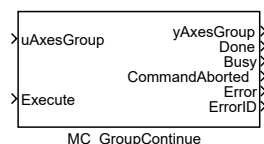
Outputs

yAxesGroup	Axes group reference	Reference
yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
	i REXYGEN general error	

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	Reference
Execute	The block is activated on rising edge	Bool

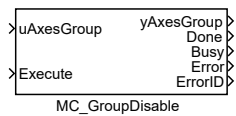
Outputs

yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
CommandAborted	Algorithm was aborted	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
	i REXYGEN general error	

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	Reference
Execute	The block is activated on rising edge	Bool

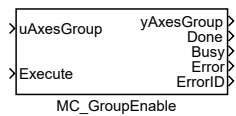
Outputs

yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
	i REXYGEN general error	

MC_GroupEnable – Changes the state of a group to GroupEnable

Block Symbol

Licence: [COORDINATED MOTION](#)



Function Description

The function block **MC_GroupEnable** changes the state for the group **uAxesGroup** from "GroupDisabled" to "GroupStandby".

Inputs

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

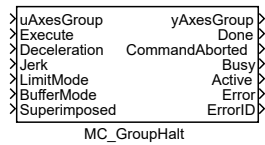
Outputs

yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
	i REXYGEN general error	

MC_GroupHalt – Stopping a group movement (interruptible)

Block Symbol

Licence: [COORDINATED MOTION](#)



Function Description

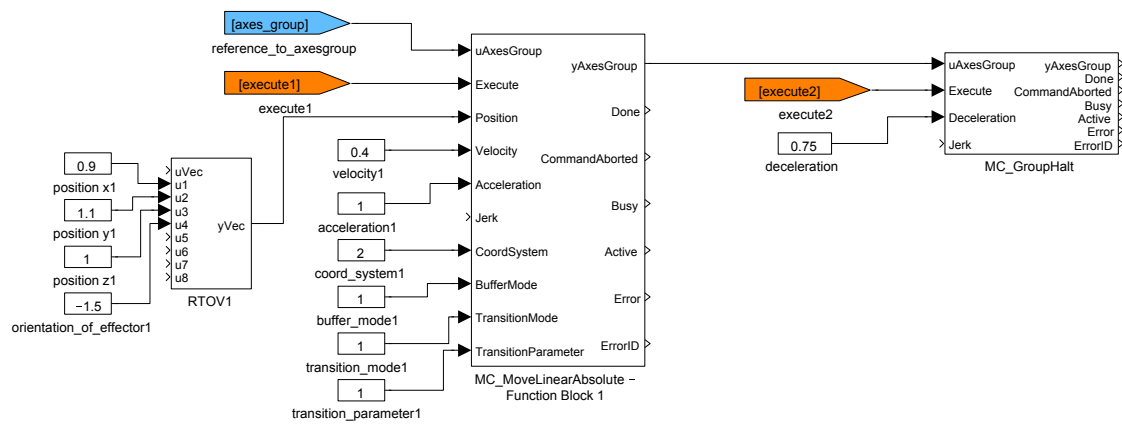
The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

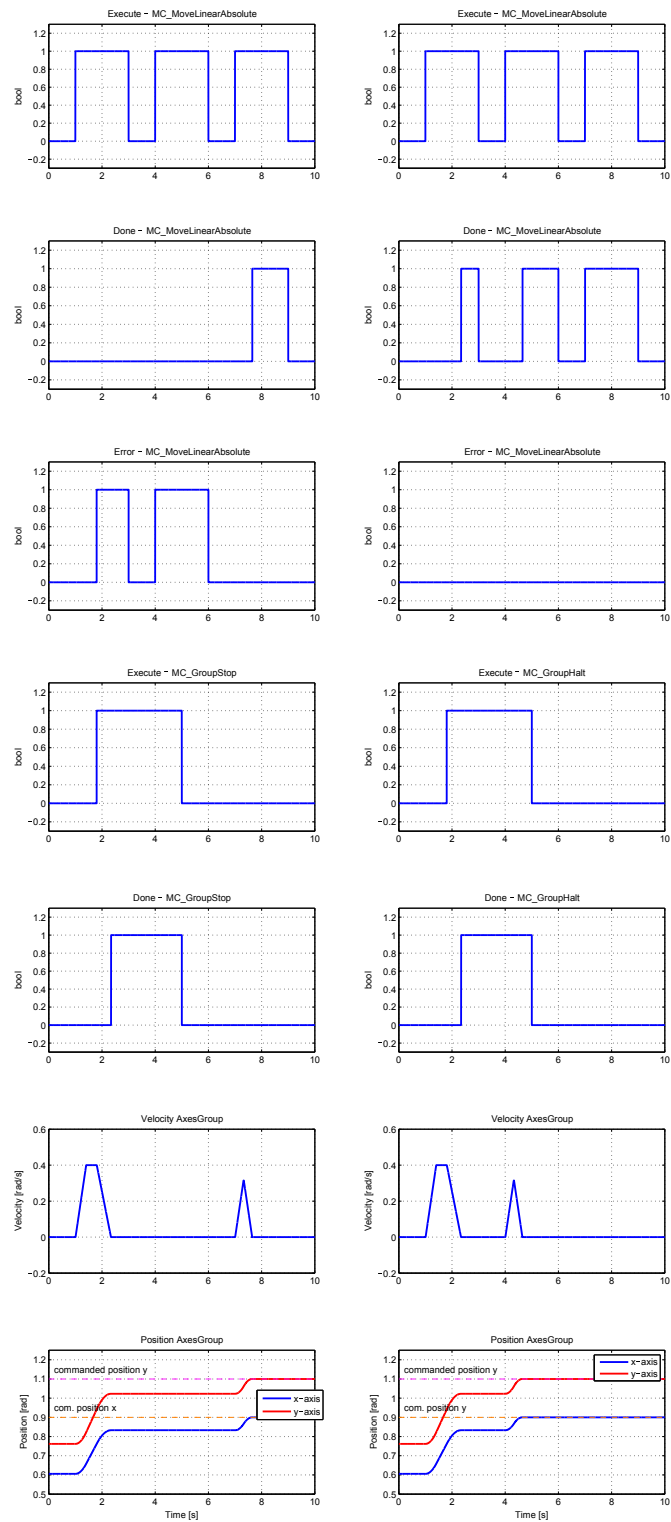
Inputs

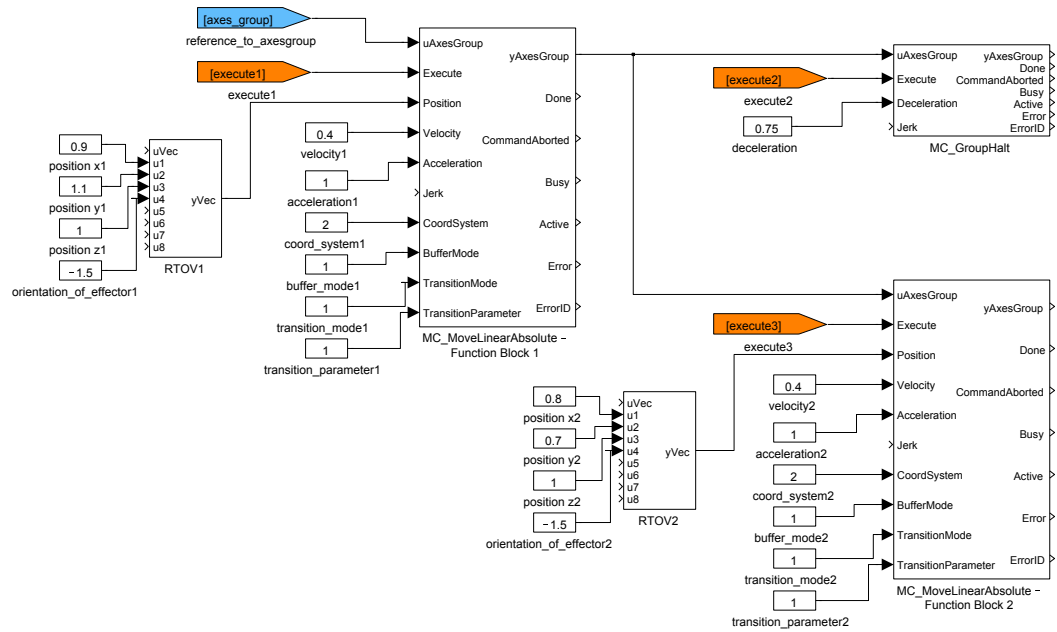
uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool
Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	Double (F64)

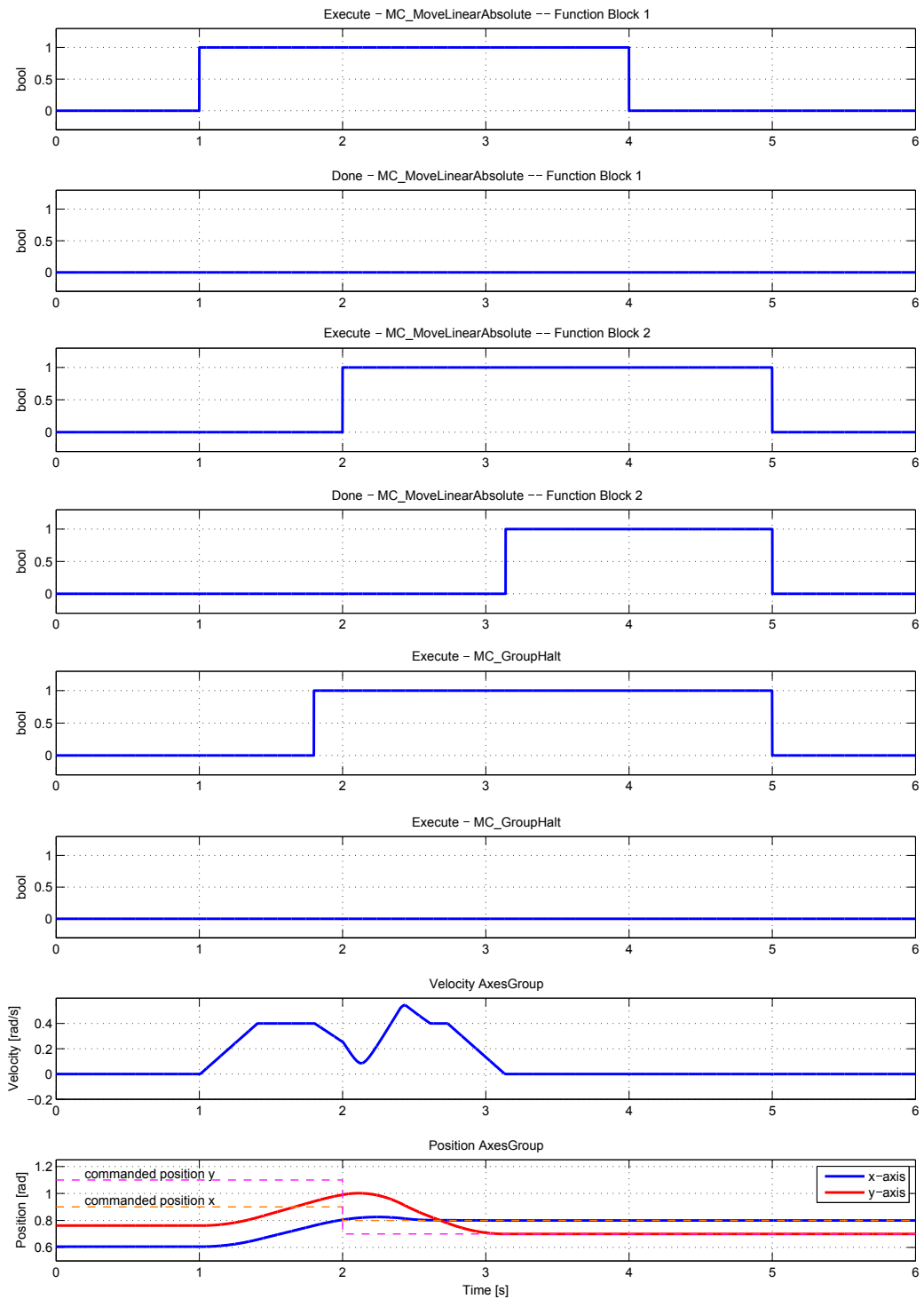
Outputs

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	





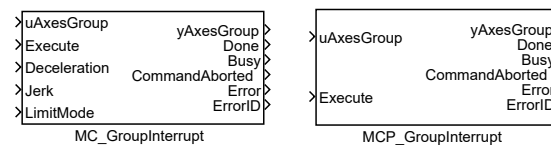




MC_GroupInterrupt, MCP_GroupInterrupt – Read a group interrupt

Block Symbols

Licence: [COORDINATED MOTION](#)



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

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool
Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	Double (F64)

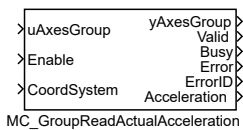
Outputs

yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
CommandAborted	Algorithm was aborted	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
i REXYGEN general error		

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

<code>uAxesGroup</code>	Axes group reference	Reference
<code>Enable</code>	Block function is enabled	Bool
<code>CoordSystem</code>	Reference to the coordinate system used	Long (I32)
	1 ACS	
	2 MCS	
	3 PCS	

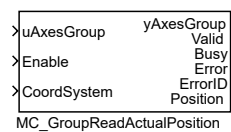
Outputs

<code>yAxesGroup</code>	Axes group reference	Reference
<code>Valid</code>	Output value is valid	Bool
<code>Busy</code>	Algorithm not finished yet	Bool
<code>Error</code>	Error occurred	Bool
<code>ErrorID</code>	Result of the last operation	Error
	i REXYGEN general error	
<code>Acceleration</code>	xxx	Reference

MC_GroupReadActualPosition – Read actual position in the selected coordinate system

Block Symbol

Licence: [COORDINATED MOTION](#)



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	Reference
Enable	Block function is enabled	Bool
CoordSystem	Reference to the coordinate system used	Long (I32)
	1 ACS	
	2 MCS	
	3 PCS	

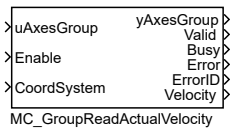
Outputs

yAxesGroup	Axes group reference	Reference
Valid	Output value is valid	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
	i REXYGEN general error	
Position	xxx	Reference

MC_GroupReadActualVelocity – Read actual velocity in the selected coordinate system

Block Symbol

Licence: 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

<code>uAxesGroup</code>	Axes group reference	Reference
<code>Enable</code>	Block function is enabled	Bool
<code>CoordSystem</code>	Reference to the coordinate system used	Long (I32)
	1 ACS	
	2 MCS	
	3 PCS	

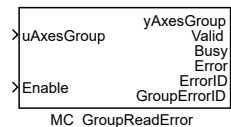
Outputs

<code>yAxesGroup</code>	Axes group reference	Reference
<code>Valid</code>	Output value is valid	Bool
<code>Busy</code>	Algorithm not finished yet	Bool
<code>Error</code>	Error occurred	Bool
<code>ErrorID</code>	Result of the last operation	Error
	i REXYGEN general error	
<code>Velocity</code>	xxx	Reference

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

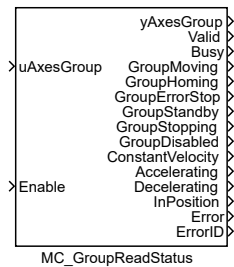
Outputs

yAxesGroup	Axes group reference	Reference
Valid	Output value is valid	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
	i REXYGEN general error	
GroupErrorID	Result of the last operation	Error
	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

uAxesGroup	Axes group reference	Reference
Enable	Block function is enabled	Bool

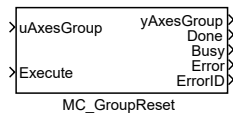
Outputs

yAxesGroup	Axes group reference	Reference
Valid	Output value is valid	Bool
Busy	Algorithm not finished yet	Bool
GroupMoving	State GroupMoving	Bool
GroupHoming	State GroupHoming	Bool
GroupErrorStop	State ErrorStop	Bool
GroupStandby	State Standby	Bool
GroupStopping	State Stopping	Bool
GroupDisabled	State Disabled	Bool
ConstantVelocity	Constant velocity motion	Bool
Accelerating	Accelerating	Bool
Decelerating	Decelerating	Bool
InPosition	Symptom achieve the desired position	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
i REXYGEN general error		

MC_GroupReset – Reset axes errors

Block Symbol

Licence: COORDINATED MOTION



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

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

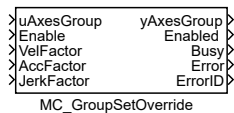
Outputs

yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
	i REXYGEN general error	

MC_GroupSetOverride – Set group override factors

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
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)
------	---	-------	--------------

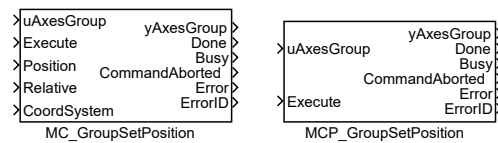
Outputs

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	

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

<code>uAxesGroup</code>	Axes group reference	Reference
<code>Execute</code>	The block is activated on rising edge	Bool
<code>Position</code>	Array of coordinates (positions and orientations)	Reference
<code>Relative</code>	Mode of position inputs	Bool
	off ... absolute	
	on relative	
<code>CoordSystem</code>	Reference to the coordinate system used	Long (I32)
	1 ACS	
	2 MCS	
	3 PCS	

Outputs

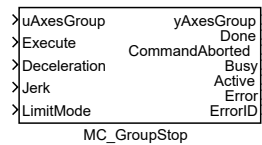
<code>yAxesGroup</code>	Axes group reference	Reference
<code>Done</code>	Algorithm finished	Bool
<code>Busy</code>	Algorithm not finished yet	Bool
<code>CommandAborted</code>	Algorithm was aborted	Bool
<code>Error</code>	Error occurred	Bool

ErrorID	Result of the last operation	Error
i	REXYGEN general error	

MC_GroupStop – Stopping a group movement

Block Symbol

Licence: [COORDINATED MOTION](#)



Function Description

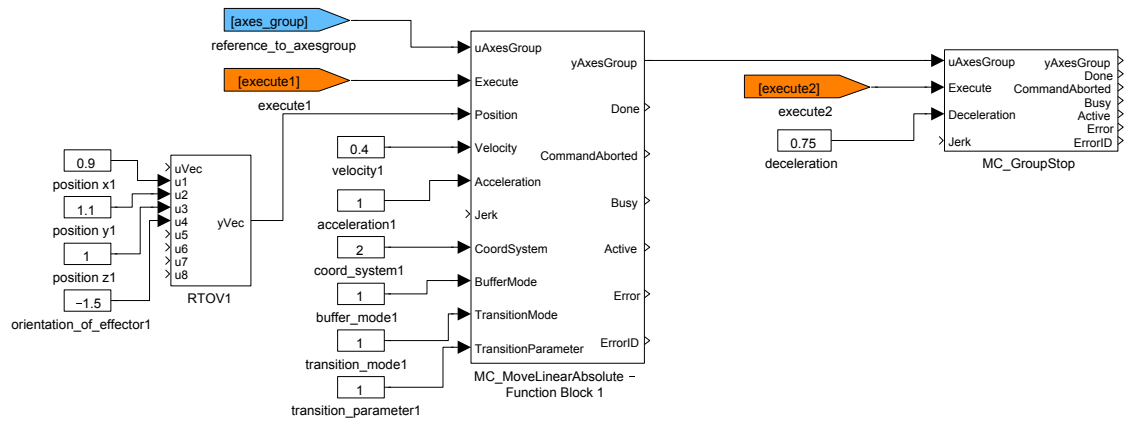
The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

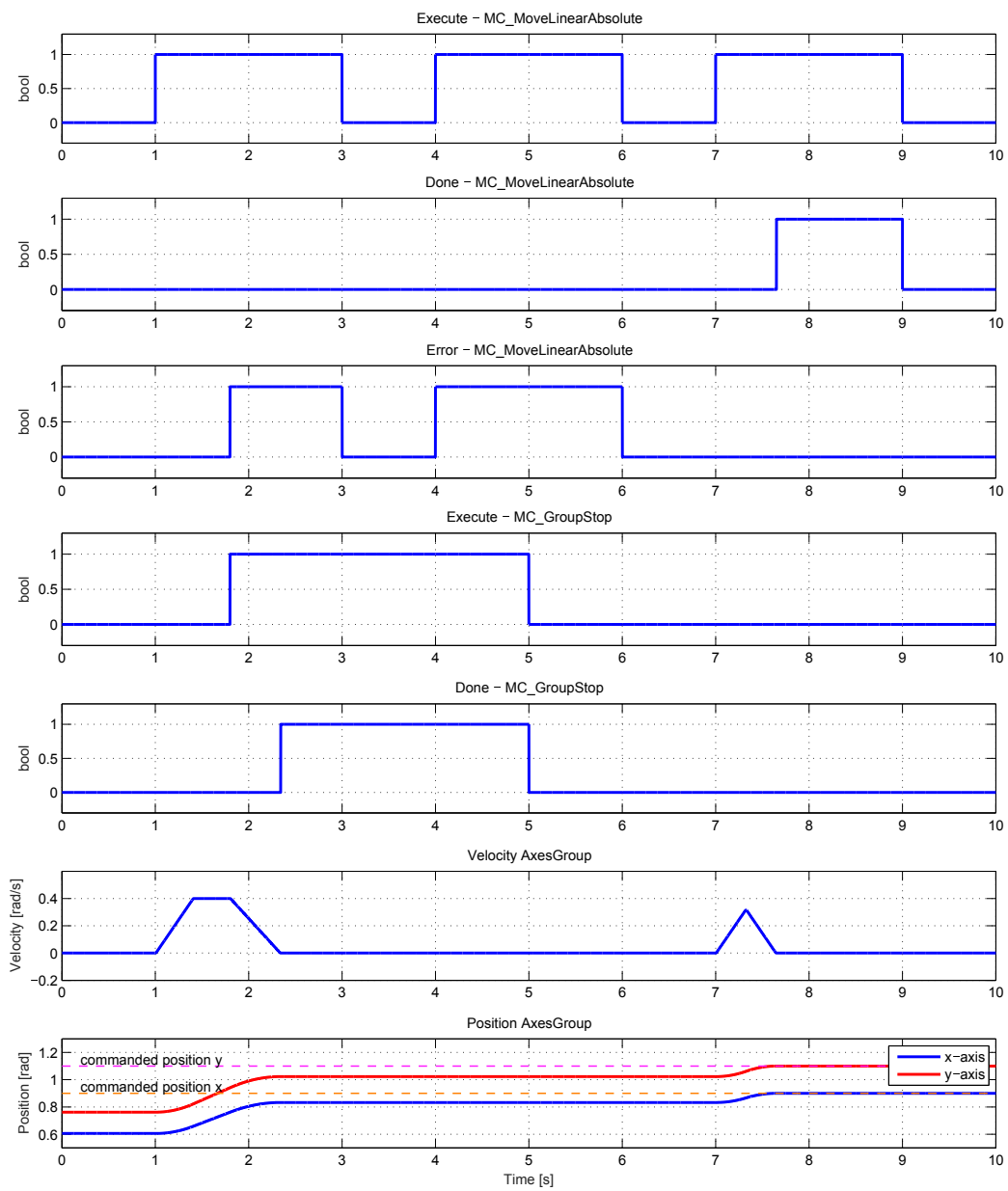
Inputs

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool
Deceleration	Maximal allowed deceleration [unit/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	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	

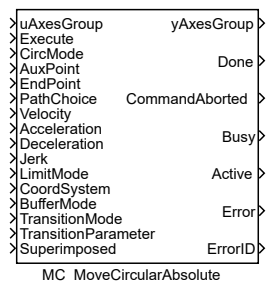




MC_MoveCircularAbsolute – Circular move to position (absolute coordinates)

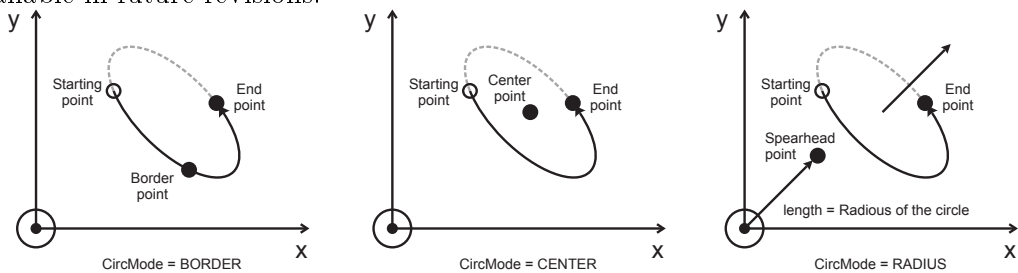
Block Symbol

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.



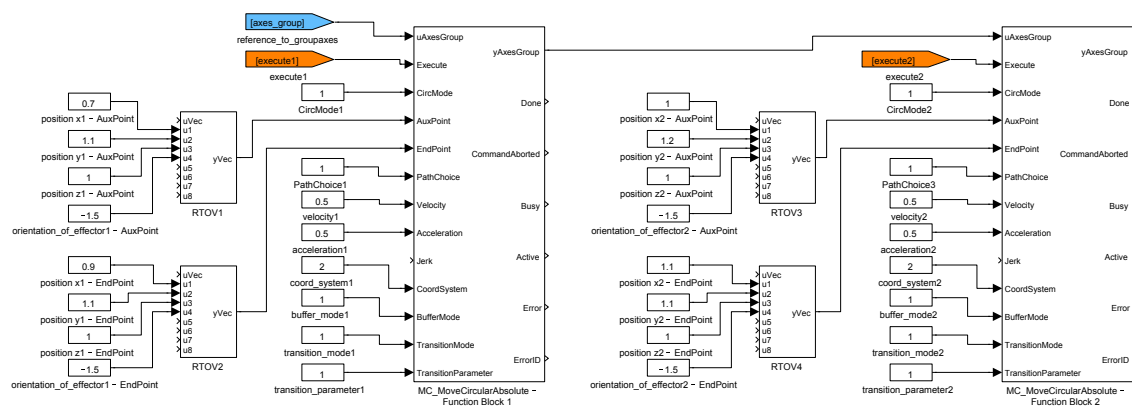
Inputs

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool
CircMode	Specifies the meaning of the input signals AuxPoint and CircDirection	Long (I32)
	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	Long (I32)
	1 Clockwise	
	2 CounterClockwise	
Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s ²]	Double (F64)

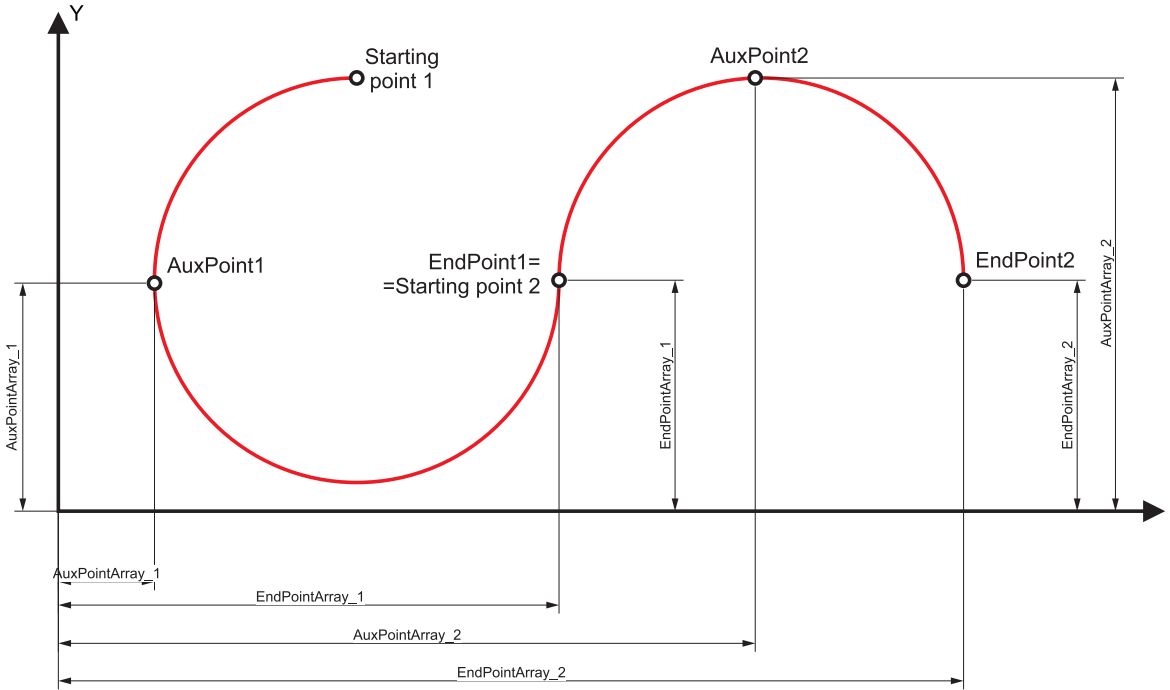
Jerk	Maximal allowed jerk [unit/s ³]	Double (F64)
CoordSystem	Reference to the coordinate system used	Long (I32)
	1 ACS	
	2 MCS	
	3 PCS	
BufferMode	Buffering mode	Long (I32)
	1 Aborting	
	2 Buffered	
	3 Blending low	
	4 Blending high	
	5 Blending previous	
	6 Blending next	
TransitionMode	Transition mode in blending mode	Long (I32)
	1 TMNone	
	2 TMStartVelocity	
	3 TMConstantVelocity	
	4 TMCornerDistance	
	5 TMMaxCornerDeviation	
	11 Smooth	
TransitionParameter	Parametr for transition (depends on transition mode)	Double (F64)

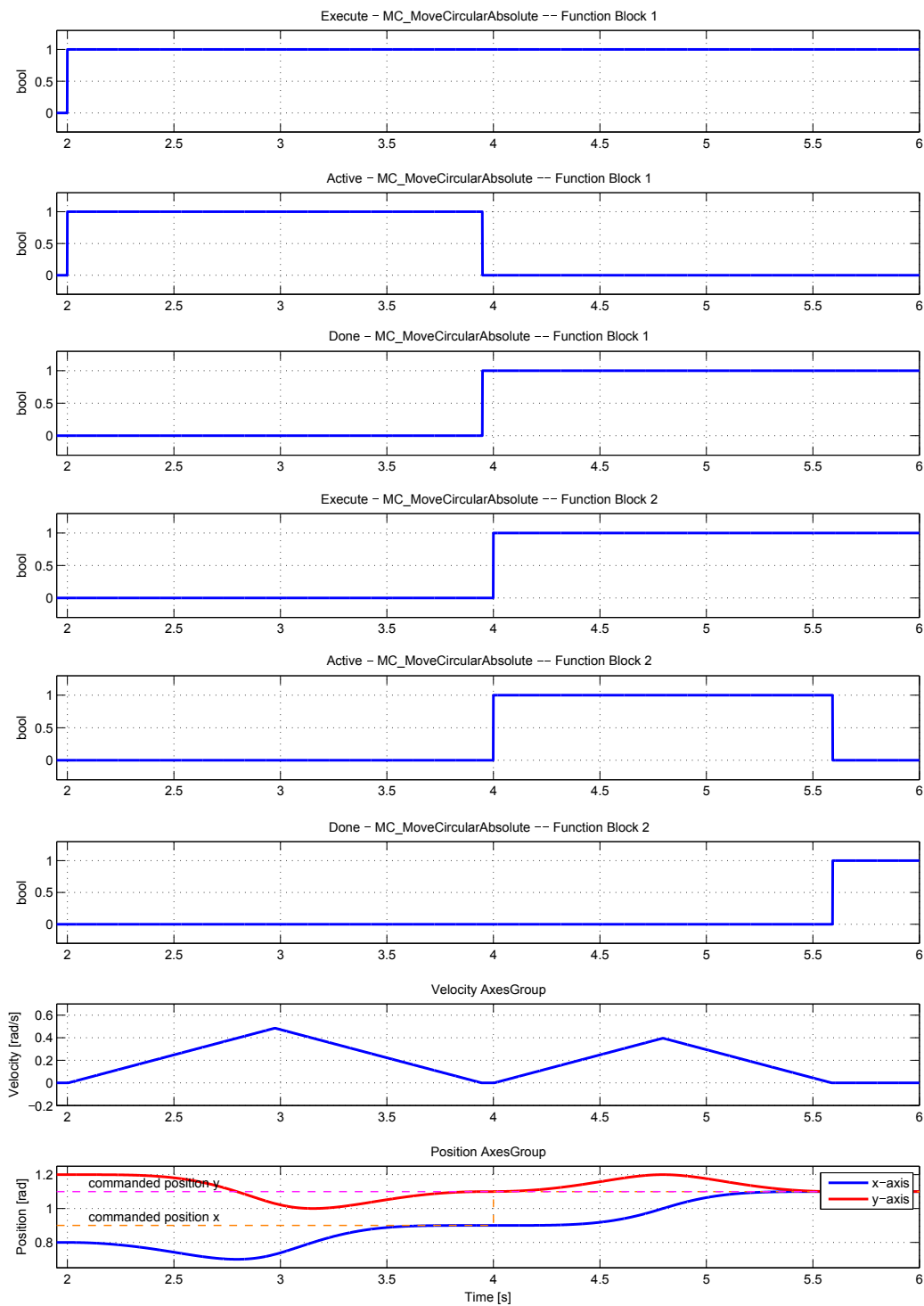
Outputs

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	



MC_MoveCircularAbsolute - Example

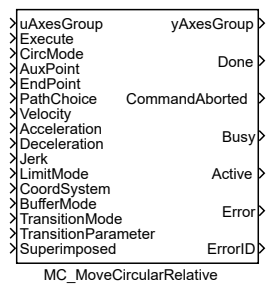




MC_MoveCircularRelative – Circular move to position (relative to execution point)

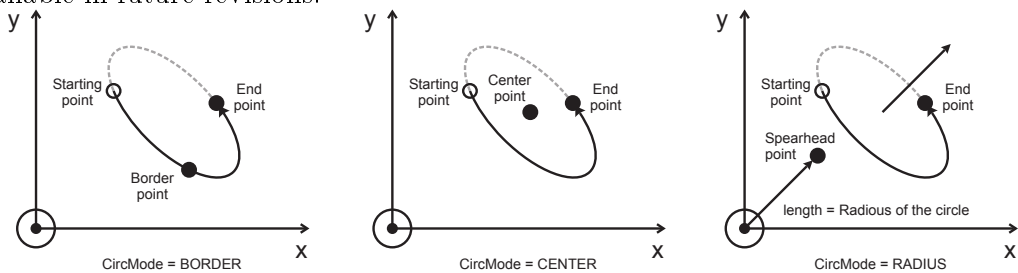
Block Symbol

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.



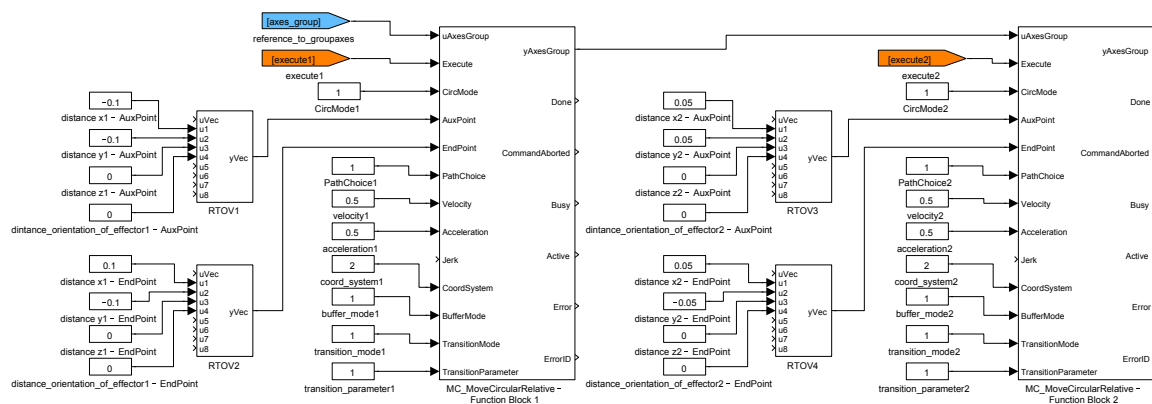
Inputs

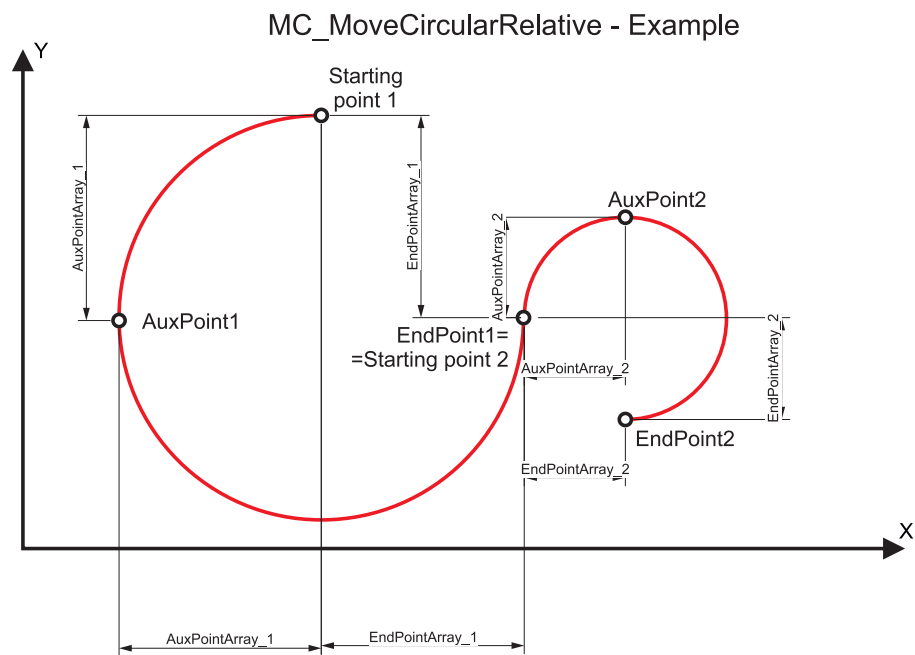
uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool
CircMode	Specifies the meaning of the input signals AuxPoint and CircDirection	Long (I32)
	1 BORDER	
	2 CENTER	
	3 RADIUS	
AuxPoint	Next coordinates to define circle (depend on CircMode)	Reference
EndPoint	Target axes coordinates position	Reference
PathChoice	Choice of path	Long (I32)
	1 Clockwise	
	2 CounterClockwise	
Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s ²]	Double (F64)

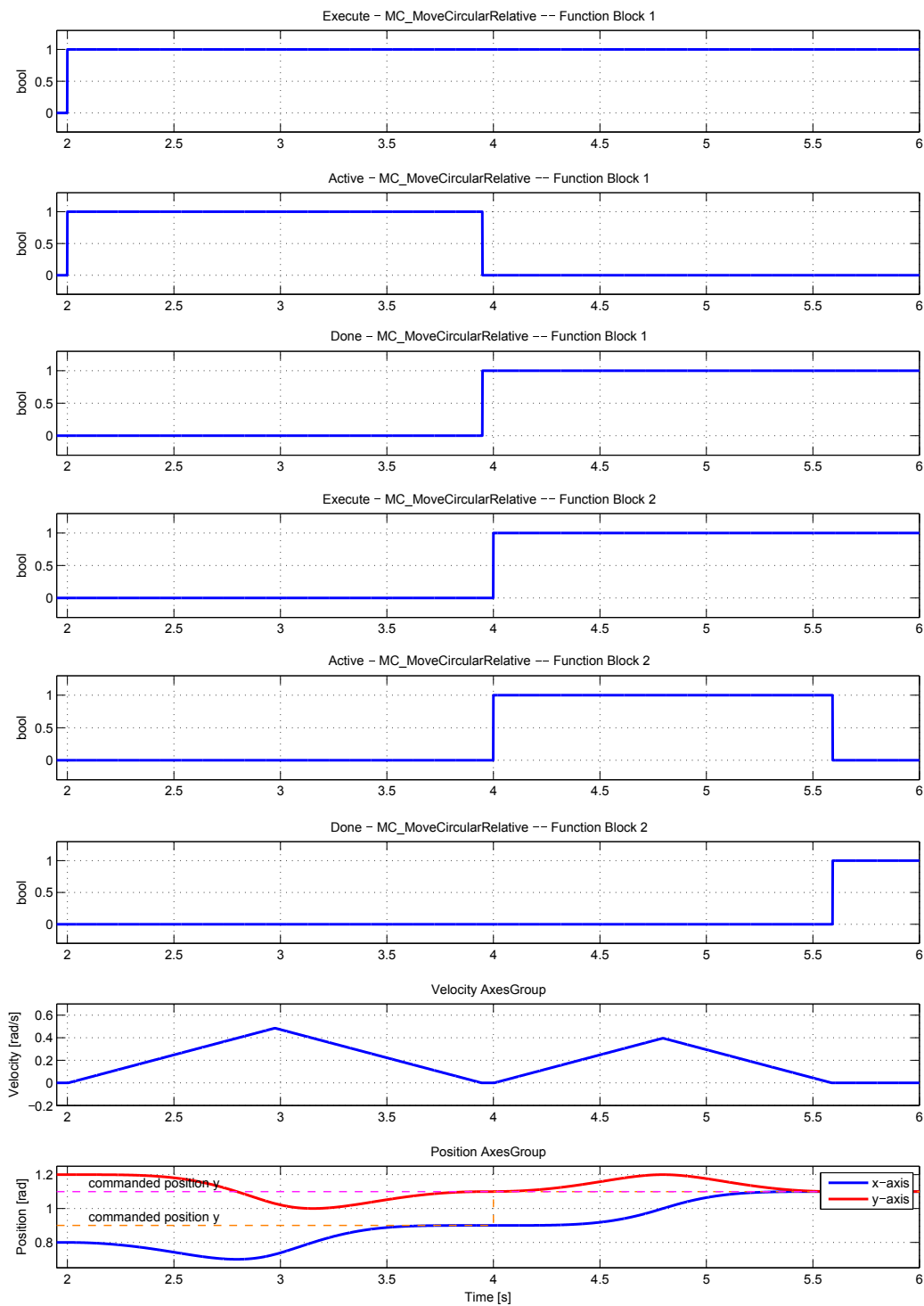
Jerk	Maximal allowed jerk [unit/s ³]	Double (F64)
CoordSystem	Reference to the coordinate system used	Long (I32)
	1 ACS	
	2 MCS	
	3 PCS	
BufferMode	Buffering mode	Long (I32)
	1 Aborting	
	2 Buffered	
	3 Blending low	
	4 Blending high	
	5 Blending previous	
	6 Blending next	
TransitionMode	Transition mode in blending mode	Long (I32)
	1 TMNone	
	2 TMStartVelocity	
	3 TMConstantVelocity	
	4 TMCornerDistance	
	5 TMMaxCornerDeviation	
	11 Smooth	
TransitionParameter	Parametr 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	



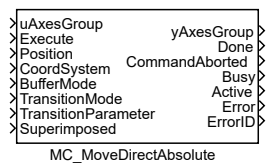




MC_MoveDirectAbsolute – Direct move to position (absolute coordinates)

Block Symbol

Licence: [COORDINATED MOTION](#)



Function Description

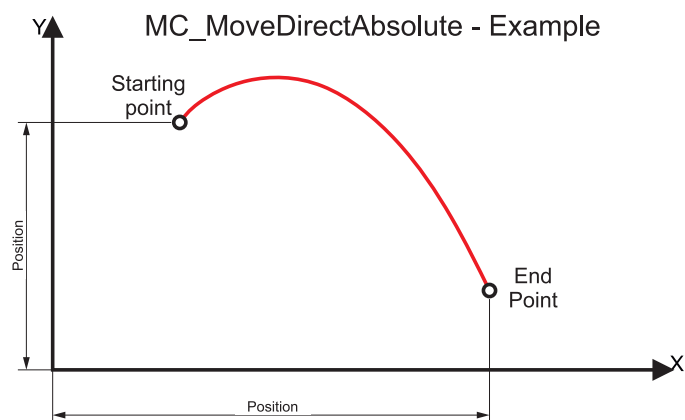
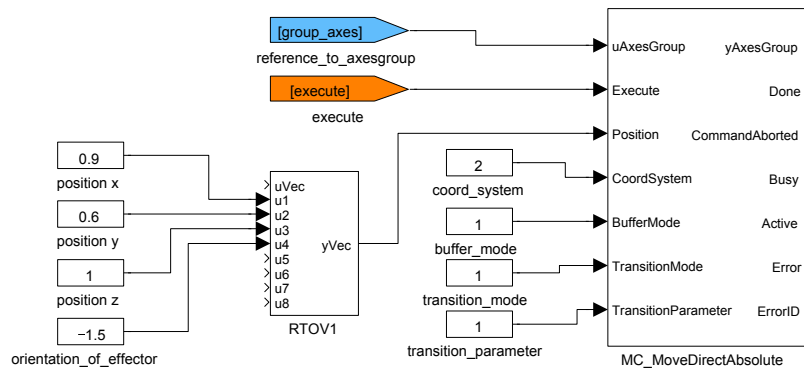
The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

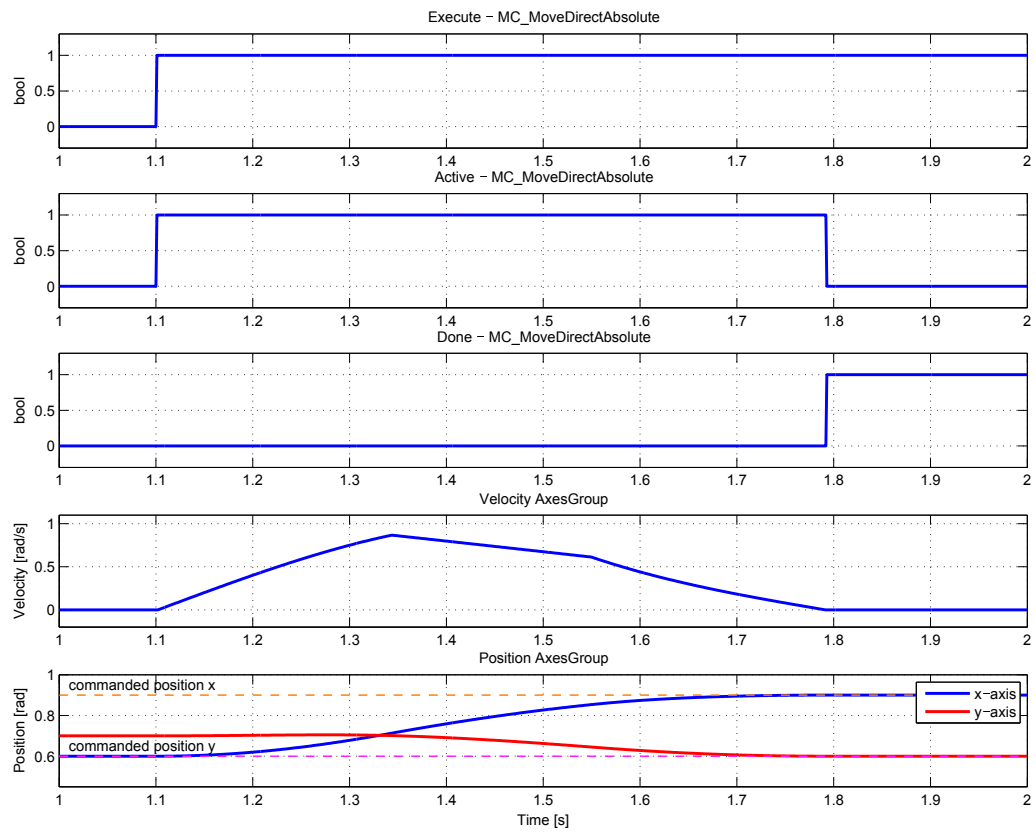
Inputs

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool
Position	Array of coordinates (positions and orientations)	Reference
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	
TransitionMode	Transition mode in blending mode	Long (I32)
	1 TMNone	
	2 TMStartVelocity	
	3 TMConstantVelocity	
	4 TMCornerDistance	
	5 TMMaxCornerDeviation	
	11 Smooth	
TransitionParameter	Parametr 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		

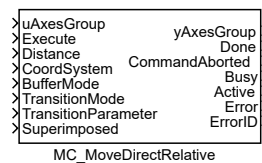




MC_MoveDirectRelative – Direct move to position (relative to execution point)

Block Symbol

Licence: [COORDINATED MOTION](#)



Function Description

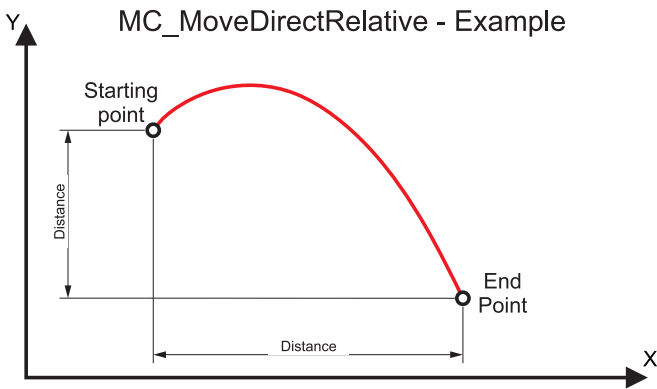
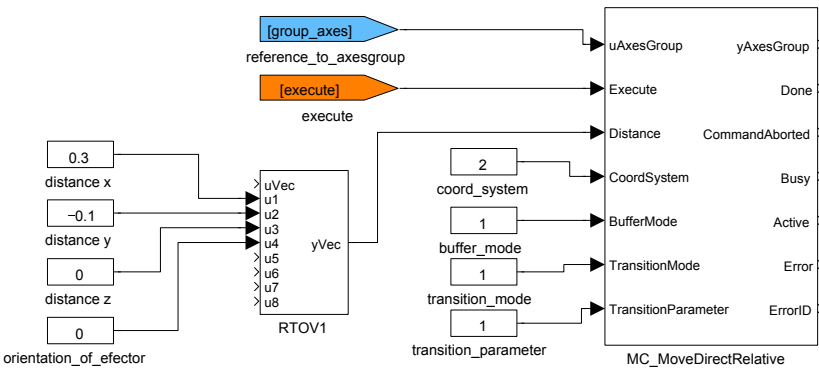
The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

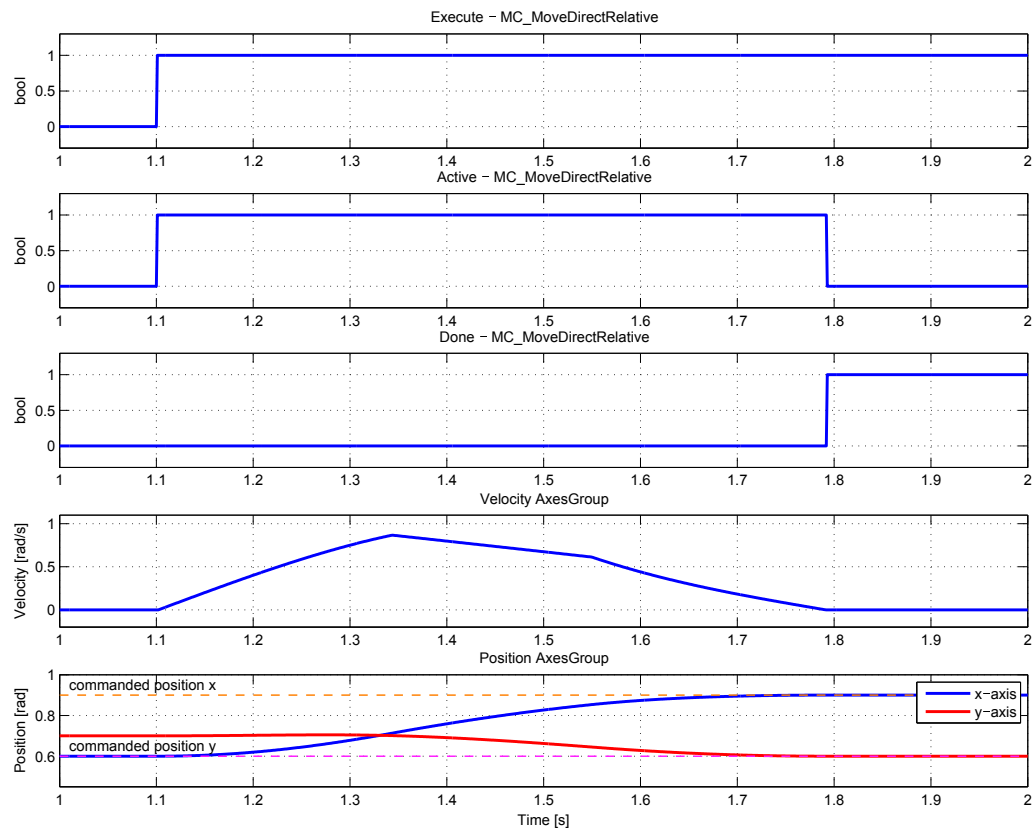
Inputs

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool
Distance	Array of coordinates (relative distances and orientations)	Reference
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	
TransitionMode	Transition mode in blending mode	Long (I32)
	1 TMNone	
	2 TMStartVelocity	
	3 TMConstantVelocity	
	4 TMCornerDistance	
	5 TMMaxCornerDeviation	
	11 Smooth	
TransitionParameter	Parametr 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		

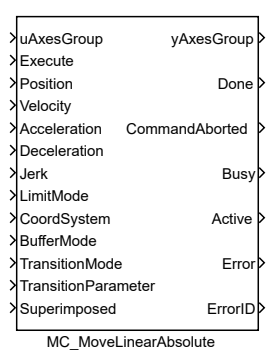




MC_MoveLinearAbsolute – Linear move to position (absolute coordinates)

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.

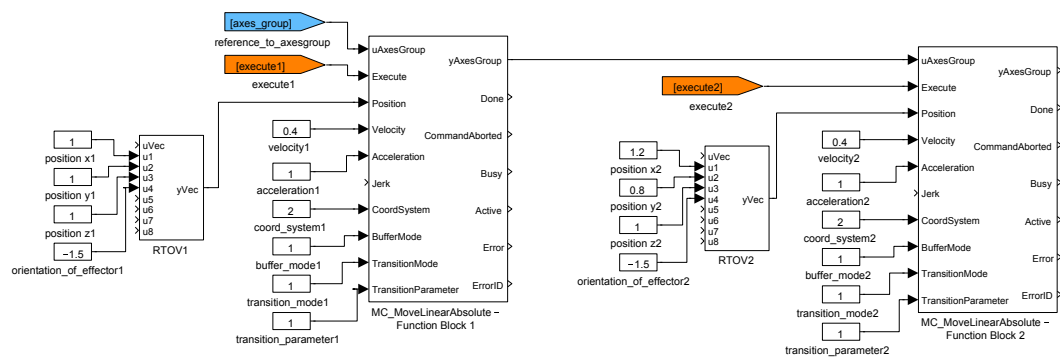
Inputs

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/s ²]	Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]	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	

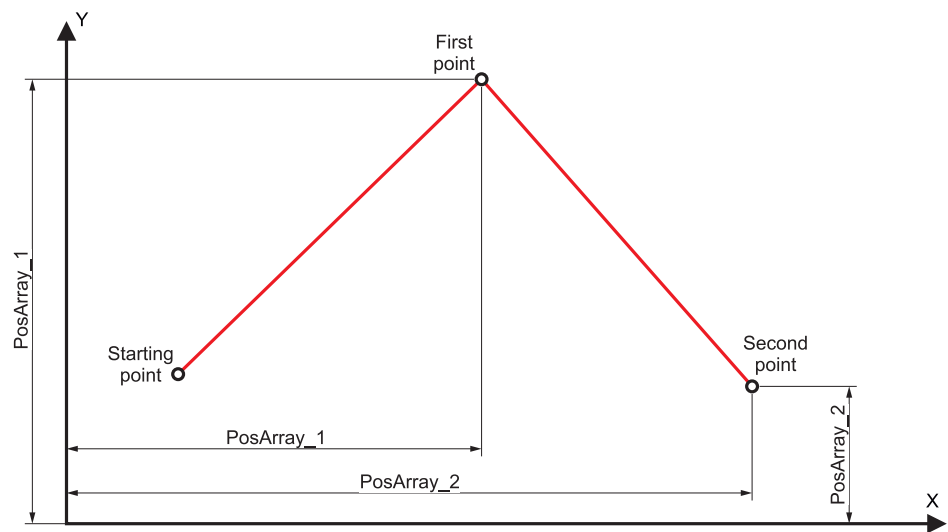
TransitionMode	Transition mode in blending mode	Long (I32)
1 TMNone	
2 TMStartVelocity	
3 TMConstantVelocity	
4 TMCornerDistance	
5 TMMaxCornerDeviation	
11 Smooth	
TransitionParameter	Parametr for transition (depends on transition mode)	Double (F64)

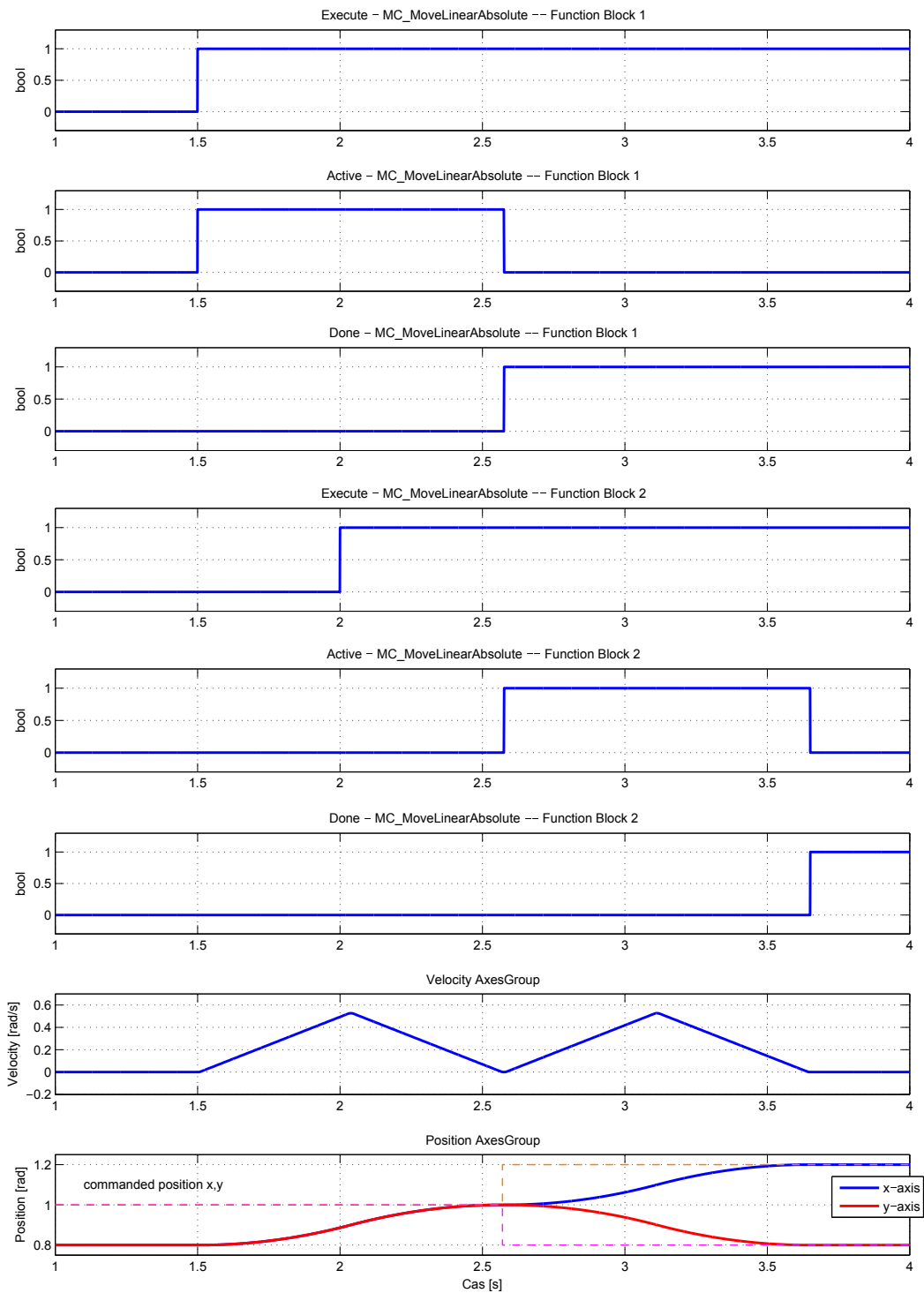
Outputs

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)

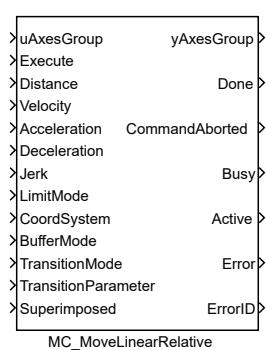




MC_MoveLinearRelative – Linear move to position (relative to execution point)

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.

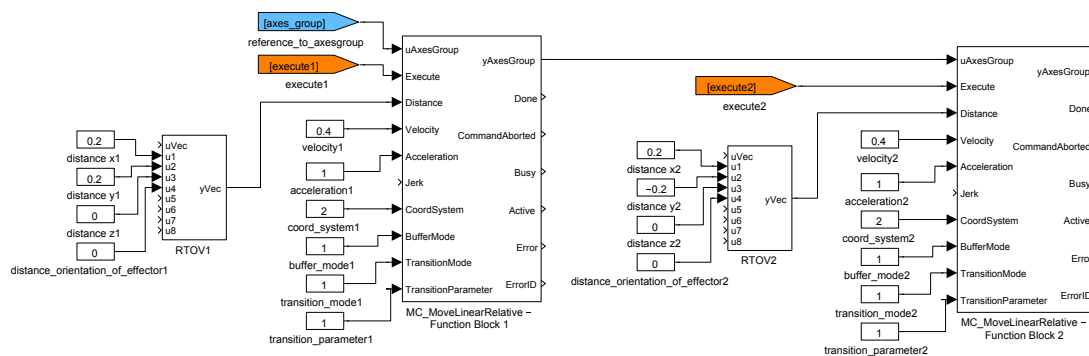
Inputs

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	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	

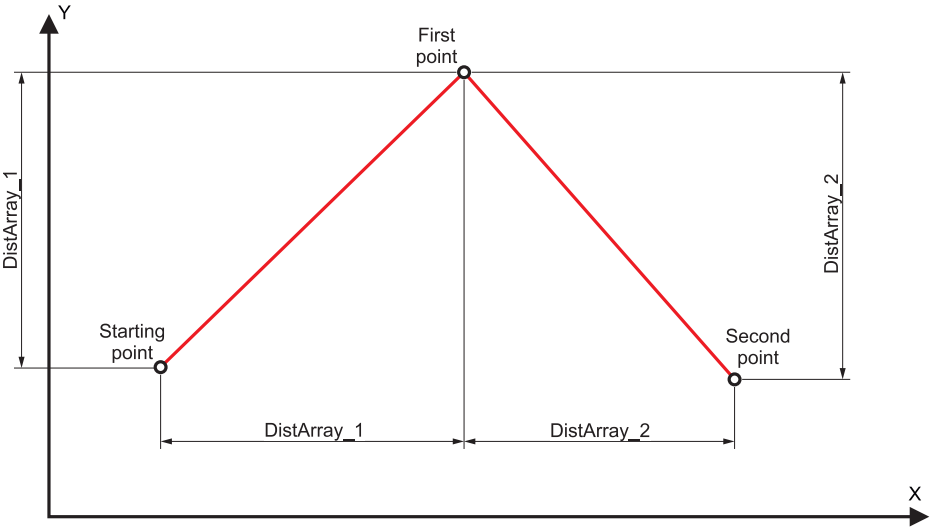
TransitionMode	Transition mode in blending mode	Long (I32)
1 TMNone	
2 TMStartVelocity	
3 TMConstantVelocity	
4 TMCornerDistance	
5 TMMaxCornerDeviation	
11 Smooth	
TransitionParameter	Parametr for transition (depends on transition mode)	Double (F64)

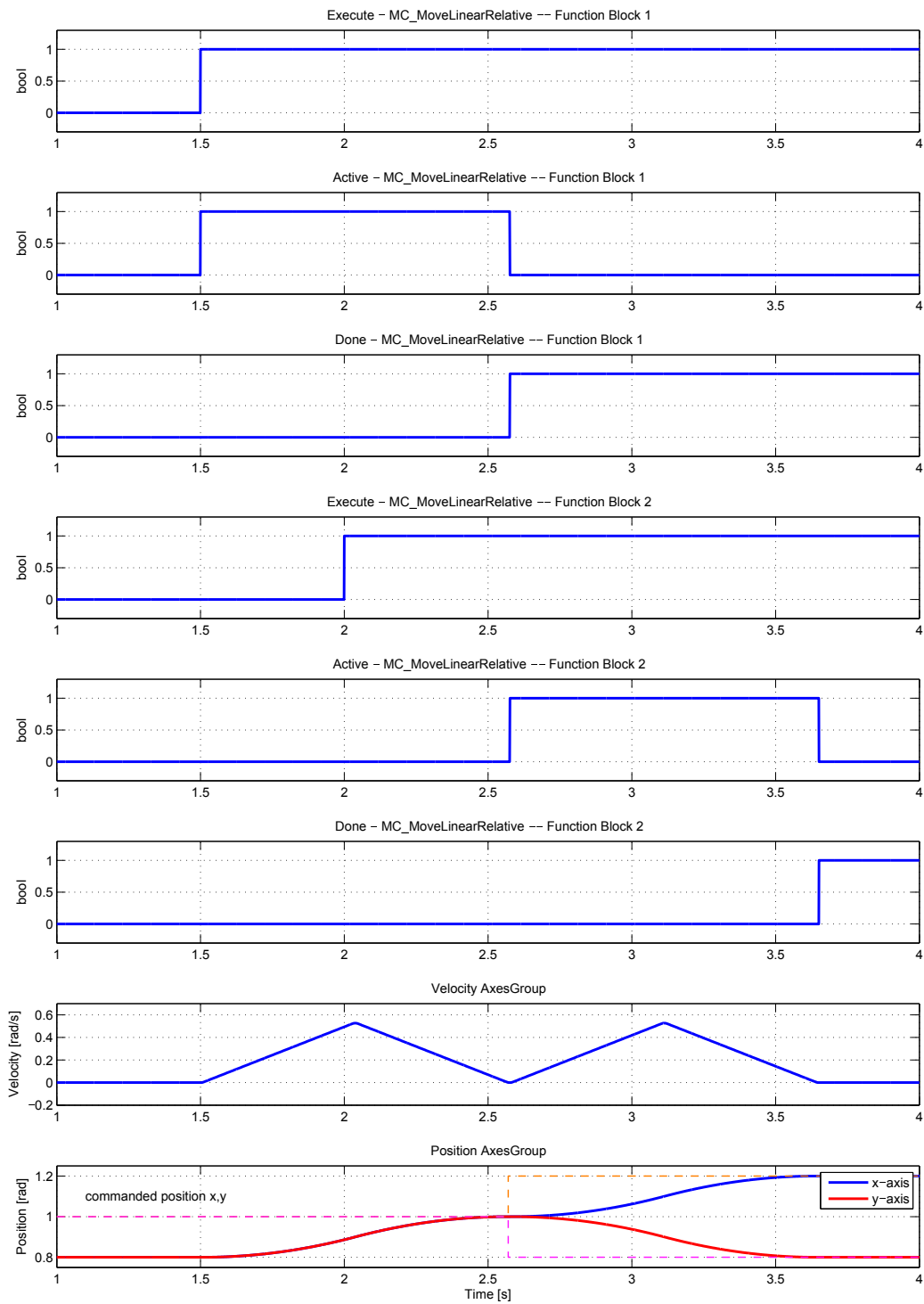
Outputs

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)

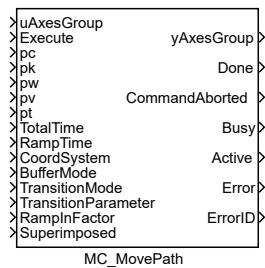




MC_MovePath – General spatial trajectory generation

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
TotalTime	Time [s] for whole move	Double (F64)
RampTime	Time [s] for acceleration/deceleration	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	
TransitionMode	Transition mode in blending mode	Long (I32)
	1 TMNone	
	2 TMStartVelocity	
	3 TMConstantVelocity	
	4 TMCornerDistance	
	5 TMMaxCornerDeviation	
	11 Smooth	
TransitionParameter	Parametr for transition (depends on transition mode)	Double (F64)

RampIn	RampIn factor (0 = RampIn mode not used)	Double (F64)
---------------	--	--------------

Parameters

pc	Control-points matrix $\odot[0.0 \ 1.0 \ 2.0; \ 0.0 \ 1.0 \ 1.0; \ 0.0 \ 1.0 \ 0.0]$	Double (F64)
pk	Knot-points vector $\odot[0.0 \ 0.0 \ 0.0 \ 0.0 \ 0.5 \ 1.0 \ 1.0]$	Double (F64)
pw	Weighting vector $\odot[1.0 \ 1.0 \ 1.0]$	Double (F64)
pv	Polynoms for feedrate definition $\odot[0.0 \ 0.05 \ 0.95; \ 0.0 \ 0.1 \ 0.1; \ 0.0 \ 0.0 \ 0.0; \ 0.1 \ 0.0 \ -0.1; \ -0.05 \ 0.0 \ 0.05; \ 0.0 \ 0.0 \ 0.0]$	Double (F64)
pt	Knot-points (time [s]) for feedrate $\odot[0.0 \ 1.0 \ 10.0 \ 11.0]$	Double (F64)
user	Only for special edit $\odot[0.0 \ 1.0 \ 2.0 \ 3.0]$	Double (F64)

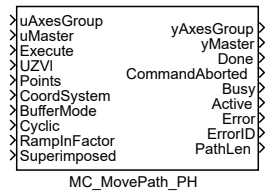
Outputs

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 i REXYGEN general error	Error

MC_MovePath_PH – * General spatial trajectory generation PH

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.

This block does not propagates the signal quality. More information can be found in the [1.4](#) section.

Input

uAxesGroup	Axes group reference	Reference
uMaster	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
CoordSystem	Reference to the coordinate system used	Long (I32)
	1 ACS	
	2 MCS	
	3 PCS	
	4 TCS	
BufferMode	Buffering mode	Long (I32)
	1 Aborting	
	2 Buffered	
	3 Blending low	
	4 Blending high	
	5 Blending previous	
	6 Blending next	
Cyclic	Profile is cyclic flag	Bool
RampIn	RampIn factor (0 = RampIn mode not used)	Double (F64)
UZV1		Reference
Points		Reference

Parameter

Superimposed	start as superimposed motion flag	Bool
--------------	-----------------------------------	------

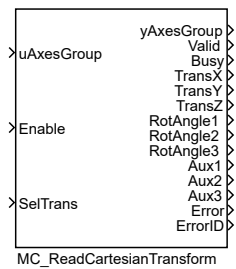
Output

yAxesGroup	Axes group reference	Reference
yMaster	Axis reference (only RM_Axis.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 last operation	Error
PathLen		Double (F64)

MC_ReadCartesianTransform – Reads the parameter of the cartesian transformation

Block Symbol

Licence: COORDINATED MOTION



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

uAxesGroup	Axes group reference	Reference
Enable	Block function is enabled	Bool

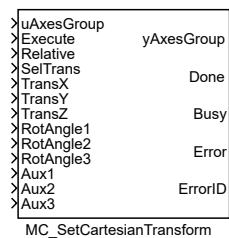
Outputs

yAxesGroup	Axes group reference	Reference
Valid	Output value is valid	Bool
Busy	Algorithm not finished yet	Bool
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	Error
i REXYGEN general error		

MC_SetCartesianTransform – Sets Cartesian transformation

Block Symbol

Licence: [COORDINATED MOTION](#)



Function Description

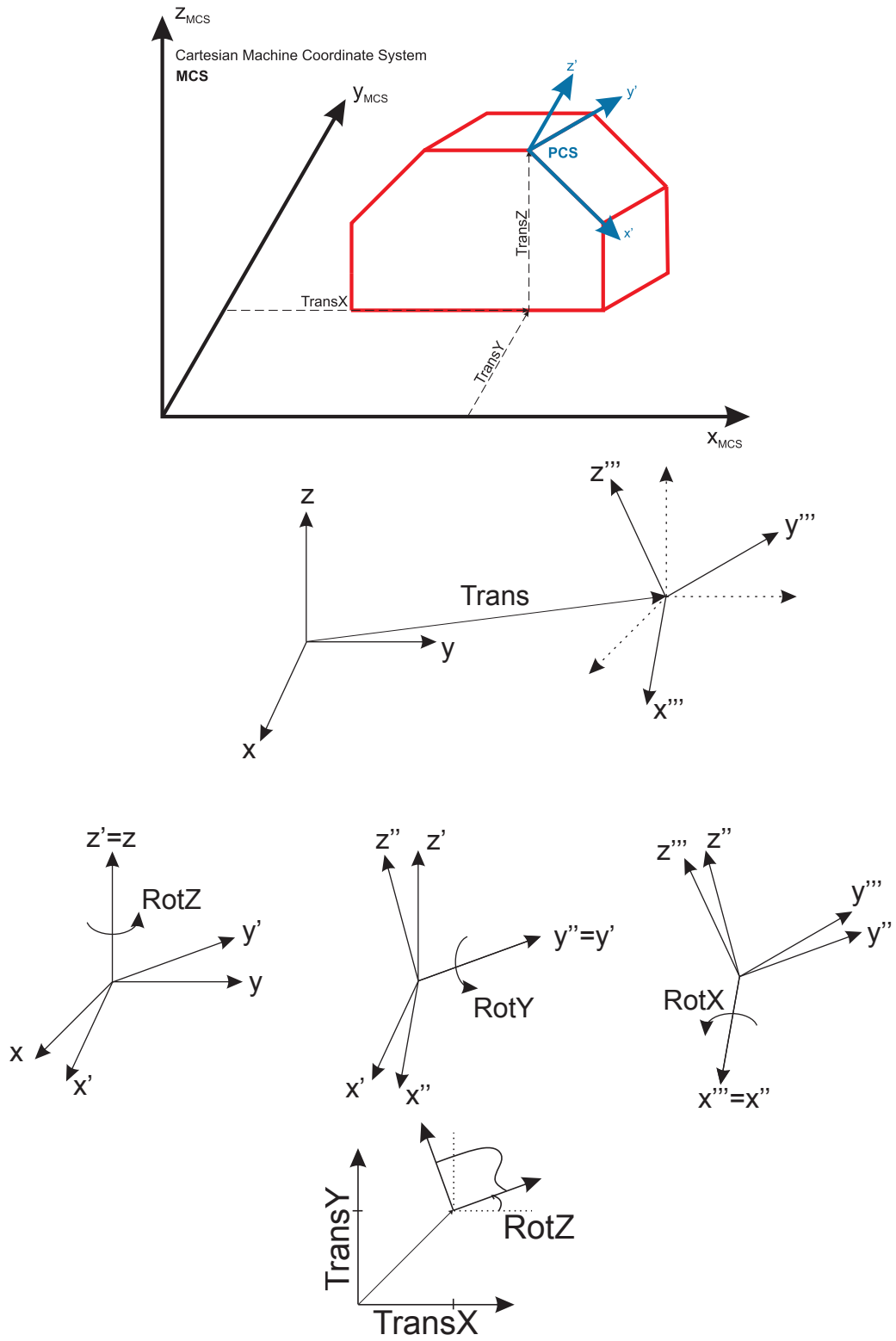
The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

		Reference
uAxesGroup	Axes group reference	
Execute	The block is activated on rising edge	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)
Relative	Mode of position inputs	Bool

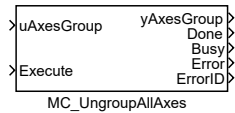
Outputs

		Reference
yAxesGroup	Axes group reference	
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
	i REXYGEN general error	



MC_UngroupAllAxes – Removes all axes from the group

Block Symbol Licence: COORDINATED MOTION



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

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

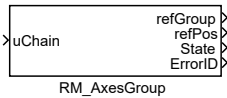
Outputs

yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
	i REXYGEN general error	

RM_AxesGroup – Axes group for coordinated motion control

Block Symbol

Licence: COORDINATED MOTION



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.

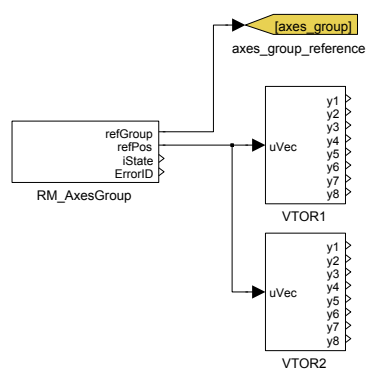
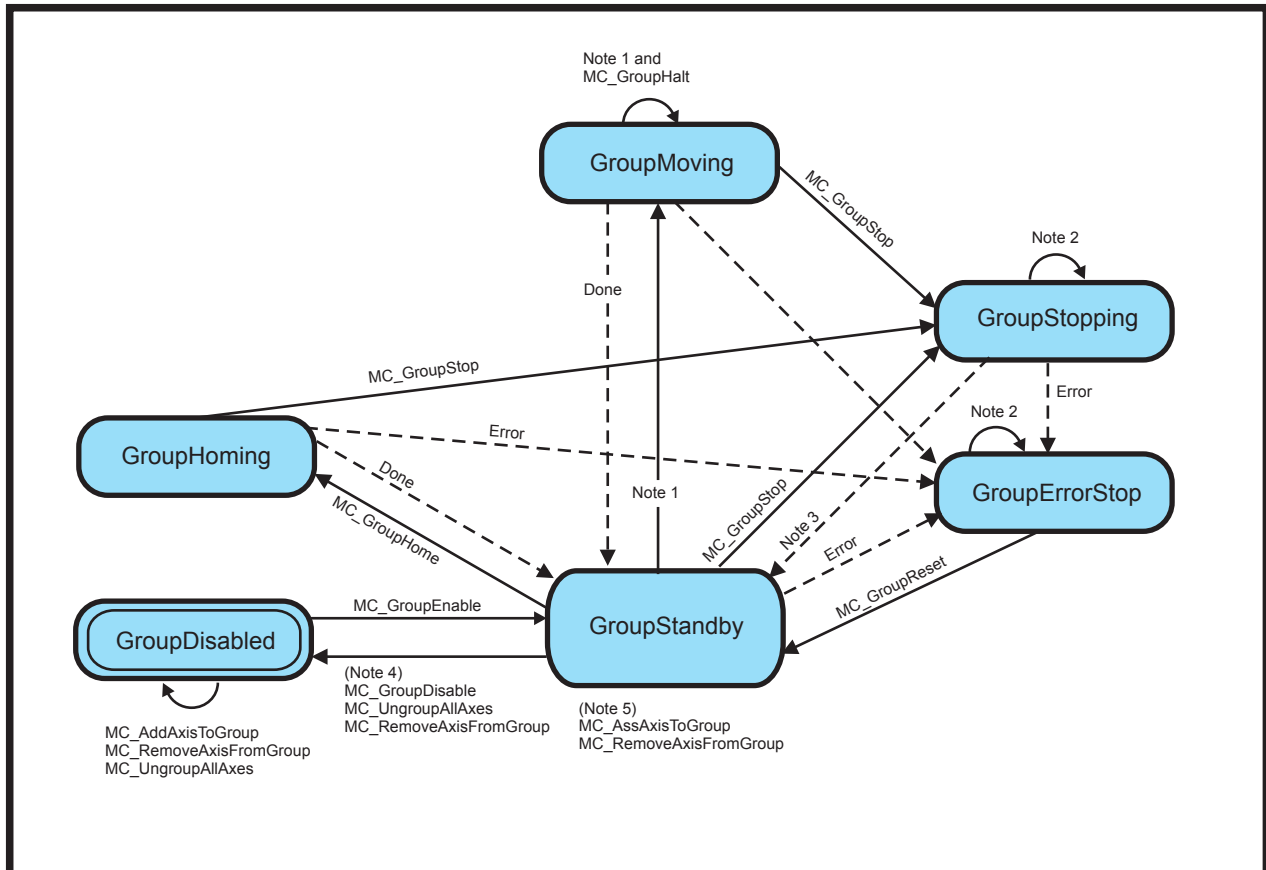
Parameters

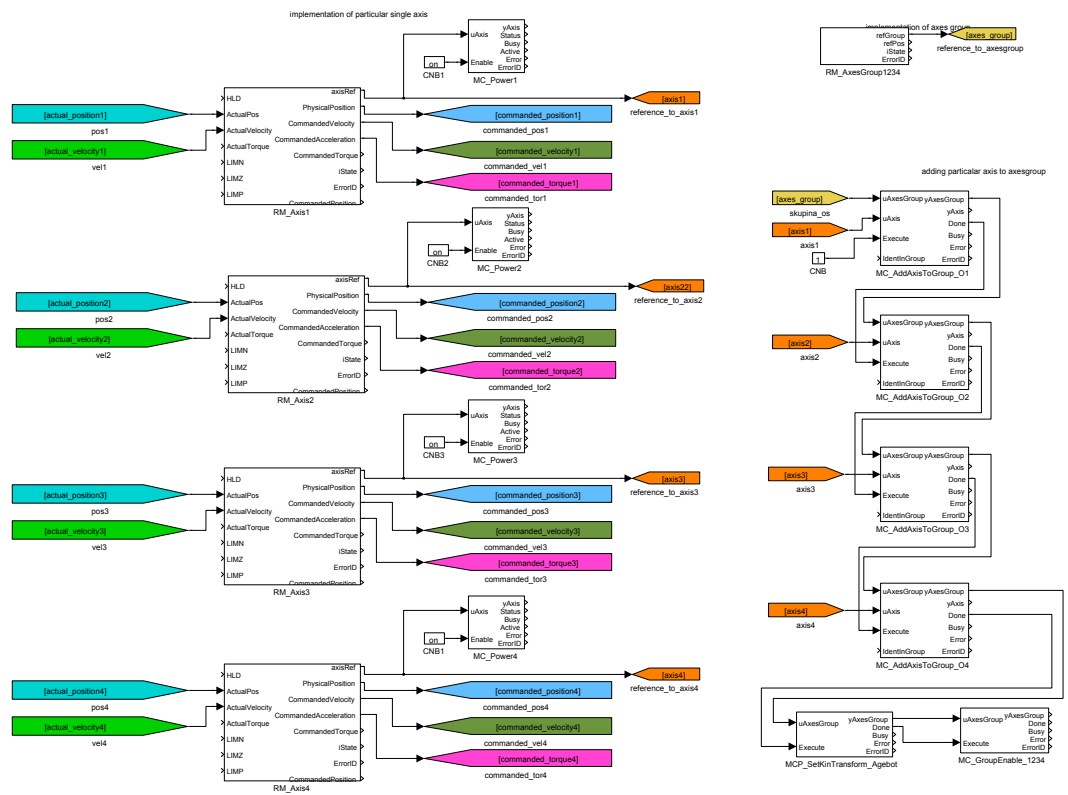
McsCount	Number of axis in MCS	↓1 ↑6 ⊙6	Long (I32)
AcsCount	Number of axis in ACS	↓1 ↑16 ⊙6	Long (I32)
PosCount	Number of position axis	↓1 ↑6 ⊙3	Long (I32)
Velocity	Maximal allowed velocity [unit/s]		Double (F64)
Acceleration	Maximal allowed acceleration [unit/s ²]		Double (F64)
Jerk	Maximal allowed jerk [unit/s ³]		Double (F64)

Outputs

refGroup	Axes group reference	Reference
refPos	Position, velocity and acceleration vector	Reference
iState	Group status	Long (I32)
	0 Disabled	
	1 Standby	
	2 Homing	
	6 Moving	
	7 Stopping	
	8 Error stop	
ErrorID	Result of the last operation	Error
	i REXYGEN general error	

The State Diagram of AxesGroup

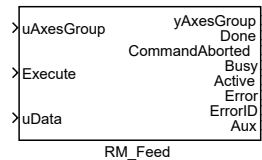




RM_Feed — * MC Feeder ???

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	↓0.01 ↑100.0 ⊙1.0	Double (F64)
Relative	0		Bool
CoordSystem	0	↓1 ↑3 ⊙2	Long (I32)
BufferMode	0	↓1 ↑6 ⊙1	Long (I32)
TransitionMode	0	↓0 ↑15 ⊙1	Long (I32)
TransitionParameter	0		Double (F64)

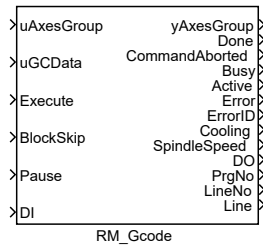
Outputs

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

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
BlockSkip	MILAN	Bool

Parameters

BaseDir	Directory of the G-code files	String
MainFile	Source file number	Long (I32)
CoordSystem	0	↓1 ↑3 ⊙3 Long (I32)
BufferMode	Buffering mode	⊙1 Long (I32)
	1 Aborting	
	2 Buffered	
	3 Blending low	
	4 Blending high	
	5 Blending previous	
	6 Blending next	
TransitionMode	Transition mode in blending mode	⊙1 Long (I32)
	1 TMNone	
	2 TMStartVelocity	
	3 TMConstantVelocity	
	4 TMCornerDistance	
	5 TMMaxCornerDeviation	
	11 Smooth	
TransitionParameter	Parametr for transition (depends on transition mode)	Double (F64)

workOffsets	Sets with initial coordinate	$\odot[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$	Double (F64)
toolOffsets	Sets of tool offset	$\odot[0\ 0\ 0]$	Double (F64)
cutterOffsets	Tool radii	$\odot[0\ 0\ 0]$	Double (F64)

Outputs

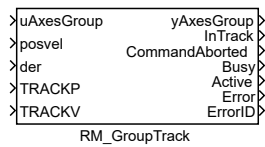
yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
CommandAborted	Algorithm was aborted	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
	i REXYGEN general error	
Cooling	Cooling	Bool
LineNo	Current executed line number	Long (I32)
Line	Current line of G-code	String
SpindleSpeed	Spindle speed	Double (F64)

RM_GroupTrack – **T**

racking position/velocity

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
posvel	Vector of desired position or velocity	Reference
TRACKP	Position tracking	Bool
TRACKV	Velocity tracking	Bool

Parameters

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	⊙2 Long (I32)
	1 ACS	
	2 MCS	
	3 PCS	
iLen	Number of samples to estimate the velocity / acceleration	Long (I32)
	↓-1 ↑99	

Outputs

yAxesGroup	Axes group reference	Reference
InTrack	Position / velocity track flag	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	

Chapter 23

CanDrv – Communication via CAN bus

Contents

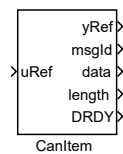
CanItem – Secondary received CAN message	876
CanRecv – Receive CAN message	878
CanSend – Send CAN message	880

The CanDrv library is dedicated to handling CAN (Controller Area Network) bus communication in REXYGEN system. It features [CanItem](#) for managing CAN data items, [CanRecv](#) for receiving messages from the bus, and [CanSend](#) for sending messages. This library provides essential tools for efficient and reliable communication over CAN networks, facilitating data exchange and control commands between various system components.

CanItem – Secondary received CAN message

Block Symbol

Licence: [CANDRV](#)



Function Description

The **CanItem** block is used with the [CanRecv](#) block. The **uRef** input 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**).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

uRef	Secondary received packet reference	Reference
-------------	-------------------------------------	-----------

Output

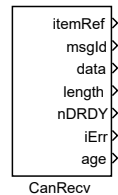
yRef	Secondary received packet reference	Reference
msgId	CAN message ID (COB-ID)	Long (I32)

data	Message data (8 bytes maximum, LSB first)		Large (I64)
		↓-9.22337E+18 ↑9.22337E+18	
length	Message length (number of bytes of data)	↓0 ↑8	Long (I32)
DRDY	Received message in the last period flag		Bool

CanRecv – Receive CAN message

Block Symbol

Licence: [CANDRV](#)



Function Description

The **CanRecv** block receives message via CAN bus. The block accepts only messages that match the filter (parameters **filterId**, **filterIdMask**, **filterLength**, **RTR**, **EXT**).

Number of messages received in the current task period (i.e. since the previous execution) is indicated by the **nRDY** 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. [Goto](#), [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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Parameter

filterId	MessageId of packets to receive by this block	↓0 ↑536870911	Long (I32)
filterIdMask	Mask for the filterId parameter (marks valid bits)	↓0 ↑536870911	Long (I32)
filterLength	Data length of packets to receive by this block (-1 allows all lengths)	↓-1 ↑8	Long (I32)
RTR	Request To Receive flag	⊙on	Bool
EXT	Extended message ID (29bits)	⊙on	Bool
timeout	Error is indicated if no packet is received within the timeout interval [s]	↓0.0	Double (F64)
nmax	Maximum number of received messages in one period	↓1 ↑255	Long (I32)

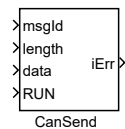
Output

<code>itemRef</code>	Secondary received packet reference		Reference
<code>msgId</code>	CAN message ID (COB-ID)		Long (I32)
<code>data</code>	Message data (8 bytes maximum, LSB first)		Large (I64)
		↓-9.22337E+18 ↑9.22337E+18	
<code>length</code>	Message length (number of bytes of data)	↓0 ↑8	Long (I32)
<code>nDRDY</code>	Number of received messages in the last period	↑255	Word (U16)
<code>iErr</code>	Error code		Error
<code>age</code>	Elapsed time since the last received message [s]	↓0.0	Double (F64)

CanSend – Send CAN message

Block Symbol

Licence: [CANDRV](#)



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 **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>_<signal>** (see e.g. [Goto](#), [OUTQUAD](#) or [OUTOCT](#) blocks for details about referencing data from I/O drivers). The **<signal>** 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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

msgId	CAN message ID (COB-ID)	↓0 ↑536870911	Long (I32)
length	Message length (number of bytes of data)	↓0 ↑8	Long (I32)
data	Message data (8 bytes maximum, LSB first)	↓-9.22337E+18 ↑9.22337E+18	Large (I64)
RUN	Sending message is enabled		Bool

Parameter

RTR	Request To Receive flag	⊙on	Bool
EXT	Extended message ID (29bits)	⊙on	Bool

Output

iErr	Error code	Error
------	------------	-------

Chapter 24

OpcUaDrv – Communication using OPC UA

Contents

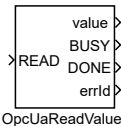
OpcUaReadValue – Read value from OPC UA Server	882
OpcUaServerValue – Expose value as an OPC UA Node	884
OpcUaWriteValue – Write value to OPC UA Server	886

The OpcUaDrv library is specialized in interfacing with OPC UA (Open Platform Communications Unified Architecture) servers for industrial automation. The first block – [OpcUaReadValue](#) is designed for reading data from servers, making it pivotal for data acquisition in automated systems. The [OpcUaWriteValue](#) block enables writing data to servers, allowing for control and command execution. Additionally, the [OpcUaServerValue](#) block facilitates the monitoring and management of server values. This library serves as a critical tool for seamless communication and interaction with OPC UA servers, enhancing the capabilities of automation systems.

OpcUaReadValue – Read value from OPC UA Server

Block Symbol

Licence: [ADVANCED](#)



Function Description

This function block depends on the OpUa driver. Please read the `OpUaDrv` manual [\[26\]](#) before use.

The `OpUaReadValue` 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 type 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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>READ</code>	Enable execution	<code>Bool</code>
-------------------	------------------	-------------------

Parameter

<code>NodeId</code>	OPC UA Node Id	<code>String</code>
---------------------	----------------	---------------------

NodeId_type	Type of Node ID	⊙1	Long (I32)
	1 string		
	2 numeric		
	3 guid		
	4 path		
type	Expected type of incoming data	⊙1	Long (I32)
	1 Bool		
	2 Byte (U8)		
	3 Short (I16)		
	4 Long (I32)		
	5 Word (U16)		
	6 DWord (U32)		
	7 Float (F32)		
	8 Double (F64)		
	10 Large (I64)		
	12 String		

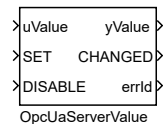
Output

value	Output signal	Any
BUSY	Busy flag	Bool
DONE	Indicator of finished transaction	Bool
errId	Error code	Error

OpUaServerValue – Expose value as an OPC UA Node

Block Symbol

Licence: [ADVANCED](#)



Function Description

This function block depends on the OpCua driver. Please read the `OpCuaDrv` manual [\[26\]](#) before use.

The `OpUaServerValue` block exposes an OPC UA Node through the `OpCuaDrv` driver in the OPC UA Server mode.

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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>uValue</code>	Input signal	Any
<code>SET</code>	Set the input value to OPC UA Node value	Bool
<code>DISABLE</code>	Disable OPC UA Node	Bool

Parameter

<code>NodeId</code>	OPC UA Node Id	String
<code>NodeId_type</code>	OPC UA Node Id type	⊙1 String
	1 string	
	2 numeric	
	3 guid	

type	Value data type	⊙1	Long (I32)
	1 Bool		
	2 Byte (U8)		
	3 Short (I16)		
	4 Long (I32)		
	5 Word (U16)		
	6 DWord (U32)		
	7 Float (F32)		
	8 Double (F64)		
	10 Large (I64)		
	12 String		
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	Bool

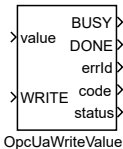
Output

yValue	Output signal	Any
CHANGED	Value of the node changed though 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 [\[26\]](#) before use.

The `OpcUaWriteValue` block writes value to the OPC UA Node through a connection established by the `OpcUaDrv`.

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 type 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 input signal `value` is converted to the specified type and is than 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.

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>value</code>	Input signal	<code>Any</code>
<code>WRITE</code>	Enable execution	<code>Bool</code>

Parameter

<code>NodeId</code>	OPC UA Node Id	<code>String</code>
---------------------	----------------	---------------------

NodeId_type	Type of Node ID	⊙1	Long (I32)
	1 string		
	2 numeric		
	3 guid		
	4 path		
type	Value data type	⊙1	Long (I32)
	1 Bool		
	2 Byte (U8)		
	3 Short (I16)		
	4 Long (I32)		
	5 Word (U16)		
	6 DWord (U32)		
	7 Float (F32)		
	8 Double (F64)		
	10 Large (I64)		
	12 String		

Output

BUSY	Busy flag	Bool
DONE	Indicator of finished transaction	Bool
errId	Error code	Error
code	OPC UA result status code	DWord (U32)
status	OPC UA result status string	String

Chapter 25

UNIPi – Communication blocks for Unipi

Contents

IM201CNT – Iris IM201CNT, 4 digital counters	891
IM201DI – Iris IM201DI, 4 digital inputs	892
IM203DO – Iris IM203DO, 8 digital outputs	893
IM203PWM – Iris IM203PWM, 8 PWM outputs	894
IM204CNT – Iris IM204CNT, 16 digital counters	895
IM204DI – Iris IM204DI, 16 digital inputs	897
IM205DO – Iris IM205DO, 16 digital outputs	898
IM205PWM – Iris IM205PWM, 16 PWM outputs	899
IM301CNT – Iris IM301CNT, 2 digital counters	901
IM301DI – Iris IM301DI, 2 digital inputs	902
IM301DO – Iris IM301DO, 2 digital outputs	903
IM502AO – Iris IM502AO, 4 analog outputs	904
IM503AI – Iris IM503AI, 8 analog inputs	905
IM504RI – Iris IM504RI, 8 resistance or temperature inputs	907
IM506AI – Iris IM506AI, 2 analog inputs	909
IM506AO – Iris IM506AO, 1 analog output	911
IRIS_MODULE – Iris - Module description info	912
UNIPi_CHANNEL – Iris module or Patron section info	913
UNIPi_PRODUCT – Product description info	915
UNIPi_S1AI – Patron section 1, analog input	916
UNIPi_S1AOR – Patron section 1, analog output or resistance input	917
UNIPi_S1CNT – Patron section 1, counters	918
UNIPi_S1DI – Patron section 1, digital inputs	919
UNIPi_S1DO – Patron section 1, digital outputs	920

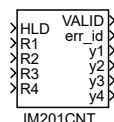
UNIPi_S1LED – Patron section 1, LED outputs	921
UNIPi_S1PWM – Patron section 1, PWM outputs	922
UNIPi_S2AI – Patron section 2, analog inputs	923
UNIPi_S2AO – Patron section 2, analog outputs	924
UNIPi_S2CNT – Patron section 2, counters	925
UNIPi_S2DI – Patron section 2, digital inputs	926
UNIPi_S2RO – Patron section 2, relay outputs	927

This library is used to control and monitor Unipi devices. It includes blocks for reading and writing digital and analog inputs and outputs, blocks for controlling relays, PWM outputs, LED diodes and reading counters. For reading buses, drivers such as OwsDrv or MbDrv can be used. The blocks work in accordance with the manufacturer's documentation [1], where technical details about individual devices and their inputs and outputs can be found.

IM201CNT – Iris IM201CNT, 4 digital counters

Block Symbol

Licence: [UNIPi](#)



Function Description

The IM201CNT block enables the use of digital inputs as counters on the Unipi Iris module. For more details, visit the manufacturer's website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
R1..R4	Reset counter	Bool

Parameter

slot	Slot	Long (I32)
0	Not used	
12	Slot #12	
22	Slot #22	
32	Slot #32	
42	Slot #42	
52	Slot #52	
11	Slot #11	
21	Slot #21	
debounce1..debounce4	Debounce time in 100 μ s	↓0 ↑65535 ⊙1 Long (I32)
reset_value1..reset_value4	Counter reset value	↑4294967295 Word (U16)

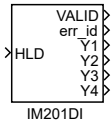
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
y1..y4	Counter value	↑4294967295 Word (U16)

IM201DI – Iris IM201DI, 4 digital inputs

Block Symbol

Licence: [UNIFI](#)



Function Description

The IM201DI block enables reading digital inputs on the Unipi Iris module. For more details, visit the manufacturer’s website [1].

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

HLD	Hold	Bool
-----	------	------

Parameter

slot	Slot	Long (I32)
	0 Not used	
	12 Slot #12	
	22 Slot #22	
	32 Slot #32	
	42 Slot #42	
	52 Slot #52	
	11 Slot #11	
	21 Slot #21	
debounce1..debounce4	Debounce time in 100 μ s	\downarrow 0 \uparrow 65535 \odot 1 Long (I32)

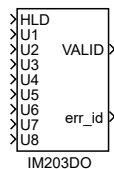
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
Y1..Y4	Digital input	Bool

IM203DO – Iris IM203DO, 8 digital outputs

Block Symbol

Licence: [UNIPi](#)



Function Description

The **IM203DO** block enables the use of digital outputs on the Unipi Iris module. For more details, visit the manufacturer's website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
U1..U8	Digital output	Bool

Parameter

slot	Slot	Long (I32)
0	Not used	
12	Slot #12	
22	Slot #22	
32	Slot #32	
42	Slot #42	
52	Slot #52	
11	Slot #11	
21	Slot #21	

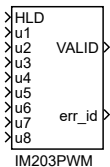
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)

IM203PWM – Iris IM203PWM, 8 PWM outputs

Block Symbol

Licence: [UNIFI](#)



Function Description

The IM203PWM block is used to control the pulse-width modulation (PWM) outputs on the Unipi Iris module. For more details, visit the manufacturer’s website [1].

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

HLD	Hold	Bool
u1..u8	PWM output	Word (U16)

Parameter

slot	Slot	Long (I32)
	0 Not used	
	12 Slot #12	
	22 Slot #22	
	32 Slot #32	
	42 Slot #42	
	52 Slot #52	
	11 Slot #11	
	21 Slot #21	
freq	Frequency	Long (I32)
	0 1kHz	
	1 100Hz	
	2 Custom prescaler	
prescaler	Prescaler	↓0 ↑65535 Long (I32)

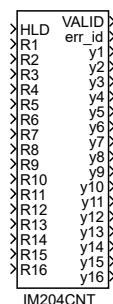
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)

IM204CNT – Iris IM204CNT, 16 digital counters

Block Symbol

Licence: [UNIPi](#)



Function Description

The IM204CNT block enables the use of digital inputs as counters on the Unipi Iris module. For more details, visit the manufacturer's website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
R1..R16	Reset counter	Bool

Parameter

slot	Slot	Long (I32)
0	Not used	
12	Slot #12	
22	Slot #22	
32	Slot #32	
42	Slot #42	
52	Slot #52	
11	Slot #11	
21	Slot #21	
debounce1..debounce16	Debounce time in 100 μ s	\downarrow 0 \uparrow 65535 \odot 1 Long (I32)
reset_value1..reset_value16	Counter reset value	\uparrow 4294967295 Word (U16)

Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)

y1..y16	Counter value	↑4294967295	Word (U16)
---------	---------------	-------------	------------

IM204DI – Iris IM204DI, 16 digital inputs

Block Symbol

Licence: [UNIFI](#)



Function Description

The IM204DI block enables reading digital inputs on the Unipi Iris module. For more details, visit the manufacturer’s website [1].

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

HLD	Hold	Bool
-----	------	------

Parameter

slot	Slot	Long (I32)
0	Not used	
12	Slot #12	
22	Slot #22	
32	Slot #32	
42	Slot #42	
52	Slot #52	
11	Slot #11	
21	Slot #21	
debounce1..debounce16	Debounce time in 100 μs	↓0 ↑65535 ⊙1 Long (I32)

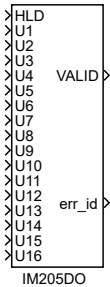
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
Y1..Y16	Digital input	Bool

IM205D0 – Iris IM205DO, 16 digital outputs

Block Symbol

Licence: [UNIFI](#)



Function Description

The IM205D0 block enables the use of digital outputs on the Unipi Iris module. For more details, visit the manufacturer’s website [1].

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

HLD	Hold	Bool
U1..U16	Digital output	Bool

Parameter

slot	Slot	Long (I32)
0	Not used	
12	Slot#12	
22	Slot#22	
32	Slot#32	
42	Slot#42	
52	Slot#52	
11	Slot#11	
21	Slot#21	

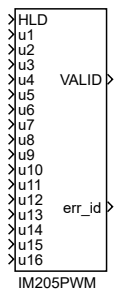
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)

IM205PWM – Iris IM205PWM, 16 PWM outputs

Block Symbol

Licence: [UNIFI](#)



Function Description

The IM205PWM block is used to control the pulse-width modulation (PWM) outputs on the Unipi Iris module. For more details, visit the manufacturer’s website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
u1..u16	PWM output	Word (U16)

Parameter

slot	Slot	Long (I32)
	0 Not used	
	12 Slot #12	
	22 Slot #22	
	32 Slot #32	
	42 Slot #42	
	52 Slot #52	
	11 Slot #11	
	21 Slot #21	
freq	Frequency	Long (I32)
	0 1kHz	
	1 100Hz	
	2 Custom prescaler	
prescaler	Prescaler	↓0 ↑65535 Long (I32)

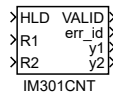
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)

IM301CNT – Iris IM301CNT, 2 digital counters

Block Symbol

Licence: [UNIPi](#)



Function Description

The IM301CNT block enables the use of digital inputs as counters on the Unipi Iris module. For more details, visit the manufacturer's website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
R1	Reset counter	Bool
R2	Reset counter	Bool

Parameter

slot	Slot	Long (I32)
0	Not used	
12	Slot #12	
22	Slot #22	
32	Slot #32	
42	Slot #42	
52	Slot #52	
11	Slot #11	
21	Slot #21	
debounce1	Debounce time in 100 μ s	$\downarrow 0 \uparrow 65535 \odot 1$ Long (I32)
debounce2	Debounce time in 100 μ s	$\downarrow 0 \uparrow 65535 \odot 1$ Long (I32)
reset_value1	Counter reset value	$\uparrow 4294967295$ Word (U16)
reset_value2	Counter reset value	$\uparrow 4294967295$ Word (U16)

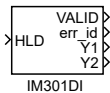
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
y1	Counter value	$\uparrow 4294967295$ Word (U16)
y2	Counter value	$\uparrow 4294967295$ Word (U16)

IM301DI – Iris IM301DI, 2 digital inputs

Block Symbol

Licence: [UNIPi](#)



Function Description

The IM301DI block enables reading digital inputs on the Unipi Iris module. For more details, visit the manufacturer’s website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
-----	------	------

Parameter

slot	Slot	Long (I32)
	0 Not used	
	12 Slot #12	
	22 Slot #22	
	32 Slot #32	
	42 Slot #42	
	52 Slot #52	
	11 Slot #11	
	21 Slot #21	
debounce1	Debounce time in 100 μ s	$\downarrow 0 \uparrow 65535 \odot 1$ Long (I32)
debounce2	Debounce time in 100 μ s	$\downarrow 0 \uparrow 65535 \odot 1$ Long (I32)

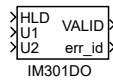
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
Y1	Digital input	Bool
Y2	Digital input	Bool

IM301D0 – Iris IM301DO, 2 digital outputs

Block Symbol

Licence: [UNIFI](#)



Function Description

The IM301D0 block enables the use of digital outputs on the Unipi Iris module. For more details, visit the manufacturer's website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
U1	Digital output	Bool
U2	Digital output	Bool

Parameter

slot	Slot	Long (I32)
0	Not used	
12	Slot #12	
22	Slot #22	
32	Slot #32	
42	Slot #42	
52	Slot #52	
11	Slot #11	
21	Slot #21	

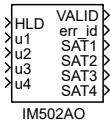
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)

IM502A0 – Iris IM502A0, 4 analog outputs

Block Symbol

Licence: [UNIFI](#)



Function Description

The IM502A0 block enables the use of analog outputs on the Unifi Iris module. For more details, visit the manufacturer’s website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
u1..u4	Analog output	Double (F64)

Parameter

slot	Slot	Long (I32)
0	Not used	
12	Slot#12	
22	Slot#22	
32	Slot#32	
42	Slot#42	
52	Slot#52	
11	Slot#11	
21	Slot#21	

Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
SAT1..SAT4	Saturation of analog output	Bool

IM503AI – Iris IM503AI, 8 analog inputs

Block Symbol

Licence: [UNIPi](#)



Function Description

The **IM503AI** block enables the use of analog inputs on the Unipi Iris module. For more details, visit the manufacturer's website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
-----	------	------

Parameter

slot	Slot	Long (I32)
	0 Not used	
	12 Slot #12	
	22 Slot #22	
	32 Slot #32	
	42 Slot #42	
	52 Slot #52	
	11 Slot #11	
	21 Slot #21	
mode1..mode8	Mode	⊙1 Long (I32)
	0 Not used	
	1 Voltage	
	2 Current	
average_mode	Average mode	⊙1 Long (I32)
	0 User	
	1 50-60Hz	
	2 50Hz	
	3 60Hz	
average_window	Average window	↓1 Word (U16)
average_count	Average count	↓1 ↑256 Word (U16)

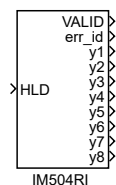
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
y1..y8	Analog input	Double (F64)

IM504RI – Iris IM504RI, 8 resistance or temperature inputs

Block Symbol

Licence: [UNIFI](#)



Function Description

The **IM504RI** block enables reading of resistance or temperature inputs on the Unipi Iris module. For more details, visit the manufacturer's website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
-----	------	------

Parameter

slot	Slot	Long (I32)
	0 Not used	
	12 Slot #12	
	22 Slot #22	
	32 Slot #32	
	42 Slot #42	
	52 Slot #52	
	11 Slot #11	
	21 Slot #21	
mode1..mode8	Mode	Long (I32)
	0 Not used	
	1 PT100	
	2 PT1000	
	3 Resistance	
average1..average8	Average	Long (I32)
	0 OFF	
	1 Fast	
	2 Slow	
filter	Error id	Long (I32)
	0 60HZ	
	1 50HZ	

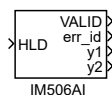
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
y1..y8	Measured value	Double (F64)

IM506AI – Iris IM506AI, 2 analog inputs

Block Symbol

Licence: [UNIPi](#)



Function Description

The IM506AI block enables the use of analog inputs on the Unipi Iris module. For more details, visit the manufacturer's website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
-----	------	------

Parameter

slot	Slot	Long (I32)
	0 Not used	
	12 Slot #12	
	22 Slot #22	
	32 Slot #32	
	42 Slot #42	
	52 Slot #52	
	11 Slot #11	
	21 Slot #21	
mode1	Mode	Long (I32)
	0 Not used	
	1 Voltage	
	2 Current	
mode2	Mode	Long (I32)
	0 Not used	
	1 Voltage	
	2 Current	
average_mode	Average mode	⊙1 Long (I32)
	0 User	
	1 50-60Hz	
	2 50Hz	
	3 60Hz	
average_window	Average window	↓1 Word (U16)

<code>average_count</code>	Average count	↓1 ↑256	Word (U16)
----------------------------	---------------	---------	------------

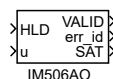
Output

<code>VALID</code>	Data validity indicator	Bool
<code>err_id</code>	Error ID	Long (I32)
<code>y1</code>	Analog input	Double (F64)
<code>y2</code>	Analog input	Double (F64)

IM506A0 – Iris IM506A0, 1 analog output

Block Symbol

Licence: [UNIFI](#)



Function Description

The IM506A0 block enables the use of analog outputs on the Unipi Iris module. For more details, visit the manufacturer's website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
u	Analog output	Double (F64)

Parameter

slot	Slot	Long (I32)
0	Not used	
12	Slot #12	
22	Slot #22	
32	Slot #32	
42	Slot #42	
52	Slot #52	
11	Slot #11	
21	Slot #21	

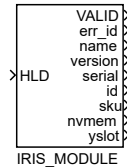
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
SAT	Saturation of analog output	Bool

IRIS_MODULE – Iris - Module description info

Block Symbol

Licence: [UNIPi](#)



Function Description

The IRIS_MODULE block retrieves information about the Unipi Iris module and displays it on its outputs. For more information, visit the manufacturer's website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
-----	------	------

Parameter

slot	Slot	Long (I32)
	0 Not used	
	12 Slot #12	
	22 Slot #22	
	32 Slot #32	
	42 Slot #42	
	52 Slot #52	
	11 Slot #11	
	21 Slot #21	

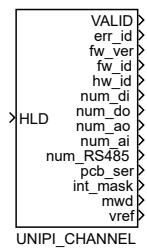
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
name	Module name	String
version	Module version	String
serial	Module serial	String
id	Module ID	String
sku	SKU	String
nvmem	Nvmem	String
yslot	Slot	String

UNUPI_CHANNEL – Iris module or Patron section info

Block Symbol

Licence: [UNUPI](#)



Function Description

The UNUPI_CHANNEL block block retrieves information about the Unipi Iris module or Unipi Patron section and displays it on its outputs. For more information, visit the manufacturer’s website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
-----	------	------

Parameter

slot	Slot	Long (I32)
0 Not used	
1 Section#1	
2 Section#2	
3 Section#3	
12 Slot #12	
22 Slot #22	
32 Slot #32	
42 Slot #42	
52 Slot #52	
11 Slot #11	
21 Slot #21	

Output

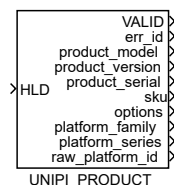
VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
fw_ver	Firmware Version	Word (U16)

<code>fw_id</code>	Firmware ID	Word (U16)
<code>hw_id</code>	Hardware ID	Word (U16)
<code>num_di</code>	Number of DIs	Word (U16)
<code>num_do</code>	Number of DOs	Word (U16)
<code>num_ao</code>	Number of AOs	Word (U16)
<code>num_ai</code>	Number of AIs	Word (U16)
<code>num_RS485</code>	Number of internal RS485 lines	Word (U16)
<code>pcb_ser</code>	PCB Serial Number	DWord (U32)
<code>int_mask</code>	Interrupt Mask	Word (U16)
<code>mwd</code>	Section MasterWatchDog (MWD) Timeout (in 1ms)	Word (U16)
<code>vref</code>	VRef of MCU	Word (U16)

UNIFI_PRODUCT – Product description info

Block Symbol

Licence: [UNIFI](#)



Function Description

The `UNIFI_PRODUCT` block retrieves information about the Unipi product and displays it on its outputs. For more information, visit the manufacturer's website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

<code>HLD</code>	Hold	<code>Bool</code>
------------------	------	-------------------

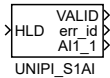
Output

<code>VALID</code>	Data validity indicator	<code>Bool</code>
<code>err_id</code>	Error ID	<code>Long (I32)</code>
<code>product_model</code>	Product model	<code>String</code>
<code>product_version</code>	Product version	<code>String</code>
<code>product_serial</code>	Product serial	<code>String</code>
<code>sku</code>	SKU	<code>String</code>
<code>options</code>	Options	<code>String</code>
<code>platform_family</code>	Platform family	<code>String</code>
<code>platform_series</code>	Platform series	<code>String</code>
<code>raw_platform_id</code>	RAW platform ID	<code>String</code>

UNIPi_S1AI – Patron section 1, analog input

Block Symbol

Licence: [UNIPi](#)



Function Description

The `UniPi_S1AI` block allows reading the value of the analog input located in Section 1 of the Unipi Patron device. This section is present in all Unipi Patron models. More information can be found on the manufacturer’s website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
-----	------	------

Parameter

mode	Mode voltage/current	⊙1 Long (I32)
	0 Voltage 0 - 10V	
	1 Current 0 - 20mA	

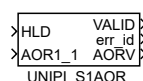
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
AI1_1	Analog input	Double (F64)

UNIPi_S1AOR – Patron section 1, analog output or resistance input

Block Symbol

Licence: [UNIPi](#)



Function Description

The **UNIPi_S1AOR** block allows controlling the analog output located in section 1 of the Unipi Patron device. This section is present in all Unipi Patron models. Depending on the value of the **mode** parameter, it is possible to generate voltage, current, or measure resistance. More information can be found on the manufacturer's website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
AOR1_1	Analog output	Bool

Parameter

mode	Mode	⊙1	Long (I32)
0 Voltage output 0 - 10V		
1 Current output 0 - 20mA		
3 Resistance measurement 0 - 2 kΩ		

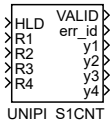
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
AORV	Measured resistance	Double (F64)

UNIFI_S1CNT – Patron section 1, counters

Block Symbol

Licence: UNIFI



Function Description

The UNIFI_S1CNT block allows the use of digital inputs located in section 1 of the Unipi Patron device in the counter input mode. The section 1 is present in all Unipi Patron models. More information can be found on the manufacturer’s website [1].

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

HLD	Hold	Bool
R1..R4	Reset counter	Bool

Parameter

debounce1..debounce4	Debounce time in 100 μ s	\downarrow 0 \uparrow 65535 \ominus -1	Long (I32)
reset_value1..reset_value4	Counter reset value	\uparrow 4294967295	Word (U16)

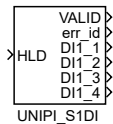
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
y1..y4	Counter value	\uparrow 4294967295 Word (U16)

UNIPi_S1DI – Patron section 1, digital inputs

Block Symbol

Licence: [UNIPi](#)



Function Description

The UNIPi_S1DI block allows reading digital inputs located in section 1 of the Unipi Patron device. This section is present in all Unipi Patron models. More information can be found on the manufacturer’s website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
-----	------	------

Parameter

debounce1..debounce4	Debounce time in 100 μ s	\downarrow 0 \uparrow 65535 \odot -1	Long (I32)
----------------------	------------------------------	--	------------

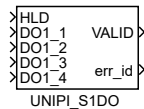
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
DI1_1	Digital input	Bool
DI1_2	Digital input	Bool
DI1_3	Digital input	Bool
DI1_4	Digital input	Bool

UNIFI_S1D0 – Patron section 1, digital outputs

Block Symbol

Licence: [UNIFI](#)



Function Description

The `UNIFI_S1D0` block allows controlling digital outputs located in section 1 of the Unipi Patron device. This section is present in all Unipi Patron models. More information can be found on the manufacturer’s website [1].

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

HLD	Hold	Bool
D01_1	Digital output	Bool
D01_2	Digital output	Bool
D01_3	Digital output	Bool
D01_4	Digital output	Bool

Parameter

enable_DS1..enable_DS4	Enable direct switch	on	Bool
invert_DS1..invert_DS4	Invert direct switch	on	Bool
toggle_DS1..toggle_DS4	Toggle direct switch	on	Bool

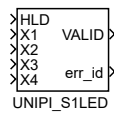
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)

UNIPi_S1LED – Patron section 1, LED outputs

Block Symbol

Licence: [UNIPi](#)



Function Description

The UNIPi_S1LED block is used to control the LEDs located in section 1 of the Unipi Patron device. This section is present in all Unipi Patron models. More information can be found on the manufacturer’s website [1].

This block does not propagate the signal quality. More information can be found in the 1.4 section.

Input

HLD	Hold	Bool
X1..X4	LED indicator	Bool

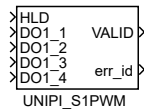
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)

UNIPi_S1PWM – Patron section 1, PWM outputs

Block Symbol

Licence: [UNIPi](#)



Function Description

The `UNIPi_S1PWM` block is used to control the digital outputs located in section 1 of the Unipi Patron device in the pulse width modulation (PWM) mode. The section 1 is present in all Unipi Patron models. More information can be found on the manufacturer's website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
D01_1	PWM output	Word (U16)
D01_2	PWM output	Word (U16)
D01_3	PWM output	Word (U16)
D01_4	PWM output	Word (U16)

Parameter

prescaler	Prescaler	↓0 ↑65535	Long (I32)
cycle_length	Cycle length	↓0 ↑65535	Long (I32)

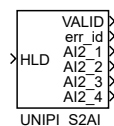
Output

err_id	Error ID	Long (I32)
VALID	Data validity indicator	Bool

UNIPi_S2AI – Patron section 2, analog inputs

Block Symbol

Licence: [UNIPi](#)



Function Description

The UNIPi_S2AI block is used to read the analog inputs located in section 2 of the Unipi Patron device. The block can be used only with specific Patron models that have more than one section. More information can be found on the manufacturer's website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
-----	------	------

Parameter

mode1..mode4	Mode voltage/current	⊙1	Long (I32)
0 Voltage 0 - 10V		
1 Current 0 - 20mA		

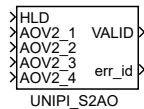
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
AI2_1	Analog input	Double (F64)
AI2_2	Analog input	Double (F64)
AI2_3	Analog input	Double (F64)
AI2_4	Analog input	Double (F64)

UNIFI_S2A0 – Patron section 2, analog outputs

Block Symbol

Licence: [UNIFI](#)



Function Description

The `UNIFI_S2A0` block is used to control the analog output located in section 2 of the Unifi Patron device. The block can be used only with specific Patron models that have more than one section. More information can be found on the manufacturer’s website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
AOV2_1	Analog output	Double (F64)
AOV2_2	Analog output	Double (F64)
AOV2_3	Analog output	Double (F64)
AOV2_4	Analog output	Double (F64)

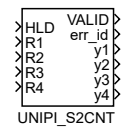
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)

UNIPi_S2CNT – Patron section 2, counters

Block Symbol

Licence: [UNIPi](#)



Function Description

The UNIPi_S2CNT block is used to read the counter inputs located in section 2 of the Unipi Patron device in the counter input mode. The block can be used only with specific Patron models that have more than one section. More information can be found on the manufacturer’s website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
R1..R4	Reset counter	Bool

Parameter

debounce1..debounce4	Debounce time in 100 μ s	\downarrow 0 \uparrow 65535 \odot -1	Long (I32)
reset_value1..reset_value4	Counter reset value	\uparrow 4294967295 \odot 1	Word (U16)

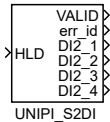
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
y1..y4	Counter value	\uparrow 4294967295 Word (U16)

UNIFI_S2DI – Patron section 2, digital inputs

Block Symbol

Licence: [UNIFI](#)



Function Description

The UNIFI_S2DI block is used to read the digital inputs located in section 2 of the Unipi Patron device. The block can be used only with specific Patron models that have more than one section. More information can be found on the manufacturer’s website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
-----	------	------

Parameter

debounce1..debounce4	Debounce time in 100 μ s	$\downarrow 0 \uparrow 65535 \odot -1$	Long (I32)
----------------------	------------------------------	--	------------

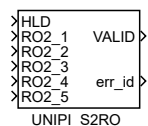
Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)
DI2_1	Digital input	Bool
DI2_2	Digital input	Bool
DI2_3	Digital input	Bool
DI2_4	Digital input	Bool

UNIPi_S2R0 – Patron section 2, relay outputs

Block Symbol

Licence: [UNIPi](#)



Function Description

The **UNIPi_S2R0** block is used to control the relay outputs located in section 2 of the Unipi Patron device. The block can be used only with specific Patron models that have more than one section. More information can be found on the manufacturer's website [\[1\]](#).

This block does not propagate the signal quality. More information can be found in the [1.4](#) section.

Input

HLD	Hold	Bool
R02_1	Digital output	Bool
R02_2	Digital output	Bool
R02_3	Digital output	Bool
R02_4	Digital output	Bool
R02_5	Digital output	Bool

Parameter

enable_DS1..enable_DS4	Enable direct switch	☉on	Bool
invert_DS1..invert_DS4	Invert direct switch	☉on	Bool
toggle_DS1..toggle_DS4	Toggle direct switch	☉on	Bool

Output

VALID	Data validity indicator	Bool
err_id	Error ID	Long (I32)

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.

	RexCore DEMO	RexCore Starter	RexCore Plus	RexCore Professional	RexCore Ultimate
<i>Function blocks</i>					
STANDARD	•	•	•	•	•
ADVANCED	•	–	•	•	•
REXLANG	•	–	•	•	•
MOTION CONTROL	•	–	○	○	•
COORDINATED MOTION	•	–	○	○	•
AUTOTUNING	–	–	○	○	•
MATRIX	•	–	○	○	•
<i>I/O drivers</i>					
Basic I/O drivers	•	•	•	•	•
Additional I/O drivers	•	○	○	•	•

(• ... 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 REXYGEN 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.

Function block name	Licensing group	
	STANDARD	Other
ABS_	•	
ABSR0T		ADVANCED
ACD	•	
ADD	•	
ADDHEXD	•	
ADDOCT	•	
ADDQUAD	•	
AFLUSH	•	
ALB	•	
ALBI	•	
ALN	•	
ALNI	•	
ARS	•	
AND_	•	
ANDHEXD	•	
ANDOCT	•	
ANDQUAD	•	
ANLS	•	
ARC	•	
ARLY	•	

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
ASW		ADVANCED
ATMT	•	
AVG	•	
AVS		ADVANCED
BDHEXD	•	
BDOCT	•	
BINS	•	
BIS	•	
BISR	•	
BITOP	•	
BMHEXD	•	
BMOCT	•	
BPF	•	
CanItem		CANDRV
CanRecv		CANDRV
CanSend		CANDRV
CDELSSM		ADVANCED
CMP	•	
CNA	•	
CNB	•	
CNDR	•	
CNE	•	
CNI	•	
CNR	•	
CNS	•	
CONCAT	•	
COND		
COUNT	•	
CSSM		ADVANCED
DATE_	•	
DATETIME	•	
DDELSSM		ADVANCED
DEL	•	
DELM	•	
DER	•	
DFIR		ADVANCED
DIF_	•	
Display	•	
DIV	•	

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
DSSM		ADVANCED
EAS	•	
EATMT		ADVANCED
EDGE_	•	
EKF		MODEL
EMD	•	
EPC		ADVANCED
EQ	•	
EVAR	•	
EXEC	•	
FIND	•	
FLCU		ADVANCED
FNX	•	
FNXY	•	
FOPDT	•	
FRID		ADVANCED
From	•	
GAIN	•	
GETPA	•	
GETPB	•	
GETPI	•	
GETPR	•	
GETPS	•	
Goto	•	
GotoTagVisibility	•	
GRADS		ADVANCED
HMI	•	
HTTP		ADVANCED
HTTP2		ADVANCED
I3PM		ADVANCED
IADD	•	
IDIV	•	
IMOD	•	
IMUL	•	
INFO	•	
INHEXD	•	
INOCT	•	
Inport	•	
INQUAD	•	

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
INSTD	•	
INTE	•	
INTSM	•	
IODRV	•	
IOTASK	•	
ISSW	•	
ISUB	•	
ITOI	•	
ITOS	•	
KDER		ADVANCED
LC	•	
LEN	•	
LIN	•	
LLC	•	
LPBRK	•	
LPF	•	
MC_AccelerationProfile		MOTION CONTROL
MC_AddAxisToGroup		COORDINATED MOTION
MC_CamIn		MOTION CONTROL
MC_CamOut		MOTION CONTROL
MC_CombineAxes		MOTION CONTROL
MC_GearIn		MOTION CONTROL
MC_GearInPos		MOTION CONTROL
MC_GearOut		MOTION CONTROL
MC_GroupContinue		COORDINATED MOTION
MC_GroupDisable		COORDINATED MOTION
MC_GroupEnable		COORDINATED MOTION
MC_GroupHalt		COORDINATED MOTION
MC_GroupInterrupt		COORDINATED MOTION
MC_GroupReadActualAcceleration		COORDINATED MOTION
MC_GroupReadActualPosition		COORDINATED MOTION
MC_GroupReadActualVelocity		COORDINATED MOTION
MC_GroupReadError		COORDINATED MOTION
MC_GroupReadStatus		COORDINATED MOTION
MC_GroupReset		COORDINATED MOTION
MC_GroupSetOverride		COORDINATED MOTION
MC_GroupSetPosition		COORDINATED MOTION
MC_GroupStop		COORDINATED MOTION
MC_Halt		MOTION CONTROL

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
MC_HaltSuperimposed		MOTION CONTROL
MC_Home		MOTION CONTROL
MC_MoveAbsolute		MOTION CONTROL
MC_MoveAdditive		MOTION CONTROL
MC_MoveCircularAbsolute		COORDINATED MOTION
MC_MoveCircularRelative		COORDINATED MOTION
MC_MoveContinuousAbsolute		MOTION CONTROL
MC_MoveContinuousRelative		MOTION CONTROL
MC_MoveDirectAbsolute		COORDINATED MOTION
MC_MoveDirectRelative		COORDINATED MOTION
MC_MoveLinearAbsolute		COORDINATED MOTION
MC_MoveLinearRelative		COORDINATED MOTION
MC_MovePath		COORDINATED MOTION
MC_MovePath_PH		COORDINATED MOTION
MC_MoveRelative		MOTION CONTROL
MC_MoveSuperimposed		MOTION CONTROL
MC_MoveVelocity		MOTION CONTROL
MC_PhasingAbsolute		MOTION CONTROL
MC_PhasingRelative		MOTION CONTROL
MC_PositionProfile		MOTION CONTROL
MC_Power		MOTION CONTROL
MC_ReadActualPosition		MOTION CONTROL
MC_ReadAxisError		MOTION CONTROL
MC_ReadBoolParameter		MOTION CONTROL
MC_ReadCartesianTransform		COORDINATED MOTION
MC_ReadParameter		MOTION CONTROL
MC_ReadStatus		MOTION CONTROL
MC_Reset		MOTION CONTROL
MC_SetCartesianTransform		COORDINATED MOTION
MC_SetOverride		MOTION CONTROL
MC_Stop		MOTION CONTROL
MC_TorqueControl		MOTION CONTROL
MC_UngroupAllAxes		COORDINATED MOTION
MC_VelocityProfile		MOTION CONTROL
MC_WriteBoolParameter		MOTION CONTROL
MC_WriteParameter		MOTION CONTROL
MCP_AccelerationProfile		MOTION CONTROL
MCP_CamIn		MOTION CONTROL
MCP_CamTableSelect		MOTION CONTROL

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
MCP_CombineAxes		MOTION CONTROL
MCP_GearIn		MOTION CONTROL
MCP_GearInPos		MOTION CONTROL
MCP_GroupHalt		COORDINATED MOTION
MCP_GroupInterrupt		COORDINATED MOTION
MCP_GroupSetOverride		COORDINATED MOTION
MCP_GroupSetPosition		COORDINATED MOTION
MCP_GroupStop		COORDINATED MOTION
MCP_Halt		MOTION CONTROL
MCP_HaltSuperimposed		MOTION CONTROL
MCP_Home		MOTION CONTROL
MCP_MoveAbsolute		MOTION CONTROL
MCP_MoveAdditive		MOTION CONTROL
MCP_MoveCircularAbsolute		COORDINATED MOTION
MCP_MoveCircularRelative		COORDINATED MOTION
MCP_MoveContinuousAbsolute		MOTION CONTROL
MCP_MoveContinuousRelative		MOTION CONTROL
MCP_MoveDirectAbsolute		COORDINATED MOTION
MCP_MoveDirectRelative		COORDINATED MOTION
MCP_MoveLinearAbsolute		COORDINATED MOTION
MCP_MoveLinearRelative		COORDINATED MOTION
MCP_MovePath		COORDINATED MOTION
MCP_MovePath_PH		COORDINATED MOTION
MCP_MoveRelative		MOTION CONTROL
MCP_MoveSuperimposed		MOTION CONTROL
MCP_MoveVelocity		MOTION CONTROL
MCP_PhasingAbsolute		MOTION CONTROL
MCP_PhasingRelative		MOTION CONTROL
MCP_PositionProfile		MOTION CONTROL
MCP_SetCartesianTransform		COORDINATED MOTION
MCP_SetKinTransform_Arm		COORDINATED MOTION
MCP_SetKinTransform_Schunk		COORDINATED MOTION
MCP_SetKinTransform_UR		COORDINATED MOTION
MCP_SetOverride		MOTION CONTROL
MCP_Stop		MOTION CONTROL
MCP_TorqueControl		MOTION CONTROL
MCP_VelocityProfile		MOTION CONTROL
MCU	•	
MDL	•	

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
MDLI	•	
MID	•	
MINMAX	•	
MODULE	•	
MP	•	
MqttPublish		MQTTDRV
MqttSubscribe		MQTTDRV
MUL	•	
MVD	•	
NOT_	•	
NSCL	•	
NSSM		MODEL
OpcUaReadValue		ADVANCED
OpcUaServerValue		ADVANCED
OpcUaWriteValue		ADVANCED
OR_	•	
ORHEXD	•	
OROCT	•	
ORQUAD	•	
OSD	•	
OSCALL	•	
OUTHEXD	•	
OUTOCT	•	
Outport	•	
OUTQUAD	•	
OUTRHEXD		ADVANCED
OUTROCT		ADVANCED
OUTRQUAD		ADVANCED
OUTRSTD		ADVANCED
OUTSTD	•	
PARA	•	
PARB	•	
PARE	•	
PARI	•	
PARR	•	
PARS	•	
PGAVR		
PGBAT		
PGBUS		

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
PGCB		
PGENG		
PGGEN		
PGGS		
PGINV		
PGLOAD		
PGMAINS		
PGSENS		
PGSG		
PGSIM		
PGSOLAR		
PGWIND		
PIDAT		AUTOTUNING
PIDE		ADVANCED
PIDGS		ADVANCED
PIDMA		AUTOTUNING
PIDU	•	
PIDUI		ADVANCED
PJROCT	•	
PJSEXOCT	•	
PJSEXOCT	•	
PJSOCT	•	
POL	•	
POUT	•	
PRBS	•	
PRGM	•	
PROJECT	•	
PSMPC		ADVANCED
PWM	•	
PYTHON		REXLANG
QFC		ADVANCED
QFD		ADVANCED
QTASK	•	
QP_MPC2QP		ADVANCED
QP_UPDATE		ADVANCED
QP_OASES		ADVANCED
QCEDPOPT		ADVANCED
RDC		ADVANCED
REC	•	

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
REGEXP		ADVANCED
REL	•	
REPLACE	•	
REXLANG		REXLANG
RLIM	•	
RLY	•	
RM_AxesGroup		COORDINATED MOTION
RM_Axis		MOTION CONTROL
RM_AxisOut		MOTION CONTROL
RM_AxisSpline		MOTION CONTROL
RM_DirectTorque		MOTION CONTROL
RM_DirectVelocity		MOTION CONTROL
RM_DriveMode		MOTION CONTROL
RM_Feed		COORDINATED MOTION
RM_Gcode		COORDINATED MOTION
RM_GroupTrack		COORDINATED MOTION
RM_HomeOffset		MOTION CONTROL
RM_Track		MOTION CONTROL
RS	•	
RTOI	•	
RTOS	•	
RTOV	•	
S_AND		
S_BC		
S_CMP		
S_CTS		
S_LB		
S_NOT		
S_OR		
S_PULS		
S_PV		
S_RS		
S_SEL		
S_SELVAL		
S_SR		
S_SUMC		
S_TDE		
S_TDR		
S_TLATCH		

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
S_VALB		
S_VALC		
S1OF2		ADVANCED
SAI		ADVANCED
SAT	•	
SC2FA		AUTOTUNING
SCU	•	
SCUV	•	
SEL	•	
SELHEXD	•	
SELOCT	•	
SELQUAD	•	
SELSOCT	•	
SELU	•	
SETPA	•	
SETPB	•	
SETPI	•	
SETPR	•	
SETPS	•	
SG	•	
SGI	•	
SGSLP		ADVANCED
SHIFTOCT	•	
SHLD	•	
SILO	•	
SILOS	•	
SINT	•	
SLEEP	•	
SMHCC		ADVANCED
SMHCCA		AUTOTUNING
SMTF		ADVANCED
SOPDT	•	
SPIKE		ADVANCED
SQR	•	
SQRT_	•	
SR	•	
SRTF		ADVANCED
SSW	•	
STEAM	•	

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
STOR	•	
SUB	•	
SubSystem	•	
SWR	•	
SWU	•	
SWVMR	•	
TASK	•	
TIME	•	
TIMER_	•	
TIODRV	•	
TRND	•	
TRNDV	•	
TSE	•	
UTOI	•	
VDEL	•	
VIN		ADVANCED
VOUT		ADVANCED
VTOR	•	
WASM		REXLANG
WSCH	•	
WWW	•	
ZV4IS		ADVANCED
DFIR		ADVANCED
PGSIM		
PGMAINS		
PGBUS		
PGLOAD		
PGGEN		
PGCB		
PGSENS		
PGENG		
PGAVR		
PGSG		
PGINV		
PGSOLAR		
PGWIND		
PGBAT		
PGGS		
CanSend		CANDRV

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
CanRecv		CANDRV
CanItem		CANDRV
MqttPublish		MQTTDRV
MqttSubscribe		MQTTDRV
EKF		MODEL
NSSM		MODEL
RM_HomeOffset		MOTION CONTROL
PARE	•	
EQ	•	
PYTHON		REXLANG
WASM		REXLANG
RM_DriveMode		MOTION CONTROL
RM_DirectTorque		MOTION CONTROL
RM_DirectVelocity		MOTION CONTROL
COND		
TESTS		
S_CMPT		
S_RCK		
S_POR		
OpcUaReadValue		
OpcUaWriteValue		
OpcUaServerValue		
STEAM	•	
PJSEXOCT	•	
BISR	•	
DP2M		
MBAL		
MOFN		
TB1		
TB2		
TB3		
TB6		
VAC		
OSD	•	
CNT	•	
CNDT	•	
CONCAT_DT	•	
SPLIT_DT	•	
STR2DT	•	

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
DT2STR	•	
WEEK	•	
T2STR	•	
TZ2UTC	•	
UTC2TZ	•	
SYSLOG	•	
SYSEVENT	•	
ALM	•	
ALARMS	•	
TRIM	•	

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 count reached limit
- 131 Text value has been truncated
- 132 Unsufficient buffer for requested operation
- 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 Communicaton closed unexpectedly
- 404 Unknown command
- 405 Unexpected command
- 406 Communicaton closed unexpectedly, probably 'Too many clients'
- 407 Communication timeout
- 408 Target device not found
- 409 Link failed
- 410 REXYGEN configuration has been changed
- 411 REXYGEN executive is being terminated
- 412 REXYGEN 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 blok
- 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 summ 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 drive FAULT
- 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 - Servodrive failed or disabled
- 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

Appendix D

Special signals of the REXYGEN system

There is a list of Special signals which can be read within REXYGEN. For details on how to use it have a look at example *0001_Special_Signals*.

Parameter	Desc	Data Type	Possibilities
perf	Performance counter frequency	LARGE	EXEC
period	Level, task, or block period	DOUBLE	LEVEL, TASK, SEQ, IODRV, QTASK, BLOCK, ARCHIVE
nblocks	Task or sequence number of blocks	SHORT	TASK, SEQ, QTASK
stack	Task stack size	LONG	TASK, IODRV, QTASK
exfac	Task execution factor	DWORD	TASK, IODRV, QTASK
start	Task start tick	DWORD	TASK
stop	Task stop tick	DWORD	TASK
ntasks	Exec or level number of tasks	SHORT	LEVEL, IODRV, EXEC
ntick	Number of level ticks	DWORD	LEVEL
pri	Level priority	SHORT	LEVEL, IODRV, QTASK
tick	Executive timer tick	LARGE	EXEC
nlevels	Number of executive levels	SHORT	EXEC
nmodules	Number of executive modules	SHORT	EXEC
ndrivers	Number of executive drivers	SHORT	EXEC
narchives	Number of executive archives	SHORT	EXEC
nqtasks	Number of executive quick-tasks	SHORT	EXEC
tcomp	Time when executive was compiled [ns from epoch]	LARGE	EXEC
tdnld	Executive download time [ns from epoch]	LARGE	EXEC
bufsize	(Archive) buffer size	LONG	ARCHIVE
timesize	(Archive) index-buffer size	LONG	ARCHIVE
daysize	(Archive) day-buffer(file) size	LARGE	ARCHIVE
errblk	Index of block with exec error	SHORT	TASK, SEQ, QTASK
errno	Exec error code	SHORT	TASK, SEQ, QTASK, IODRV
status	(Driver) status code	LONG	IODRV
over	(Qtask) number of overlap/-colisions	LARGE	QTASK, TASK, IODRV
excnt	Count of task's starts	LARGE	LEVEL, TASK, SEQ, IODRV, QTASK
tlast	Number of last execution [performance-counter-ticks]	LARGE	LEVEL, TASK, SEQ, IODRV, QTASK
tmin	Minimum number of execution [performance-counter-ticks]	LARGE	LEVEL, TASK, SEQ, IODRV, QTASK
tmax	Maximum number of execution [performance-counter-ticks]	LARGE	LEVEL, TASK, SEQ, IODRV, QTASK
tsum	Execution sum [ns or ticks]	LARGE	LEVEL, TASK, SEQ, IODRV, QTASK
tavg	Execution ticks average (tsum/excnt)[performance-counter-ticks]	LARGE	LEVEL, TASK, SEQ, IODRV, QTASK
dstart	Start tick delay [performance-counter-ticks]	LARGE	IODRV, TASK
dstop	Stop tick delay [performance-counter-ticks]	LARGE	IODRV, TASK

Bibliography

- [1] Unipi Technology s.r.o. Unipi technology. <http://www.unipi.technology>, 2024. →.
- [2] OPC Foundation. *Data Access Custom Interface Specification Version 3.00*. OPC Foundation, P.O. Box 140524, Austin, Texas, USA, 2003.
- [3] REX Controls s.r.o.. *REXYGEN Studio – User manual*, 2020. →.
- [4] Schlegel Miloš. Fuzzy controller: a tutorial. *www.rexcontrols.com*.
- [5] M. Goubelj. Kalman filter based observer design for real-time frequency identification in motion control systems. <https://ieeexplore.ieee.org/document/7169979>, 2015.
- [6] Miloš Schlegel, Pavel Balda, and Milan Štětina. Robust PID autotuner: method of moments. *Automatizace*, 46(4):242–246, 2003.
- [7] M. Schlegel and P. Balda. Diskretizace spojitého lineárního systému (in Czech). *Automatizace*, 11, 1987.
- [8] J. Königsmarková and M. Schlegel. Identification of n-link inverted pendulum on a cart. <https://ieeexplore.ieee.org/document/7976186>, 2017.
- [9] BLAS 3.8.0. – netlib.org. Basic Linear Algebra Subprograms Version 3.8.0. <http://www.netlib.org/blas/>, 2017.
- [10] LAPACK 3.8.0 – netlib.org. Linear Algebra PACKage Version 3.8.0. <http://www.netlib.org/lapack/explore-html/index.html>, 2018.
- [11] Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, 2003.
- [12] J.M. Maciejowski. *Predictive Control with Constraints*. Prentice Hall, England., 2002.
- [13] H.J. Ferreau, C. Kirches, A. Potschka, H.G. Bock, and M. Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.
- [14] H.J. Ferreau. *qpOASES User’s Manual, Version 3.2 (April 2017)*, 2017.

- [15] Y. Ye. *Interior algorithms for linear quadratic and linearly constrained nonlinear programming*. Ph.d. dissertation, Dept. Elect. Eng. Syst., Stanford Univ., Stanford, CA, 1987.
- [16] Y. Ye. *User's Manual for Matlab Linear Programming Codes*, 1989. →.
- [17] LabLua PUC-Rio. Lua 5.4 Reference Manual. <https://www.lua.org/manual/5.4/>, 2020.
- [18] Python Software Foundation. Python documentation. <https://docs.python.org/3/>, 2019.
- [19] Martin Reinecke. PocketFFT. <https://gitlab.mpcdf.mpg.de/mtr/pocketfft>, 2019.
- [20] Paul N. Swarztrauber. FFTPACK Version 4 – netlib.org. <https://www.netlib.org/fftpack>, 1985.
- [21] Paul N. Swarztrauber. Vectorizing the ffts**this chapter was written while the author was visiting the scientific computing division at the national bureau of standards. In GARRY RODRIGUE, editor, *Parallel Computations*, pages 51–83. Academic Press, 1982.
- [22] P. Welch. The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. *IEEE Transactions on Audio and Electroacoustics*, 15(2):70–73, 1967.
- [23] Wikipedia. Welch's method. https://en.wikipedia.org/wiki/Welch's_method.
- [24] REX Controls s.r.o.. *MQTTDrv driver of REXYGEN – user guide*, 2020. →.
- [25] OASIS. MQTT Version 3.1.1. <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>, 2014.
- [26] REX Controls s.r.o.. *OpcUaDrv driver of REXYGEN – user guide*, 2020. →.

Index

REXYGEN Compiler
 compiler, 42
REXYGEN Compiler compiler, 34

ABS, 18, 26, 94, 95
ABS_, 931
ABSR0T, 26, 137, 931
ACD, 323, 325, 931
ACLEAR, 323, 327
ADD, 18, 26, 94, 96, 97, 931
ADDHEXD, 26, 97, 931
ADDOCT, 96, 97, 133, 931
ADDQUAD, 97, 931
AFLUSH, 323, 328, 931
ALARMS, 29, 329, 331, 332, 943
ALB, 29–31, 323, 329, 931
ALBI, 29–31, 323, 329, 931
ALM, 29–31, 323, 331, 943
ALMI, 29–31, 323, 331
ALN, 29–31, 323, 332, 931
ALNI, 29–31, 323, 332, 931
AND, 18, 26, 283–285
AND_, 931
ANDHEXD, 26, 284, 285, 931
ANDOCT, 284, 285, 931
ANDQUAD, 284, 285, 931
ANLS, 18, 189, 190, 931
application
 of the REXYGEN System, 34
ARC, 32, 324, 328, 931
architecture
 open, 43
archiv, 323
archive
 backed-up memory, 324
 configuration, 32

 disk, 324
 RAM memory, 323
ARLY, 26, 205, 931
ARS, 323, 335, 931
ASW, 18, 26, 136, 139, 932
ATMT, 18, 26, 283, 286, 293, 368, 383, 389,
 932
AVG, 18, 26, 136, 141, 932
AVS, 26, 142, 932
AVSI, 26, 143

band
 frequency transmission, 146
bandwidth, 159
BDHEXD, 26, 288, 293, 932
BDOCT, 18, 283, 288, 293, 932
Bessel filter, 159
BINS, 18, 189, 192, 194, 196, 932
BIS, 18, 189, 192, 194, 198, 932
BISR, 18, 189, 196, 932, 942
BITOP, 18, 26, 134, 283, 289, 932
block
 description, 23
 description format, 23
 execution, 48
 inputs, 23
 outputs, 23
 parameters, 23
 symbol, 23
BMHEXD, 26, 290, 293, 932
BMOCT, 18, 283, 290, 293, 932
BPF, 18, 26, 136, 146, 932
BSFIFO, 20, 613, 614
BSGET, 20, 613, 616, 618, 620
BSGETOCT, 616, 620
BSGETOCTV, 618, 621

- BSGETV, 20, 613, 618, 621
- BSSET, 20, 613, 616, 620
- BSSETOCT, 616, 620
- BSSETOCTV, 618, 621
- BSSETV, 20, 613, 618, 621
- Butterworth filter, 159
- CanItem, 22, 875, 876, 878, 932, 942
- CanRecv, 22, 875, 876, 878, 932, 942
- CanSend, 22, 875, 880, 932, 941
- CDELSSM, 19, 26, 398, 399, 932
- CMP, 26, 147, 932
- CNA, 26, 367, 444, 932
- CNB, 26, 98, 932
- CNDR, 26, 148, 932
- CNDT, 942
- CNE, 99, 932
- CNI, 26, 100, 932
- CNR, 26, 101, 932
- CNS, 19, 26, 343, 344, 932
- CNT, 942
- coefficient
 - relative damping, 146, 159
- compatibility
 - REXYGEN and Simulink, 42
- compiler
 - REXYGEN Compiler, 34, 42
- CONCAT, 19, 26, 343, 345, 932
- CONCAT_DT, 942
- COND, 932, 942
- configuration
 - REXYGEN System, 34
 - archives, 34
 - computation task, 34
 - input-output drivers, 34
 - modules, 34
- COUNT, 18, 24, 26, 283, 291, 932
- CSSM, 19, 26, 398, 402, 932
- data types, 23
- DATE, 19, 309–311
- DATE_, 932
- DATETIME, 19, 309–311, 316, 932
- DDELSSM, 19, 26, 398, 405, 932
- DEL, 18, 25, 26, 136, 150, 932
- DELM, 26, 151, 932
- denominator, 105
- DER, 26, 152, 932
- DFIR, 19, 26, 398, 407, 932, 941
- DIF, 26, 102
- DIF_, 932
- Display, 65, 932
- DIV, 18, 25, 26, 94, 103, 124, 932
- DP2M, 942
- driver
 - REXYGEN System, 39
 - .rio file extension, 39
 - configuration data, 39
 - user manual, 41
- DSSM, 19, 26, 398, 408, 933
- DT2STR, 943
- EAS, 26, 104, 933
- EATMT, 26, 293, 933
- EDGE, 26, 176, 296
- EDGE_, 933
- EKF, 19, 398, 411, 425, 933, 942
- EMD, 26, 105, 933
- EPC, 20, 44, 561, 562, 933
- EQ, 26, 297, 933, 942
- error
 - fatal, 46
- EVAR, 26, 154, 933
- EXEC, 17, 28, 32, 34, 39, 41, 43, 46, 47, 57, 163, 314, 933
- FFT, 20, 613, 622
- filter
 - Bessel, 159
 - Butterworth, 159
 - nonlinear, 180
- filtering
 - digital, 46
- FIND, 19, 26, 343, 346, 933
- first order system, 434
- FIWR, 26, 206
- FLCU, 18, 26, 204, 209, 933
- FMUCS, 19, 398

- FMUINFO, [19](#), [398](#)
- FNX, [18](#), [26](#), [94](#), [106](#), [933](#)
- FNXY, [18](#), [26](#), [94](#), [109](#), [933](#)
- FOPDT, [434](#)
- FOPDT, [19](#), [26](#), [232](#), [398](#), [414](#), [933](#)
- frequency transmission band, [146](#)
- FRID, [26](#), [211](#), [933](#)
- From, [69](#), [73](#), [81](#), [933](#)
- FromFile, [67](#)
- FromWorkspace, [68](#)
- function
 - single variable, [106](#)
- GAIN, [26](#), [111](#), [933](#)
- generator
 - time function, [244](#)
- GETPA, [19](#), [26](#), [70](#), [84](#), [365](#), [366](#), [369](#), [382](#), [933](#)
- GETPB, [368](#), [370](#), [371](#), [933](#)
- GETPI, [368](#), [370](#), [371](#), [581](#), [582](#), [933](#)
- GETPR, [26](#), [70](#), [368](#), [370](#), [371](#), [389](#), [933](#)
- GETPS, [19](#), [26](#), [70](#), [365](#), [370](#), [371](#), [933](#)
- GETPX, [26](#), [369–371](#)
- Goto, [69](#), [73](#), [81](#), [878](#), [880](#), [933](#)
- GotoTagVisibility, [69](#), [81](#), [933](#)
- GRADS, [112](#), [933](#)
- HMI, [17](#), [28](#), [36](#), [933](#)
- HTTP, [565](#), [933](#)
- HTTP2, [20](#), [561](#), [565](#), [567](#), [933](#)
- I3PM, [215](#), [933](#)
- IADD, [18](#), [26](#), [94](#), [114](#), [933](#)
- IDIV, [18](#), [26](#), [94](#), [116](#), [933](#)
- IM201CNT, [891](#)
- IM201DI, [892](#)
- IM203DO, [893](#)
- IM203PWM, [894](#)
- IM204CNT, [895](#)
- IM204DI, [897](#)
- IM205DO, [898](#)
- IM205PWM, [899](#)
- IM301CNT, [901](#)
- IM301DI, [902](#)
- IM301DO, [903](#)
- IM502AO, [904](#)
- IM503AI, [905](#)
- IM504RI, [907](#)
- IM506AI, [909](#)
- IM506AO, [911](#)
- IMOD, [18](#), [26](#), [94](#), [117](#), [933](#)
- IMUL, [18](#), [26](#), [94](#), [118](#), [933](#)
- INCONN, [70](#), [367](#), [369](#), [370](#), [378](#)
- INFO, [38](#), [933](#)
- INHEXD, [71](#), [933](#)
- INOCT, [71](#), [75](#), [933](#)
- Inport, [83](#), [87](#), [369](#), [384](#), [736](#), [933](#)
- INQUAD, [71](#), [933](#)
- INSTD, [71](#), [73](#), [75](#), [77](#), [934](#)
- INTE, [26](#), [155](#), [179](#), [934](#)
- INTSM, [26](#), [134](#), [298](#), [934](#)
- IOASYNC, [26](#), [75](#)
- IODRV, [39](#), [73](#), [81](#), [934](#)
- IOTASK, [41](#), [48](#), [59](#), [366](#), [369](#), [381](#), [384](#), [399](#), [402](#), [934](#)
- IPEN2, [415](#), [418](#)
- IPEN2pu, [415](#), [418](#)
- IPEN3, [415](#), [418](#)
- IPEN3pu, [415](#), [418](#)
- IRIS_MODULE, [912](#)
- ISSW, [26](#), [299](#), [934](#)
- ISUB, [26](#), [120](#), [934](#)
- ITOI, [26](#), [300](#), [934](#)
- ITOS, [19](#), [26](#), [343](#), [347](#), [934](#)
- KDER, [26](#), [157](#), [934](#)
- LC, [26](#), [217](#), [934](#)
- least squares method, [152](#)
- LEN, [19](#), [26](#), [343](#), [348](#), [934](#)
- LIN, [18](#), [26](#), [94](#), [122](#), [934](#)
- LLC, [26](#), [218](#), [434](#), [934](#)
- LPBRK, [26](#), [42](#), [139](#), [934](#)
- LPF, [26](#), [159](#), [934](#)
- LPI, [26](#), [219](#)
- LSM, [152](#)
- LUA, [604](#)
- LUAHEXD, [604](#)
- LUAOCT, [604](#)

- LUAQUAD, 604
- MB_DASUM, 446
- MB_DAXPY, 448
- MB_DCOPY, 450
- MB_DDOT, 452
- MB_DGEMM, 20, 443, 454
- MB_DGEMV, 456
- MB_DGER, 20, 443, 458
- MB_DNRM2, 460, 499
- MB_DROT, 462
- MB_DSCAL, 464
- MB_DSWAP, 466
- MB_DTRMM, 20, 443, 468
- MB_DTRMV, 470
- MB_DTRSV, 472
- MBAL, 942
- MC_AccelerationProfile, 21, 640, 668, 700, 701, 720, 721, 934
- MC_AddAxisToGroup, 21, 785, 814, 934
- MC_CamIn, 21, 744, 748, 759, 768, 778, 781, 934
- MC_CamOut, 21, 744, 759, 763, 934
- MC_CombineAxes, 21, 743, 765, 934
- MC_GearIn, 21, 743, 768, 771, 776, 778, 781, 934
- MC_GearInPos, 21, 744, 771, 934
- MC_GearOut, 21, 743, 776, 934
- MC_GroupContinue, 815, 823, 934
- MC_GroupDisable, 21, 785, 816, 934
- MC_GroupEnable, 21, 785, 817, 934
- MC_GroupHalt, 21, 785, 818, 934
- MC_GroupInterrupt, 815, 823, 934
- MC_GroupReadActualAcceleration, 824, 934
- MC_GroupReadActualPosition, 22, 785, 825, 934
- MC_GroupReadActualVelocity, 22, 785, 826, 934
- MC_GroupReadError, 22, 785, 827, 934
- MC_GroupReadStatus, 22, 785, 828, 934
- MC_GroupReset, 829, 934
- MC_GroupSetOverride, 830, 934
- MC_GroupSetPosition, 831, 934
- MC_GroupStop, 833, 934
- MC_Halt, 21, 640, 672, 934
- MC_HaltSuperimposed, 673, 935
- MC_Home, 21, 640, 674, 705, 935
- MC_MoveAbsolute, 21, 640, 676, 683, 713, 740, 772, 935
- MC_MoveAdditive, 676, 680, 935
- MC_MoveCircularAbsolute, 21, 785, 836, 935
- MC_MoveCircularRelative, 21, 785, 840, 935
- MC_MoveContinuousAbsolute, 683, 935
- MC_MoveContinuousRelative, 686, 935
- MC_MoveDirectAbsolute, 21, 785, 844, 935
- MC_MoveDirectRelative, 21, 785, 847, 935
- MC_MoveLinearAbsolute, 21, 784, 850, 935
- MC_MoveLinearRelative, 21, 784, 854, 935
- MC_MovePath, 22, 785, 858, 935
- MC_MovePath_PH, 860, 935
- MC_MoveRelative, 21, 640, 676, 686, 690, 693, 935
- MC_MoveSuperimposed, 676, 693, 778, 781, 935
- MC_MoveVelocity, 21, 640, 696, 740, 935
- MC_PhasingAbsolute, 21, 743, 778, 935
- MC_PhasingRelative, 21, 743, 781, 935
- MC_PositionProfile, 21, 640, 668, 669, 700, 720, 721, 740, 747, 935
- MC_Power, 21, 640, 704, 935
- MC_ReadActualPosition, 21, 640, 705, 935
- MC_ReadAxisError, 21, 640, 706, 935
- MC_ReadBoolParameter, 707, 935
- MC_ReadCartesianTransform, 22, 785, 862, 935
- MC_ReadParameter, 21, 640, 708, 935
- MC_ReadStatus, 710, 935
- MC_Reset, 21, 640, 712, 829, 935
- MC_SetCartesianTransform, 22, 785, 863, 935
- MC_SetOverride, 713, 935
- MC_SetPosition, 674
- MC_Stop, 21, 640, 672, 715, 935
- MC_TorqueControl, 717, 935
- MC_UngroupAllAxes, 865, 935

- MC_VelocityProfile, 21, 640, 668, 669, 700, 701, 720, 935
- MC_WriteBoolParameter, 724, 935
- MC_WriteParameter, 21, 640, 725, 935
- MCP_AccelerationProfile, 641, 668, 935
- MCP_CamIn, 745, 759, 935
- MCP_CamTableSelect, 21, 744, 747, 759, 760, 935
- MCP_CombineAxes, 749, 765, 936
- MCP_GearIn, 751, 768, 936
- MCP_GearInPos, 753, 771, 936
- MCP_GroupHalt, 787, 936
- MCP_GroupInterrupt, 788, 823, 936
- MCP_GroupSetOverride, 789, 936
- MCP_GroupSetPosition, 790, 831, 936
- MCP_GroupStop, 791, 936
- MCP_Halt, 643, 672, 936
- MCP_HaltSuperimposed, 644, 673, 936
- MCP_Home, 645, 674, 936
- MCP_MoveAbsolute, 647, 676, 936
- MCP_MoveAdditive, 649, 680, 936
- MCP_MoveCircularAbsolute, 792, 936
- MCP_MoveCircularRelative, 794, 936
- MCP_MoveContinuousAbsolute, 651, 683, 936
- MCP_MoveContinuousRelative, 653, 686, 936
- MCP_MoveDirectAbsolute, 796, 936
- MCP_MoveDirectRelative, 798, 936
- MCP_MoveLinearAbsolute, 800, 936
- MCP_MoveLinearRelative, 802, 936
- MCP_MovePath, 804, 936
- MCP_MovePath_PH, 806, 936
- MCP_MoveRelative, 655, 690, 936
- MCP_MoveSuperimposed, 657, 693, 936
- MCP_MoveVelocity, 658, 696, 936
- MCP_PhasingAbsolute, 755, 778, 936
- MCP_PhasingRelative, 757, 781, 936
- MCP_PositionProfile, 660, 700, 936
- MCP_SetCartesianTransform, 808, 936
- MCP_SetKinTransform_Arm, 810, 936
- MCP_SetKinTransform_Schunk, 936
- MCP_SetKinTransform_UR, 812, 936
- MCP_SetOverride, 662, 713, 936
- MCP_Stop, 663, 715, 936
- MCP_TorqueControl, 664, 717, 936
- MCP_VelocityProfile, 666, 720, 936
- MCU, 26, 221, 281, 936
- MDL, 19, 26, 398, 421, 422, 936
- MDLI, 19, 26, 398, 422, 937
- MID, 19, 26, 343, 349, 937
- MINMAX, 26, 161, 937
- ML_DGEBAK, 474
- ML_DGEBAL, 476
- ML_DGEBRD, 20, 443, 478
- ML_DGECON, 480
- ML_DGEES, 20, 443, 483
- ML_DGEEV, 20, 443, 485
- ML_DGEHRD, 487
- ML_DGELQF, 489
- ML_DGELSD, 491
- ML_DGEQRF, 20, 443, 493
- ML_DGESDD, 20, 443, 495
- ML_DLACPY, 497
- ML_DLANGE, 460, 499
- ML_DLASET, 501
- ML_DTRSYL, 503
- model
 - FOPDT, 434
- MODULE, 39, 43, 937
- module, 43
 - extension, 39
- MOFN, 942
- MOSS, 20, 613, 624
- MP, 18, 189, 198, 329, 332, 937
- MqttPublish, 21, 633, 634, 937, 942
- MqttSubscribe, 21, 633, 636, 937, 942
- MUL, 18, 26, 94, 123, 937
- MVD, 19, 26, 398, 423, 937
- MX_AT, 505
- MX_ATSET, 506
- MX_CNADD, 507
- MX_CNMUL, 508
- MX_CTODPA, 509
- MX_DIM, 511
- MX_DIMSET, 512
- MX_DSAGET, 514
- MX_DSAREF, 516
- MX_DSASET, 518
- MX_DTRNSP, 20, 443, 520

- MX_DTRNSQ, 522
- MX_FILL, 20, 443, 524
- MX_MAT, 20, 367, 443, 525, 543, 553
- MX_RAND, 20, 443, 526
- MX_REFCOPY, 528
- MX_SLFS, 529
- MX_VEC, 20, 443, 532
- MX_WRITE, 533

- NANINF, 26, 124
- NOT, 18, 26, 283, 301
- NOT_, 937
- NSCL, 26, 162, 937
- NSSM, 411, 424, 937, 942
- NUREACT, 427

- OpcUaReadValue, 22, 881, 882, 937, 942
- OpcUaServerValue, 22, 881, 884, 937, 942
- OpcUaWriteValue, 22, 881, 886, 937, 942
- operation
 - binary, 128
- OR, 18, 26, 283, 302, 303
- OR_, 937
- order
 - driver execution, 39
 - driver initialization, 39
 - module initialization, 43
 - of task execution, 57
 - of task initialization, 57
- ORHEXD, 26, 302, 303, 937
- OROCT, 302, 303, 937
- ORQUAD, 302, 303, 937
- OSCALL, 44, 563, 937
- OSD, 26, 163, 937, 942
- OUTCONN, 76, 382, 384, 385
- OUTHEXD, 77, 79, 937
- OUTOCT, 75, 77, 79, 878, 880, 937
- Outport, 83, 87, 369, 384, 736, 937
- OUTQUAD, 77, 79, 878, 880, 937
- OUTRHEXD, 79, 937
- OUTROCT, 79, 937
- OUTRQUAD, 79, 937
- OUTRSTD, 80, 937
- OUTSTD, 71, 75, 77, 80, 81, 937

- overhead
 - control system core, 34

- PARA, 373, 937
- PARAM, 587
- parameter
 - tick, 34
 - remote, 381
 - remote acquirement, 366, 368
- PARB, 376, 378, 379, 937
- PARE, 375, 937, 942
- PARI, 375, 376, 378, 379, 937
- PARR, 19, 178, 365, 376, 378, 379, 937
- PARS, 378, 379, 937
- PARX, 376, 378, 379
- path
 - full, 48
- PCI, 626
- period
 - of quick task execution, 46
 - of task execution, 57
- PGAVR, 937, 941
- PGBAT, 937, 941
- PGBUS, 937, 941
- PGCB, 938, 941
- PGENG, 938, 941
- PGGEN, 938, 941
- PGGS, 938, 941
- PGINV, 938, 941
- PGLOAD, 938, 941
- PGMAINS, 938, 941
- PGSENS, 938, 941
- PGSG, 938, 941
- PGSIM, 938, 941
- PGSOLAR, 938, 941
- PGWIND, 938, 941
- PIDAT, 18, 26, 204, 223, 938
- PIDE, 26, 226, 938
- PIDGS, 18, 26, 204, 228, 938
- PIDMA, 18, 26, 204, 215, 230, 262, 570, 938
- PIDU, 26, 223, 226, 228, 237, 241, 262, 265, 281, 570, 726, 938
- PIDUI, 26, 241, 938
- PJROCT, 19, 343, 350, 938

- PJSEXOCT, [352](#), [938](#), [942](#)
- PJSOCT, [352](#), [354](#), [938](#)
- POL, [18](#), [26](#), [94](#), [126](#), [938](#)
- POUT, [26](#), [243](#), [938](#)
- PRBS, [18](#), [189](#), [199](#), [938](#)
- PRGM, [244](#), [938](#)
- priority
 - dependancy on the operating system, [34](#)
 - logic, [34](#)
 - logical, [39](#), [46](#)
 - of tasks, [57](#)
- program
 - REXYGEN Studio, [32](#), [39](#), [46](#), [48](#), [57](#), [59](#), [83](#), [366](#), [369](#), [381](#), [384](#)
 - REXYGEN Studio, Enable checkbox, [48](#)
 - REXYGEN Studio, Halt/Run button, [48](#)
 - REXYGEN Studio, Reset button, [49](#)
- PROJECT, [45](#), [938](#)
- project
 - main file, [34](#), [39](#)
- protocol
 - UDP/IP, [569](#)
- PSD, [20](#), [613](#), [627](#)
- PSMPC, [18](#), [26](#), [204](#), [215](#), [246](#), [938](#)
- puls, [243](#)
- pulse counting
 - bidirectional, [291](#)
- PWM, [26](#), [231](#), [238](#), [250](#), [270](#), [274](#), [275](#), [938](#)
- PYTHON, [600](#), [608](#), [616](#), [938](#), [942](#)
- QCEDPOPT, [20](#), [539](#), [938](#)
- QCOPT, [428](#)
- QFC, [85](#), [86](#), [580](#), [938](#)
- QFD, [79](#), [80](#), [85](#), [86](#), [580](#), [938](#)
- QP_MPC2QP, [20](#), [539](#), [540](#), [552](#), [938](#)
- QP_OASES, [20](#), [539](#), [540](#), [547](#), [938](#)
- QP_UPDATE, [20](#), [539](#), [540](#), [552](#), [938](#)
- QTASK, [17](#), [28](#), [34](#), [41](#), [46](#), [48](#), [57](#), [399](#), [402](#), [565](#), [567](#), [593](#), [938](#)
- rate monotonic scheduling, [35](#)
- RDC, [20](#), [561](#), [569](#), [938](#)
- RDFT, [26](#), [164](#)
- REC, [26](#), [127](#), [938](#)
- REGEXP, [19](#), [343](#), [356](#), [939](#)
- REL, [26](#), [128](#), [939](#)
- relative damping coefficient, [146](#), [159](#)
- REPLACE, [19](#), [26](#), [343](#), [358](#), [939](#)
- REXLANG, [20](#), [335](#), [412](#), [424](#), [425](#), [557](#), [561](#), [573](#), [600](#), [603](#), [604](#), [608](#), [616](#), [939](#)
- RLIM, [26](#), [166](#), [939](#)
- RLY, [26](#), [205](#), [252](#), [939](#)
- RM_AxesGroup, [814](#), [827](#), [866](#), [939](#)
- RM_Axis, [537](#), [674](#), [704–708](#), [715](#), [726](#), [733](#), [736](#), [814](#), [939](#)
- RM_AxisOut, [733](#), [939](#)
- RM_AxisSpline, [84](#), [143](#), [726](#), [734](#), [939](#)
- RM_DirectTorque, [939](#), [942](#)
- RM_DirectVelocity, [939](#), [942](#)
- RM_DriveMode, [939](#), [942](#)
- RM_Feed, [869](#), [939](#)
- RM_Gcode, [870](#), [939](#)
- RM_GroupTrack, [872](#), [939](#)
- RM_HomeOffset, [739](#), [939](#), [942](#)
- RM_Track, [740](#), [939](#)
- RS, [26](#), [304](#), [306](#), [939](#)
- RTOI, [26](#), [129](#), [939](#)
- RTOS, [19](#), [26](#), [343](#), [359](#), [939](#)
- RTOV, [26](#), [367](#), [535](#), [563](#), [939](#)
- S10F2, [25](#), [167](#), [940](#)
- S_AND, [939](#)
- S_BC, [939](#)
- S_CMP, [939](#)
- S_CMPT, [942](#)
- S_CTS, [939](#)
- S_LB, [939](#)
- S_NOT, [939](#)
- S_OR, [939](#)
- S_POR, [942](#)
- S_PULS, [939](#)
- S_PV, [939](#)
- S_RCK, [942](#)
- S_RS, [939](#)
- S_SEL, [939](#)
- S_SELVAL, [939](#)
- S_SR, [939](#)

- S_SUMC, 939
- S_TDE, 939
- S_TDR, 939
- S_TLATCH, 939
- S_VALB, 940
- S_VALC, 940
- SAI, 25, 167, 168, 170, 940
- SAT, 26, 253, 940
- SC2FA, 26, 255, 940
- SCU, 26, 231, 238, 262, 265, 940
- SCUV, 26, 231, 238, 265, 940
- SEL, 173, 940
- SELHEXD, 26, 173, 174, 940
- SELOCT, 173, 174, 940
- SELQUAD, 173, 174, 940
- SELSOCT, 26, 360, 940
- SELU, 26, 268, 537, 940
- SETPA, 19, 26, 76, 83, 365, 366, 381, 384, 940
- SETPB, 383, 385, 386, 940
- SETPI, 383, 385, 386, 582, 940
- SETPR, 19, 26, 76, 365, 383, 385, 386, 389, 940
- SETPS, 19, 26, 76, 365, 385, 386, 940
- SETPX, 26, 384–386
- SG, 18, 189, 201, 570, 940
- SGEN, 430
- SGENTX, 432
- SGI, 201, 940
- SGSLP, 388, 392, 940
- SHIFTOCT, 26, 176, 940
- SHLD, 26, 178, 376, 940
- SILO, 19, 365, 390, 392, 940
- SILOS, 19, 365, 394, 940
- simulation
 - parameters, 47
 - real-time, 47
- Simulink, 47, 569
- SINT, 26, 155, 179, 940
- SLEEP, 47, 940
- SMHCC, 26, 270, 940
- SMHCCA, 26, 274, 940
- SMTP, 20, 561, 593, 940
- SOLNP, 557
- SOPDT, 19, 26, 232, 398, 434, 940
- SPIKE, 26, 170, 171, 180, 940
- SPLIT_DT, 942
- SQR, 18, 26, 94, 131, 940
- SQRT, 18, 26, 94, 124, 132
- SQRT_, 940
- SR, 26, 305, 940
- SRTF, 48, 940
- SSW, 26, 182, 537, 570, 940
- stack
 - size, 39
- STATELOAD, 50
- STATESAVE, 50, 52
- STEAM, 20, 561, 595, 940, 942
- STMGEN, 436
- STOR, 19, 26, 343, 362, 941
- STR2DT, 942
- STURB, 438
- SUB, 18, 26, 94, 97, 133, 941
- SUBSYSTEM, 367
- SubSystem, 71, 74, 77, 82, 87, 941
- subsystem
 - archiving, 19, 323
 - execution, 48
- SWR, 26, 183, 537, 941
- SWU, 26, 268, 281, 941
- SWVMR, 537, 941
- SYSEVENT, 54, 943
- SYSLOG, 56, 943
- T2STR, 943
- TASK, 17, 28, 34, 41, 46, 48, 57, 399, 402, 941
- task
 - execution, 48
 - execution period, 57
 - priority, 57
 - quick, 46
 - quick, execution period, 46
- TB1, 942
- TB2, 942
- TB3, 942
- TB6, 942
- TC, 19, 309, 314

- TESTS, [942](#)
- TIME, [19](#), [309](#), [311](#), [316](#), [941](#)
- TIMER, [18](#), [26](#), [283](#), [306](#)
- timer
 - system, [41](#)
- TIMER_, [941](#)
- TIODRV, [41](#), [59](#), [941](#)
- ToFile, [89](#)
- ToWorkspace, [90](#)
- trajectory
 - time-optimal, [142](#)
- TRIM, [26](#), [363](#), [943](#)
- TRND, [24](#), [30](#), [323](#), [337](#), [340](#), [941](#)
- TRNDV, [323](#), [340](#), [941](#)
- TS, [317](#), [319](#), [337](#)
- TS2NS, [319](#)
- TSE, [26](#), [262](#), [265](#), [282](#), [941](#)
- type
 - input, [23](#)
 - output, [23](#)
 - parameter, [23](#)
- types
 - of variables, [23](#)
- TZ2UTC, [943](#)

- UART, [20](#), [561](#), [583](#), [600](#)
- UNIPi_CHANNEL, [913](#)
- UNIPi_PRODUCT, [915](#)
- UNIPi_S1AI, [916](#)
- UNIPi_S1AOR, [917](#)
- UNIPi_S1CNT, [918](#)
- UNIPi_S1DI, [919](#)
- UNIPi_S1DO, [920](#)
- UNIPi_S1LED, [921](#)
- UNIPi_S1PWM, [922](#)
- UNIPi_S2AI, [923](#)
- UNIPi_S2AO, [924](#)
- UNIPi_S2CNT, [925](#)
- UNIPi_S2DI, [926](#)
- UNIPi_S2RO, [927](#)
- UTC2TZ, [943](#)
- UTOI, [26](#), [134](#), [941](#)

- VAC, [942](#)

- value
 - default, [23](#)
 - maximal, [23](#)
 - minimal, [23](#)
 - substitute, [103](#), [105](#), [106](#), [109](#), [117](#), [132](#)
- valve
 - servo, [423](#)
- VDEL, [26](#), [184](#), [941](#)
- VIN, [25](#), [26](#), [79](#), [80](#), [86](#), [91](#), [580](#), [941](#)
- VOUT, [85](#), [92](#), [580](#), [941](#)
- VTOR, [26](#), [164](#), [538](#), [563](#), [941](#)

- WASM, [941](#), [942](#)
- WEEK, [943](#)
- WSCH, [19](#), [309](#), [320](#), [941](#)
- WWW, [61](#), [941](#)

- ZV4IS, [26](#), [185](#), [941](#)

