

# Function Blocks of REXYGEN Reference manual

REX Controls s.r.o.

Version 2.50.7

2018-05-22

Plzeň (Pilsen), Czech Republic



# Contents

<b>1</b>	<b>Introduction</b>	<b>13</b>
1.1	How to use this manual . . . . .	13
1.2	The function block description format . . . . .	15
1.3	Conventions for variables, blocks and subsystems naming . . . . .	16
1.4	The signal quality corresponding with OPC . . . . .	17
<b>2</b>	<b>EXEC – Real-time executive configuration</b>	<b>19</b>
	ARC – The REXYGEN system archive . . . . .	20
	EXEC – Real-time executive . . . . .	22
	HMI – Human-Machine Interface Configuration . . . . .	25
	INFO – Description of Algorithm . . . . .	27
	IODRV – The REXYGEN system input/output driver . . . . .	28
	IOTASK – Driver-triggered task of the REXYGEN system . . . . .	30
	LPBRK – Loop break . . . . .	31
	MODULE – Extension module of the REXYGEN system . . . . .	32
	PROJECT – Additional Project Settings . . . . .	33
	QTASK – Quick task of the REXYGEN system . . . . .	34
	SLEEP – Timing in Simulink . . . . .	35
	SRTF – Set run-time flags . . . . .	36
	OSCALL – Operating system calls . . . . .	38
	TASK – Standard task of the REXYGEN system . . . . .	39
	TIODRV – The REXYGEN system input/output driver with tasks . . . . .	41
	WWW – Internal Web Server Content . . . . .	43
<b>3</b>	<b>INOUT – Input and output blocks</b>	<b>45</b>
	Display – Numeric display of input values . . . . .	46
	From, INSTD – Signal connection or input . . . . .	47
	Goto, OUTSTD – Signal source or output . . . . .	49
	GotoTagVisibility – Visibility of the signal source . . . . .	51
	Inport, Outport – Input and output port . . . . .	52
	SubSystem – Subsystem block . . . . .	54
	INQUAD, INOCT, INHEXD – Multi-input blocks . . . . .	56
	OUTQUAD, OUTOCT, OUTHEXD – Multi-output blocks . . . . .	58

OUTRQUAD, OUTROCT, OUTRHEXD – Multi-output blocks with verification . . .	60
OUTRSTD – Output block with verification . . . . .	62
QFC – Quality flags coding . . . . .	63
QFD – Quality flags decoding . . . . .	64
VIN – Validation of the input signal . . . . .	65
VOUT – Validation of the output signal . . . . .	66
<b>4 MATH – Math blocks</b>	<b>67</b>
ABS_ – Absolute value . . . . .	69
ADD – Addition of two signals . . . . .	70
ADDQUAD, ADDOCT, ADDHEXD – Multi-input addition . . . . .	71
CNB – Boolean (logic) constant . . . . .	73
CNE – Enumeration constant . . . . .	74
CNI – Integer constant . . . . .	75
CNR – Real constant . . . . .	76
DIF_ – Difference . . . . .	77
DIV – Division of two signals . . . . .	78
EAS – Extended addition and subtraction . . . . .	79
EMD – Extended multiplication and division . . . . .	80
FNX – Evaluation of single-variable function . . . . .	81
FNXY – Evaluation of two-variables function . . . . .	83
GAIN – Multiplication by a constant . . . . .	85
GRADS – Gradient search optimization . . . . .	86
IADD – Integer addition . . . . .	88
ISUB – Integer subtraction . . . . .	90
IMUL – Integer multiplication . . . . .	92
IDIV – Integer division . . . . .	94
IMOD – Remainder after integer division . . . . .	95
LIN – Linear interpolation . . . . .	96
MUL – Multiplication of two signals . . . . .	97
POL – Polynomial evaluation . . . . .	98
REC – Reciprocal value . . . . .	99
REL – Relational operator . . . . .	100
RTOI – Real to integer number conversion . . . . .	101
SQR – Square value . . . . .	102
SQRT_ – Square root . . . . .	103
SUB – Subtraction of two signals . . . . .	104
<b>5 ANALOG – Analog signal processing</b>	<b>105</b>
ABSR0T – Processing data from absolute position sensor . . . . .	107
ASW – Switch with automatic selection of input . . . . .	109
AVG – Moving average filter . . . . .	111
AVS – Motion control unit . . . . .	112
BPF – Band-pass filter . . . . .	113



CMP – Comparator with hysteresis . . . . .	114
CNDR – Nonlinear conditioner . . . . .	115
DEL – Delay with initialization . . . . .	117
DELM – Time delay . . . . .	118
DER – Derivation, filtering and prediction from the last $n+1$ samples . . . . .	119
EVAR – Moving mean value and standard deviation . . . . .	121
INTE – Controlled integrator . . . . .	122
KDER – Derivation and filtering of the input signal . . . . .	124
LPF – Low-pass filter . . . . .	126
MINMAX – Running minimum and maximum . . . . .	127
NSCL – Nonlinear scaling factor . . . . .	128
RDFT – Running discrete Fourier transform . . . . .	129
RLIM – Rate limiter . . . . .	131
S1OF2 – One of two analog signals selector . . . . .	132
SAI – Safety analog input . . . . .	135
SEL – Analog signal selector . . . . .	138
SELQUAD, SELOCT, SELHEXD – Analog signal selectors . . . . .	140
SHIFTOCT – Data shift register . . . . .	142
SHLD – Sample and hold . . . . .	144
SINT – Simple integrator . . . . .	145
SPIKE – Spike filter . . . . .	146
SSW – Simple switch . . . . .	148
SWR – Selector with ramp . . . . .	149
VDEL – Variable time delay . . . . .	150
ZV4IS – Zero vibration input shaper . . . . .	151
<b>6 GEN – Signal generators</b>	<b>155</b>
ANLS – Controlled generator of piecewise linear function . . . . .	156
BINS – Controlled binary sequence generator . . . . .	158
BIS – Binary sequence generator . . . . .	160
MP – Manual pulse generator . . . . .	161
PRBS – Pseudo-random binary sequence generator . . . . .	162
SG, SGI – Signal generators . . . . .	164
<b>7 REG – Function blocks for control</b>	<b>167</b>
ARLY – Advance relay . . . . .	169
FLCU – Fuzzy logic controller unit . . . . .	170
FRID – * Frequency response identification . . . . .	173
I3PM – Identification of a three parameter model . . . . .	176
LC – Lead compensator . . . . .	178
LLC – Lead-lag compensator . . . . .	179
MCU – Manual control unit . . . . .	180
PIDAT – PID controller with relay autotuner . . . . .	182
PIDE – PID controller with defined static error . . . . .	185

PIDGS – PID controller with gain scheduling . . . . .	188
PIDMA – PID controller with moment autotuner . . . . .	190
PIDU – PID controller unit . . . . .	196
PIDUI – PID controller unit with variable parameters . . . . .	199
POUT – Pulse output . . . . .	201
PRGM – Setpoint programmer . . . . .	202
PSMPC – Pulse-step model predictive controller . . . . .	204
PWM – Pulse width modulation . . . . .	208
RLY – Relay with hysteresis . . . . .	210
SAT – Saturation with variable limits . . . . .	211
SC2FA – State controller for 2nd order system with frequency autotuner . . . . .	213
SCU – Step controller with position feedback . . . . .	220
SCUV – Step controller unit with velocity input . . . . .	223
SELU – Controller selector unit . . . . .	227
SMHCC – Sliding mode heating/cooling controller . . . . .	229
SMHCCA – Sliding mode heating/cooling controller with autotuner . . . . .	233
SWU – Switch unit . . . . .	240
TSE – Three-state element . . . . .	241
<b>8 LOGIC – Logic control</b>	<b>243</b>
AND_ – Logical product of two signals . . . . .	244
ANDQUAD, ANDOCT, ANDHEXD – Logical product of multiple signals . . . . .	245
ATMT – Finite-state automaton . . . . .	247
BDOCT, BDHEXD – Bitwise demultiplexers . . . . .	250
BITOP – Bitwise operation . . . . .	251
BMOCT, BMHEXD – Bitwise multiplexers . . . . .	252
COUNT – Controlled counter . . . . .	253
EATMT – Extended finite-state automaton . . . . .	255
EDGE_ – Falling/rising edge detection in a binary signal . . . . .	258
INTSM – Integer number bit shift and mask . . . . .	259
ISSW – Simple switch for integer signals . . . . .	260
INTSM – Integer number bit shift and mask . . . . .	261
ITOI – Transformation of integer and binary numbers . . . . .	262
NOT_ – Boolean complementation . . . . .	264
OR_ – Logical sum of two signals . . . . .	265
ORQUAD, OROCT, ORHEXD – Logical sum of multiple signals . . . . .	266
RS – Reset-set flip-flop circuit . . . . .	268
SR – Set-reset flip-flop circuit . . . . .	269
TIMER_ – Multipurpose timer . . . . .	270
<b>9 TIME – Blocks for handling time</b>	<b>273</b>
DATE_ – Current date . . . . .	274
DATETIME – Get, set and convert time . . . . .	275
TIME – Current time . . . . .	278

WSCH – Weekly schedule . . . . .	279
<b>10 ARC – Data archiving</b>	<b>281</b>
10.1 Functionality of the archiving subsystem . . . . .	282
10.2 Generating alarms and events . . . . .	283
ALB, ALBI – Alarms for Boolean value . . . . .	283
ALN, ALNI – Alarms for numerical value . . . . .	285
10.3 Trends recording . . . . .	287
ACD – Archive compression using Delta criterion . . . . .	287
TRND – Real-time trend recording . . . . .	289
TRNDV – Real-time trend recording with vector input . . . . .	292
TRNDLF – * Real-time trend recording (lock-free) . . . . .	294
TRNDVLF – * Real-time trend recording (for vector signals, lock-free) . . . . .	296
10.4 Archive management . . . . .	297
AFLUSH – Forced archive flushing . . . . .	297
<b>11 STRING – Blocks for string operations</b>	<b>299</b>
CNS – String constant . . . . .	300
CONCAT – Concat string by pattern . . . . .	301
FIND – Find a Substring . . . . .	302
ITOS – Integer number to string conversion . . . . .	303
LEN – String length . . . . .	304
MID – Substring Extraction . . . . .	305
PJROCT – Parse JSON string (real output) . . . . .	306
PJSOCT – Parse JSON string (string output) . . . . .	307
REGEXP – Regular expresion parser . . . . .	308
REPLACE – Replace substring . . . . .	311
RTOS – Real Number to String Conversion . . . . .	312
SELSOCT – String selector . . . . .	313
STOR – String to real number conversion . . . . .	315
<b>12 PARAM – Blocks for parameter handling</b>	<b>317</b>
GETPA – Block for remote array parameter acquirement . . . . .	318
GETPR, GETPI, GETPB – Blocks for remote parameter acquirement . . . . .	320
GETPS – * Block for remote string parameter acquirement . . . . .	322
PARA – Block with input-defined array parameter . . . . .	323
PARR, PARI, PARB – Blocks with input-defined parameter . . . . .	324
PARS – * Block with input-defined string parameter . . . . .	326
SETPA – Block for remote array parameter setting . . . . .	327
SETPR, SETPI, SETPB – Blocks for remote parameter setting . . . . .	329
SETPS – * Block for remote string parameter setting . . . . .	331
SGSLP – Set, get, save and load parameters . . . . .	332
SILO – Save input value, load output value . . . . .	336
SILOS – Save input string, load output string . . . . .	338

<b>13 MODEL – Dynamic systems simulation</b>	<b>341</b>
CDELSSM – Continuous state space model of a linear system with time delay	342
CSSM – Continuous state space model of a linear system	345
DDELSSM – Discrete state space model of a linear system with time delay	348
DSSM – Discrete state space model of a linear system	350
FMUCS – * Import modelu FMU CS (pro Co-Simulation)	352
FMUINFO – * Informace o importovaném modelu FMU	355
FOPDT – First order plus dead-time model	356
MDL – Process model	357
MDLI – Process model with input-defined parameters	358
MVD – Motorized valve drive	359
SOPDT – Second order plus dead-time model	360
<b>14 MATRIX – Blocks for matrix and vector operations</b>	<b>363</b>
CNA – Array (vector/matrix) constant	365
MB_DASUM – Sum of the absolute values	366
MB_DAXPY – Performs $y := a*x + y$ for vectors $x, y$	368
MB_DCOPY – Copies vector $x$ to vector $y$	370
MB_DDOT – Dot product of two vectors	372
MB_DGEMM – Performs $C := \alpha*op(A)*op(B) + \beta*C$ , where $op(X) = X$ or $op(X) = X^T$	374
MB_DGEMV – Performs $y := \alpha*A*x + \beta*y$ or $y := \alpha*A^T*x + \beta*y$	376
MB_DGER – Performs $A := \alpha*x*y^T + A$	378
MB_DNRM2 – Euclidean norm of a vector	380
MB_DRROT – Plain rotation of a vector	382
MB_DSCAL – Scales a vector by a constant	384
MB_DSWAP – Interchanges two vectors	386
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 $A$	388
MB_DTRMV – Performs $x := A*x$ or $x := A^T*x$ for triangular matrix $A$	390
MB_DTRSV – Solves one of the system of equations $A*x = b$ or $A^T*x = b$ for triangular matrix $A$	392
ML_DGEBAK – Backward transformation to ML_DGEBAL of left or right eigenvectors	394
ML_DGEBAL – Balancing of a general real matrix	396
ML_DGEBRD – Reduces a general real matrix to bidiagonal form by an orthogonal transformation	399
ML_DGECON – Estimates the reciprocal of the condition number of a general real matrix	401
ML_DGEEES – Computes the eigenvalues, the Schur form, and, optionally, the matrix of Schur vectors	403
ML_DGEEV – Computes the eigenvalues and, optionally, the left and/or right eigenvectors	405

ML_DGEHRD – Reduces a real general matrix A to upper Hessenberg form . . . . .	407
ML_DGELQF – Computes an LQ factorization of a real M-by-N matrix A . . . . .	409
ML_DGELSD – Computes the minimum-norm solution to a real linear least squares problem . . . . .	411
ML_DGEQRF – Computes an QR factorization of a real M-by-N matrix A . . . . .	413
ML_DGESDD – Computes the singular value decomposition (SVD) of a real M-by-N matrix A . . . . .	415
ML_DTRSYL – Solves the real Sylvester matrix equation for quasi-triangular matrices A and B . . . . .	417
MX_CTODPA – Discretizes continuous model given by (A,B) to (Ad,Bd) using Pade approximations . . . . .	419
MX_DIM – Matrix/Vector dimensions . . . . .	421
MX_DSAGET – Set subarray of A into B . . . . .	422
MX_DSAREF – Set reference to subarray of A into B . . . . .	424
MX_DSASET – Set A into subarray of B . . . . .	426
MX_DTRNSP – General matrix transposition: $B := \alpha * A^T$ . . . . .	428
MX_DTRNSQ – Square matrix in-place transposition: $A := \alpha * A^T$ . . . . .	430
MX_FILL – Fill real matrix or vector . . . . .	432
MX_MAT – Matrix data storage block . . . . .	433
MX_RAND – Randomly generated matrix or vector . . . . .	434
MX_REFCOPY – Copies input references of matrices A and B to their output references . . . . .	436
MX_VEC – Vector data storage block . . . . .	437
MX_WRITE – Write a Matrix/Vector to the console/system log . . . . .	438
RTOV – Vector multiplexer . . . . .	440
SWVMR – Vector/matrix/reference signal switch . . . . .	442
VTOR – Vector demultiplexer . . . . .	443
<b>15 SPEC – Special blocks . . . . .</b>	<b>445</b>
EPC – External program call . . . . .	446
HTTP – HTTP GET or POST request (obsolete) . . . . .	449
HTTP2 – Block for generating HTTP GET or POST requests . . . . .	451
SMTP – Send email message via SMTP . . . . .	453
RDC – Remote data connection . . . . .	455
REXLANG – User programmable block . . . . .	460
<b>16 MC_SINGLE – Motion control - single axis blocks . . . . .</b>	<b>477</b>
RM_Axis – Motion control axis . . . . .	480
MC_AccelerationProfile, MCP_AccelerationProfile – Acceleration profile . . . . .	486
MC_Halt, MCP_Halt – Stopping a movement (interruptible) . . . . .	490
MC_HaltSuperimposed, MCP_HaltSuperimposed – Stopping a movement (superimposed and interruptible) . . . . .	492
MC_Home, MCP_Home – Homing . . . . .	494

MC_MoveAbsolute, MCP_MoveAbsolute – Move to position (absolute coordinate) . . . . .	496
MC_MoveAdditive, MCP_MoveAdditive – Move to position (relative to previous motion) . . . . .	500
MC_MoveRelative, MCP_MoveRelative – Move to position (relative to execution point) . . . . .	503
MC_MoveSuperimposed, MCP_MoveSuperimposed – Superimposed move . .	506
MC_MoveContinuousAbsolute, MCP_MoveContinuousAbsolute – Move to position (absolute coordinate) . . . . .	509
MC_MoveContinuousRelative, MCP_MoveContinuousRelative – Move to position (relative to previous motion) . . . . .	513
MC_MoveVelocity, MCP_MoveVelocity – Move with constant velocity . .	517
MC_PositionProfile, MCP_PositionProfile – Position profile . . . . .	521
MC_Power – Axis activation (power on/off) . . . . .	525
MC_ReadActualPosition – Read actual position . . . . .	527
MC_ReadAxisError – Read axis error . . . . .	528
MC_ReadBoolParameter – Read axis parameter (bool) . . . . .	529
MC_ReadParameter – Read axis parameter . . . . .	530
MC_ReadStatus – Read axis status . . . . .	532
MC_Reset – Reset axis errors . . . . .	534
MC_SetOverride, MCP_SetOverride – Set override factors . . . . .	535
MC_Stop, MCP_Stop – Stopping a movement . . . . .	537
MC_TorqueControl, MCP_TorqueControl – Torque/force control . . . . .	539
MC_VelocityProfile, MCP_VelocityProfile – Velocity profile . . . . .	542
MC_WriteBoolParameter – Write axis parameter (bool) . . . . .	546
MC_WriteParameter – Write axis parameter . . . . .	547
RM_AxisOut – Axis output . . . . .	549
RM_AxisSpline – Commanded values interpolation . . . . .	551
RM_Track – Tracking and inching . . . . .	553
<b>17 MC_MULTI – Motion control - multi axis blocks</b>	<b>555</b>
MC_CamIn, MCP_CamIn – Engage the cam . . . . .	556
MC_CamOut – Disengage the cam . . . . .	560
MCP_CamTableSelect – Cam definition . . . . .	562
MC_CombineAxes, MCP_CombineAxes – Combine the motion of 2 axes into a third axis . . . . .	564
MC_GearIn, MCP_GearIn – Engage the master/slave velocity ratio . . . .	568
MC_GearInPos, MCP_GearInPos – Engage the master/slave velocity ratio in defined position . . . . .	571
MC_GearOut – Disengage the master/slave velocity ratio . . . . .	577
MC_PhasingAbsolute, MCP_PhasingAbsolute – Phase shift in synchronized motion (absolute coordinates) . . . . .	579
MC_PhasingRelative, MCP_PhasingRelative – Phase shift in synchronized motion (relative coordinates) . . . . .	582

<b>18 MC_COORD – Motion control - coordinated movement blocks</b>	<b>585</b>
RM_AxesGroup – Axes group for coordinated motion control . . . . .	588
RM_Feed – * MC Feeder ??? . . . . .	591
RM_Gcode – * CNC motion control . . . . .	593
MC_AddAxisToGroup – Adds one axis to a group . . . . .	595
MC_UngroupAllAxes – Removes all axes from the group . . . . .	596
MC_GroupEnable – Changes the state of a group to GroupEnable . . . . .	597
MC_GroupDisable – Changes the state of a group to GroupDisabled . . . . .	598
MC_SetCartesianTransform – Sets Cartesian transformation . . . . .	599
MC_ReadCartesianTransform – Reads the parameter of the cartesian transformation . . . . .	601
MC_GroupSetPosition, MCP_GroupSetPosition – Sets the position of all axes in a group . . . . .	603
MC_GroupReadActualPosition – Read actual position in the selected coordinate system . . . . .	605
MC_GroupReadActualVelocity – Read actual velocity in the selected coordinate system . . . . .	606
MC_GroupReadActualAcceleration – Read actual acceleration in the selected coordinate system . . . . .	607
MC_GroupStop – Stopping a group movement . . . . .	608
MC_GroupHalt – Stopping a group movement (interruptible) . . . . .	611
MC_GroupInterrupt, MCP_GroupInterrupt – Read a group interrupt . . . . .	616
MC_GroupContinue – Continuation of interrupted movement . . . . .	618
MC_GroupReadStatus – Read a group status . . . . .	619
MC_GroupReadError – Read a group error . . . . .	621
MC_GroupReset – Reset axes errors . . . . .	622
MC_MoveLinearAbsolute – Linear move to position (absolute coordinates) . . . . .	623
MC_MoveLinearRelative – Linear move to position (relative to execution point) . . . . .	627
MC_MoveCircularAbsolute – Circular move to position (absolute coordinates) . . . . .	631
MC_MoveCircularRelative – Circular move to position (relative to execution point) . . . . .	635
MC_MoveDirectAbsolute – Direct move to position (absolute coordinates) . . . . .	639
MC_MoveDirectRelative – Direct move to position (relative to execution point) . . . . .	642
MC_MovePath – General spatial trajectory generation . . . . .	645
MC_GroupSetOverride – Set group override factors . . . . .	647
<b>A Licensing options</b>	<b>649</b>
<b>B Licensing of individual function blocks</b>	<b>651</b>
<b>C Error codes of the REXYGEN system</b>	<b>661</b>

<b>Bibliography</b>	<b>667</b>
<b>Index</b>	<b>669</b>

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



# Chapter 1

## Introduction

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

### 1.1 How to use this manual

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

The content of individual chapters of this manual is following:

#### 1 Introduction

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

#### 2 EXEC – Real-time executive configuration

Blocks used mainly for configuration of the structure, priorities and timing of individual objects linked to the real-time subsystem of the REXYGEN system (the **RexCore** program) are described in this chapter. These blocks are not used for simulation in Simulink except two special blocks **LPBRK** and **SLEEP** which are essential for executing the simulation in Simulink environment.

#### 3 INOUT – Input and output blocks

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

#### 4 MATH – Mathematic blocks

The blocks for arithmetic operations and basic math functions. Similar blocks can be found in native Simulink libraries, but only blocks from this library can be used for applications whose target platform is the REXYGEN control system.

#### 5 ANALOG – Analog signal processing

The integrator, derivator, time delay, moving average, various filters, comparators and selectors can be found among the blocks for analog signal processing. The starting unit block ([AVS](#)) is also very interesting.

#### 9 GEN – Signal generators

This chapter deals with analog and logic signal generators.

#### 7 REG – Function blocks for control

The control function blocks form the most extensive sublibrary of the *RexLib* library. Blocks ranging from simple dynamic compensators to several modifications of PID (P, I, PI, PD a PID) controller and some advanced controllers are included. The blocks for control schemes switching and conversion of output signals for various types of actuators can be found in this sublibrary. The involved controllers include the [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

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

#### 10 ARC – Data archiving

This sublibrary contains blocks for alarms generation and blocks for storing trend data directly on the target device. No such blocks can be found in the Simulink system.

#### 12 PARAM – Parameter handling

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

#### 13 MODEL – Dynamic systems modeling

The REXYGEN system can also be used for creating real-time mathematical models of dynamic systems. The function blocks of this sublibrary were developed for such cases.

#### 14 MATRIX – Working with matrix and vector data

Function blocks for handling vector and matrix data in REXYGEN are included in this sublibrary.

**16 MC\_SINGLE – Single-axis motion control**

Function blocks of this sublibrary were developed according to the PLCopen Motion Control standard for single axis motion control.

**17 MC\_MULTI – Multi-axes motion control**

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

**18 MC\_COORD – Coordinated motion control**

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

**15 SPEC – Special blocks**

The most interesting blocks of this sublibrary are the [REXLANG](#) and [RDC](#) blocks. It is possible to compile and interpret user algorithms using the [REXLANG](#) block, whose programming language is very similar to the C language (the syntax of the [REXLANG](#) commands is mostly the same as in the C language). The [RDC](#) block can be used for real-time communication between two Simulinks (even on two different networked computers), two [REXYGEN](#) targets or between the Simulink and the [REXYGEN](#) system. The [RDC](#) block can also provide data for the Matlab OPC server.

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>** [*nam*] Detailed description of the input (output, parameter) **<name>**. **<type>**  
 Mathematical symbol *nam* on the right side of the first column is used in the equations in the **Function Description** section. It is listed only if it differs from the name more than typographically. If the variable value is limited to only enumerated values, the meaning of these values is explained in this column. [ $\odot$ <def>] [ $\downarrow$ <min>] [ $\uparrow$ <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** for integer variables and **double** 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  $\odot$  symbol. Also it has upper and lower limits, preceded by the symbols  $\downarrow$  and  $\uparrow$  respectively. All these three values are optional (marked by [ ]). If the value  $\odot$ <def> is not listed in the second column, it is equal to zero. If the values of  $\downarrow$ <min> and/or  $\uparrow$ <max> are missing, the limits are given by the the minimum and/or maximum of the corresponding type (see table 1.1).

Type	Meaning	Minimum	Maximum
<b>bool</b>	Boolean value 0 or 1	0	1
<b>byte</b>	8 bit integer number without the sign	0	255
<b>short</b>	16 bit integer number with the sign	-32768	32767
<b>long</b>	32 bit integer number with the sign	-2147483648	2147483647
<b>word</b>	16 bit integer number without the sign	0	65535
<b>dword</b>	32 bit integer number without the sign	0	4294967295
<b>float</b>	32 bit real number in floating point arithmetics	< -3.4E+38	>3.4E+38
<b>double</b>	64 bit real number in floating point arithmetics	< -1.7E+308	>1.7E+308
<b>string</b>	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) is referred to as **on** in this manual. Similarly the logic zero (no, false, off) is represented by **off**. Nevertheless, some tools may display these values as **on** for 1 and **off** for 0 for Matlab-Simulink compatibility reasons. The names of logic variables consist of uppercase letters, e.g. **RUN**, **YCN**, **R1**, **UP**, etc.

**long** – 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**, **pptype1**, **pfac** and **afac** in the **TRND** block, etc.

**double** – 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. **k**, **hilim**, **y**, **ti**, **tt**.

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

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

## 1.4 The signal quality corresponding with OPC

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

Bit number	7	6	5	4	3	2	1	0
Bit weight	128	64	32	16	8	4	2	1
<b>Bit field</b>	<b>Quality</b>		<b>Substatus</b>				<b>Limits</b>	
	<b>Q</b>	<b>Q</b>	<b>S</b>	<b>S</b>	<b>S</b>	<b>S</b>	<b>L</b>	<b>L</b>
<b>BAD</b>	0	0	S	S	S	S	L	L
<b>UNCERTAIN</b>	0	1	S	S	S	S	L	L
not used in OPC	1	0	S	S	S	S	L	L
<b>GOOD</b>	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 [\[1\]](#), chapter 6.8.

## Chapter 2

# EXEC – Real-time executive configuration

### Contents

---

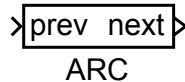
ARC – The <b>REXYGEN</b> system archive . . . . .	20
EXEC – Real-time executive . . . . .	22
HMI – Human-Machine Interface Configuration . . . . .	25
INFO – Description of Algorithm . . . . .	27
IODRV – The <b>REXYGEN</b> system input/output driver . . . . .	28
IOTASK – Driver-triggered task of the <b>REXYGEN</b> system . . . . .	30
LPBRK – Loop break . . . . .	31
MODULE – Extension module of the <b>REXYGEN</b> system . . . . .	32
PROJECT – Additional Project Settings . . . . .	33
QTASK – Quick task of the <b>REXYGEN</b> system . . . . .	34
SLEEP – Timing in Simulink . . . . .	35
SRTF – Set run-time flags . . . . .	36
OSCALL – Operating system calls . . . . .	38
TASK – Standard task of the <b>REXYGEN</b> system . . . . .	39
TIODRV – The <b>REXYGEN</b> system input/output driver with tasks . . . . .	41
WWW – Internal Web Server Content . . . . .	43

---

## ARC – The REXYGEN system archive

Block Symbol

Licence: [STANDARD](#)



### Function Description

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

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

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

### Input

<b>prev</b>	Input for connecting with the <b>next</b> output of the preceding archive or with the <b>Archives</b> output of the <a href="#">EXEC</a> block in the case of the first archive	<b>long</b>
-------------	---	-------------

### Output

<b>next</b>	Output for creating sequences of archives by connecting to the <b>prev</b> input of the following archive	<b>long</b>
-------------	---	-------------



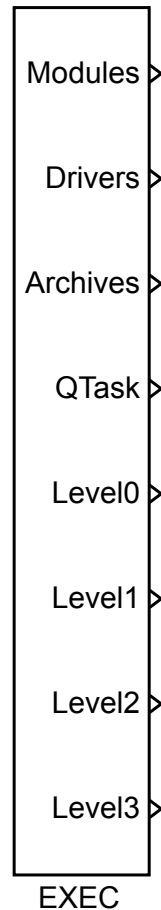
## Parameters

<b>atype</b>	Archive type	⊙1	long
	1 ..... archive is allocated in the RAM memory (data is irreversibly lost after restarting the target device)		
	2 ..... archive is allocated in backed-up memory, e.g. CMOS (data remains available after restarting the target device)		
	3 ..... archive is allocated on a drive (data remains available in the file after restarting)		
<b>asize</b>	Size of the archive in bytes	↓256 ⊙102400	long
<b>nmarks</b>	Number of time stamps for speeding-up sequential seeking in the archive	↓2 ⊙720	long
<b>ldaymax</b>	Maximum size of archive per day [bytes]	↓1000 ↑2147480000 ⊙1048576	large
<b>period</b>	Period of writing data to disk [s]	⊙60.0	double

## EXEC – Real-time executive

Block Symbol

Licence: [STANDARD](#)



### Function Description

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

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

[TASK](#) function block description for details).

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

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

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

## Outputs

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

## Parameters

<b>target</b>	Target device Generic target device	⊙PC - Windows	string
<b>tick</b>	The base period (tick) of the <b>REXYGEN</b> system core and also the quick task ( <a href="#">QTASK</a> ) period (in seconds)	⊙0.05	double
<b>ntick0</b>	The multiplication <code>tick*ntick0</code> determines the period of tasks connected to <b>Level0</b>	↓1 ⊙10	long
<b>ntick1</b>	The multiplication <code>tick*ntick1</code> determines the period of tasks connected to <b>Level1</b>	↓ntick0+1 ⊙50	long

<code>ntick2</code>	The multiplication <code>tick*ntick2</code> determines the period of tasks connected to Level2	$\downarrow \text{ntick1} + 1 \odot 100$	long
<code>ntick3</code>	The multiplication <code>tick*ntick3</code> determines the period of tasks connected to Level3	$\downarrow \text{ntick2} + 1 \odot 1200$	long
<code>pri0</code>	Priority of all Level0 tasks	$\downarrow 3 \uparrow 31 \odot 5$	long
<code>pri1</code>	Priority of all Level1 tasks	$\downarrow \text{pri0} + 1 \uparrow 31 \odot 9$	long
<code>pri2</code>	Priority of all Level2 tasks	$\downarrow \text{pri1} + 1 \uparrow 31 \odot 13$	long
<code>pri3</code>	Priority of all Level3 tasks	$\downarrow \text{pri2} + 1 \uparrow 31 \odot 18$	long

## HMI – Human-Machine Interface Configuration

Block Symbol

Licence: [STANDARD](#)



HMI

### Function Description

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

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

- **WebWatch** is an auto-generated HMI from the **REXYGEN Studio** development tool during project compilation. It has similar look, attributes and functions as the online mode of the **REXYGEN Studio** development tool. The main difference is that **WebWatch** is stored on the target device, is available from the integrated web server and may be viewed with any modern web browser or any application that is compatible with HTML, SVG and JavaScript. The **WebWatch** is a perfect tool for instant creation of HMI that is suitable for system developers or integrators. It provides a graphical interaction with almost all signals in the control algorithm.
- **WebBuDi**, which is an acronym for Web Buttons and Displays, is a simple JavaScript file with several declarative blocks that describe data points which the HMI is connected to and assemble a table in which all the data is presented. It provides a textual interaction with selected signals and is suitable for system developers and integrators or may serve as a fall-back mode HMI for non-standard situations.
- **RexHMI** is a standard SVG file that is edited with the **HMI Designer** with the *RexHMI* extensions. The **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 **HMI Designer** must be installed.

## Parameters

<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 files	<input type="radio"/> on	bool
<code>GenerateRexHMI</code>	Generate HMI from SVG and JS files	<input type="radio"/> on	bool
<code>RedirectToHMI</code>	Web server will automatically redirect to HMI webpage if enabled otherwise it will serve a standard home page as a starting page.	<input type="radio"/> on	bool
<code>Compression</code>	Enables data compression in gzip format.		bool

## INFO – Description of Algorithm

Block Symbol

Licence: [STANDARD](#)



INFO

### Function Description

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

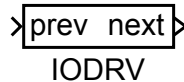
### Parameters

<b>Title</b>	Project title	<b>string</b>
<b>Author</b>	Project author	<b>string</b>
<b>Description</b>	Brief description of the project	<b>string</b>
<b>Customer</b>	Information about a customer	<b>string</b>

## IODRV – The REXYGEN system input/output driver

Block Symbol

Licence: [STANDARD](#)



### Function Description

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

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

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

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



program.

## Input

<code>prev</code>	Input for connecting the driver with the <code>Drivers</code> output of the <code>EXEC</code> block or with the <code>next</code> output of the preceding driver	long
-------------------	--	------

## Output

<code>next</code>	Output for connecting to the <code>prev</code> input of the succeeding driver	long
-------------------	---	------

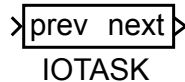
## Parameters

<code>module</code>	Name of the module, which includes the input/output driver (mandatory only if module name differs from <code>classname</code> )	<code>string</code>
<code>classname</code>	I/O driver class name; case sensitive!	<code>⊙DrvClass string</code>
<code>cfgname</code>	Name of the driver configuration file	<code>⊙iodrv.rio string</code>
<code>factor</code>	Multiple of the <code>EXEC</code> block's <code>tick</code> parameter defining the driver's task execution period	long <code>↓1 ⊙10</code>
<code>stack</code>	Stack size of the driver's task in bytes	<code>↓1024 ⊙10240 long</code>
<code>pri</code>	Logical priority of the driver's task	<code>↓1 ↑31 ⊙3 long</code>
<code>timer</code>	Driver is a source of time	<code>bool</code>

## IOTASK – Driver-triggered task of the REXYGEN system

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.

### Input

<code>prev</code>	Input for connecting the first task to the <code>Tasks</code> output of the <a href="#">TIODRV</a> block or for connecting to the previous task's <code>next</code> output	<code>long</code>
-------------------	--	-------------------

### Output

<code>next</code>	Output for sequencing the tasks by connecting to the <code>prev</code> input of the following task	<code>long</code>
-------------------	--	-------------------

### Parameters

<code>factor</code>	Execution factor which can be used to determine the task execution period, see the user guide of the corresponding I/O driver	<code>long</code>
<code>stack</code>	Stack size [bytes]	<code>long</code>
<code>filename</code>	Name of the file with the <code>.mdl</code> extension which contains the task algorithm; in the case <code>filename</code> is not specified, the filename is given by the name of the <code>IOTASK</code> block in the project main file (the <code>.mdl</code> extension is attached automatically)	<code>string</code>

## LPBRK – Loop break

Block Symbol

Licence: [STANDARD](#)



### Function Description

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

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

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

### Input

u	Input signal	double
---	--------------	--------

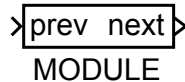
### Output

y	Output signal	double
---	---------------	--------

## MODULE – Extension module of the REXYGEN system

Block Symbol

Licence: [STANDARD](#)



### Function Description

The REXYGEN system has an open architecture thus its functionality can be extended. Such extension is provided by modules. Each module is identified by its name placed below the block symbol. The individual modules are added to the REXYGEN system configuration 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 configuration. There can be only one module connected to the **next** output of the **MODULE** block. The **next** output of the last module in the configuration remains unconnected. This means that the modules create a unidirectional chain which defines the order of initialization and execution of the individual modules.

Each module exists in two versions: one for the development platform (**Host**) and one for the target platform (**Target**). The modules are implemented as DLL libraries in Windows and Windows CE operating systems. The naming `<modname>_H.dll` (for development platform) and `<modname>_T.dll` (for target platform) is used, where `<modname>` is the module name.

### Input

<b>prev</b>	Input for connecting the module with the <b>Modules</b> output of the <a href="#">EXEC</a> block or with the <b>next</b> output of the preceding module	long
-------------	---	------

### Output

<b>next</b>	Output for connecting to the <b>prev</b> input of the succeeding module	long
-------------	---	------

## PROJECT – Additional Project Settings

Block Symbol

Licence: [STANDARD](#)




### Function Description

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

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

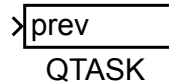
### Parameters

<b>CompileParams</b>	Command-line options which are passed to REXYGEN Compiler during project compilation.	<b>string</b>
<b>SourcesOnTarget</b>	Store source files on target device	 <b>bool</b>
<b>TargetURL</b>	URL address of a target on which the configuration should be run. The address is inserted into all connection dialogs automatically.	<b>string</b>

## QTASK – Quick task of the REXYGEN system

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 execution time of the block is displayed in the **REXYGEN** Diagnostics diagnostic program.

### Input

<b>prev</b>	Input for connecting the task with the <b>QTask</b> output of the <a href="#">EXEC</a> block	long
-------------	--	------

### Parameters

<b>factor</b>	Multiple of the <a href="#">EXEC</a> block's <b>tick</b> parameter defining the quick task execution period	long ⊙1
<b>stack</b>	Stack size [bytes]	⊙10240 long
<b>filename</b>	Name of the file with the <b>.mdl</b> extension which contains the quick task algorithm; in the case <b>filename</b> is not specified, the filename is given by the name of the <b>QTASK</b> block in the project main file (the <b>.mdl</b> extension is attached automatically)	string

## SLEEP – Timing in Simulink

Block Symbol

Licence: [STANDARD](#)



### Function Description

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

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

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

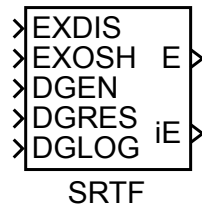
### Parameter

<b>ts</b>	Simulation scheme execution period (in seconds)	⊙0.1    double
-----------	---	----------------

## 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 run-time flags allow the following operations:

- **Disable execution** of the object by setting the **EXDIS** input to **on**. The execution can be enabled again by using the input signal **EXDIS = off**. The **EXDIS** input sets the same run-time flag as the **Halt/Run** button in the upper right corner of the **Workspace** tab in the **REXYGEN** Diagnostics diagnostic program.
- **One-shot execution** of the object. If the object execution is disabled by the **EXDIS = on** input or by the **REXYGEN** Diagnostics program, it is possible to trigger one-shot execution by **EXOSH = on**.
- **Enable diagnostics** for the given object by **DGEN = on**. The result is equivalent to ticking the **Enable** checkbox in the diagnostic pane of the corresponding tab (**I/O Driver**, **Level**, **Quick Task**, **Task**, **I/O Task**, **Sequence**) in the **REXYGEN** Diagnostics program.
- **Reset diagnostic data** of the given object by **DGRES = on**. The same flag can be set by the **Reset** button in the diagnostic pane of the corresponding tab in the **REXYGEN** Diagnostics program. The flag is automatically set back to 0 when the data reset is performed.

The following table shows the flags available for various objects in the **REXYGEN** system.



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

## Inputs

EXDIS	Disable execution	bool
EXOSH	One-shot execution	bool
DGEN	Enable diagnostics	bool
DGRES	Reset diagnostic data	bool
DLOG	Enable more verbose logging	bool

## Outputs

E	Error flag	bool
	off ... No error	
	on .... An error occurred	
iE	Error code (for E = on)	long
	0 ..... No error	
	1 ..... The object specified by the <b>bname</b> parameter was not found	
	2 ..... REXYGEN system internal error (invalid pointers)	
	3 ..... Flag could not be set (timeout)	

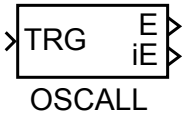
## Parameter

<b>bname</b>	Full path to the block/object. Case sensitive. Individual layers are separated by dots, the object names excluding tasks ( <a href="#">TASK</a> , <a href="#">QTASK</a> ) start with the following special characters: ^ ..... Computational level, e.g. ^0 for Level0 & ..... Input/Output Driver, e.g. &WcnDrv Name of the task triggered by input/output driver ( <a href="#">IOTASK</a> ) has the form &<driver_name>.<task_name>.	string
--------------	---	--------

## OSCALL – Operating system calls

Block Symbol

Licence: [STANDARD](#)



OSCALL

### Function Description

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

Note that there is also the [EPC](#) block available, which allows execution of external programs.

### Input

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

### Outputs

<code>E</code>	Error flag	<code>bool</code>
<code>iE</code>	Error code	<code>long</code>
	<code>i</code> ..... REXYGEN general error	

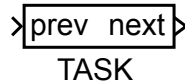
### Parameter

<code>action</code>	System function to perform	$\odot$ 1 <code>long</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	

## TASK – Standard task of the REXYGEN system

Block Symbol

Licence: [STANDARD](#)



### Function Description

The overall control algorithm of the REXYGEN system consists of individual tasks. These are included by using the **TASK** block. There can be one or more tasks in the control algorithm. The REXYGEN system contains four main computational levels represented by the **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 the tasks of the given level *<i>* are executed with the same priority given by the **pri<i>** parameter of the [EXEC](#) block. The execution period of the task is given by a multiple of the **factor** parameter and the base tick of the given level *<i>* **ntick<i>**\***tick** in the [EXEC](#) block. The time allocated for the task to execute starts at the **start** tick and ends at the **stop** tick, where the inequality  $0 \leq \text{start} < \text{stop} \leq \text{ntick< i> tick}$  must hold for the **start** and **stop** parameters. The REXYGEN Compiler compiler further checks whether the **stop** parameter of the preceding task is less or equal to the **stop** parameter of the succeeding task, i.e. the allocated time intervals for individual tasks cannot overlap. In the case the timing of individual levels is inappropriate, the tasks are interrupted by tasks and other events with higher priority and might not execute in the allocated time. In such a case the execution is not aborted but delayed (in contrary to the [QTASK](#) block). The REXYGEN Diagnostics program diagnoses whether the execution delay is occasional or permanent (the **Level** and **Task** tabs).

### Input

<b>prev</b>	Input for connecting the task with the corresponding <b>Level&lt;i&gt;</b> output of the <a href="#">EXEC</a> block or with the <b>next</b> output of the preceding task of the given level	<b>long</b>
-------------	---	-------------

## Output

<code>next</code>	Output for connecting to the <code>prev</code> input of the succeeding task in the given level	<code>long</code>
-------------------	--	-------------------

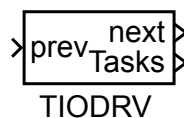
## Parameters

<code>factor</code>	Execution factor; multiple of the execution period of the <code>i</code> -th level of the <code>EXEC</code> block defining the execution period of the task: $\text{factor} * \text{tick} * \text{ntick} \langle i \rangle$	<code>long</code> $\odot 1$
<code>start</code>	Number of tick of the given computational level which should trigger the task execution	<code>long</code> $\downarrow 0 \uparrow \text{ntick} \langle i \rangle \odot 0$
<code>stop</code>	Number of tick of the given computational level by which the task execution should finish	<code>long</code> $\downarrow \text{start} + 1 \uparrow \text{ntick} \langle i \rangle \odot 1$
<code>stack</code>	Stack size [bytes]	<code>long</code> $\odot 10240$
<code>filename</code>	Name of the file with the <code>.mdl</code> extension which contains the task algorithm. In the case <code>filename</code> is not specified, the filename is given by the name of the <code>TASK</code> block in the project main file (the <code>.mdl</code> extension is attached automatically)	<code>string</code>

## TIODRV – The REXYGEN system 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 **REXYGEN Diagnostics** program.

### Input

<b>prev</b>	Input for connecting the driver with the <b>Drivers</b> output of the <b>EXEC</b> block or with the <b>next</b> output of the preceding driver	long
-------------	--	------

### Outputs

<b>next</b>	Output for connecting to the <b>prev</b> input of the succeeding driver	long
<b>Tasks</b>	The <b>IOTASK</b> blocks executed by the driver are connected to this output using the <b>prev</b> input	long

### Parameters

<b>module</b>	Name of the module, which includes the input/output driver (mandatory only if module name differs from <b>classname</b> )	<b>string</b>
<b>classname</b>	Name of the driver class; case sensitive!	⊙DrvClass <b>string</b>
<b>cfgname</b>	Name of the driver configuration file	⊙iodrv.rio <b>string</b>

<b>factor</b>	Multiple of the <b>EXEC</b> block's <b>tick</b> parameter defining the driver's task execution period	$\downarrow 1 \odot 10$	<b>long</b>
<b>stack</b>	Stack size of the driver's task in bytes	$\downarrow 1024 \odot 10240$	<b>long</b>
<b>pri</b>	Logical priority of the driver's task	$\downarrow 1 \uparrow 31 \odot 3$	<b>long</b>
<b>timer</b>	Driver is a source of time		<b>bool</b>

## WWW – Internal Web Server Content

Block Symbol

Licence: [STANDARD](#)



### Function Description

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

### Parameters

<b>Source</b>	Specifies a source directory or a file name that should be placed on the target and should be available via integrated web server using standard HTTP and/or HTTPS protocol. The path may be absolute or relative to path of a main project file.	<b>string</b>
<b>Target</b>	Specifies a target directory or a file name on the integrated web server.	<b>string</b>
<b>Compression</b>	Enables data compression in gzip format.	<b>bool</b>





## Chapter 3

# INOUT – Input and output blocks

### Contents

---

Display – Numeric display of input values . . . . .	46
From, INSTD – Signal connection or input . . . . .	47
Goto, OUTSTD – Signal source or output . . . . .	49
GotoTagVisibility – Visibility of the signal source . . . . .	51
Inport, Outport – Input and output port . . . . .	52
SubSystem – Subsystem block . . . . .	54
INQUAD, INOCT, INHEXD – Multi-input blocks . . . . .	56
OUTQUAD, OUTOCT, OUTHEXD – Multi-output blocks . . . . .	58
OUTRQUAD, OUTROCT, OUTRHEXD – Multi-output blocks with verification	60
OUTRSTD – Output block with verification . . . . .	62
QFC – Quality flags coding . . . . .	63
QFD – Quality flags decoding . . . . .	64
VIN – Validation of the input signal . . . . .	65
VOUT – Validation of the output signal . . . . .	66

---

Display – Numeric display of input values

Block Symbol

Licence: [STANDARD](#)



Function Description

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

Input

u	Input signal	unknown
---	--------------	---------

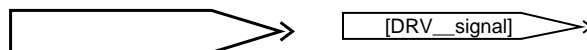
Parameters

Format	Format of displayed value	⊙1	long
	<code>Best fit</code>		
	<code>short</code>		
	<code>long ..</code>		
	<code>short_e</code>		
	<code>long_e</code>		
	<code>bank ..</code>		
	<code>hex ...</code>		
	<code>bin ...</code>		
	<code>det ...</code>		
Decimation	Value is evaluated in each <code>Decimation</code> period	↓1 ↑100000 ⊙1	long
Suffix	A string to be appended to the value		string

## From, INSTD – Signal connection or input

### Block Symbols

Licence: [STANDARD](#)



### Function Description

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

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

- If the parameter **GotoTag** contains the `__` delimiter (two successive `'_'` characters), then the block is of the **INSTD** type. The part (substring) of the parameter before the delimiter (**DRV** in the block symbol above) is considered to be the name of an **IODRV** type block contained in the main file of the project. The **REXYGEN Compiler** compiler returns an error when such block does not exist. If the driver exists in the project, the other part of the **GotoTag** parameter (following the delimiter, **signal** in this case) is considered to be the name of a signal within the corresponding driver. This name is validated by the driver and in the case of success, an instance of the **INSTD** block is created. This instance collects real-time data from the driver and feeds the data into the control algorithm at each execution of the task it is included in.
- If there is no `__` delimiter in the **GotoTag** parameter, the block is of type **From**. A matching **Goto** block with the same **GotoTag** parameter and required visibility given by the **TagVisibility** parameter (see the **Goto** block description) is searched. In case it is not found, the **REXYGEN Compiler** compiler issues a warning and deletes the **From** block. Otherwise an "invisible" connection is created between the corresponding blocks. The **From** block is removed also in this case and thus it is not contained in the resulting control system configuration.

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

`<DRV>__<signal>`

E.g. the first digital input of a Modbus I/O device might be referenced by `MBM__DI1`. Detailed information about signal naming can be found in the user manual of the corresponding I/O driver.

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

## Output

<code>value</code>	Signal coming from I/O driver or <a href="#">Goto</a> block. The type of output is determined by the type of the signal which is being referred by the <code>GotoTag</code> parameter.	<code>unknown</code>
--------------------	--	----------------------

## Parameter

<code>GotoTag</code>	Reference to a <a href="#">Goto</a> block with the same <code>GotoTag</code> parameter, which should be connected with the <code>From</code> block or a reference to input signal of the REXYGEN I/O driver, which should provide data through the block's output.	<code>string</code>
----------------------	--	---------------------

## Goto, OUTSTD – Signal source or output

### Block Symbols

Licence: [STANDARD](#)



### Function Description

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

In the function block library, you can only find the **Goto** block. It is converted to the **OUTSTD** block at compile time if necessary. The following rules define how the **REXYGEN Compiler** 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** 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** issues a warning and deletes the **Goto** block. Otherwise an "invisible" connection is created between the corresponding blocks. The **Goto** block is removed also in this case thus it is not contained in the resulting control system configuration.

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

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

```
<DRV>__<signal>
```

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.

## Input

<b>value</b>	Signal going to I/O driver or <a href="#">From</a> block. In case of connection to an I/O driver, the type of input is determined by the I/O driver from the <code>GotoTag</code> parameter.	<b>unknown</b>
--------------	--	----------------

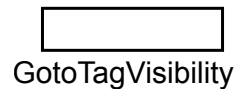
## Parameters

<b>GotoTag</b>	Reference to a <a href="#">From</a> block with the same <code>GotoTag</code> parameter, which should be connected with the <code>Goto</code> block or a reference to output signal of the REXYGEN control system driver, which should send the data from block input to the process.	<b>string</b>
<b>TagVisibility</b>	Visibility (availability) of the block within the <code>.mdl</code> file. Defines conditions under which the two corresponding <code>Goto</code> and <a href="#">From</a> blocks are reciprocally available: <div style="margin-left: 20px;"> <div> <div>local</div> <div>the two blocks must be in the same subsystem</div> </div> <div> <div>global</div> <div>blocks can be anywhere in the given task (<code>.mdl</code> file)</div> </div> <div> <div>scoped</div> <div>the <a href="#">From</a> block must be placed in the same subsystem or in any lower hierarchical level below the <a href="#">GotoTagVisibility</a> block with the same <code>GotoTag</code> parameter</div> </div> </div> <div style="text-align: right;">☉local</div>	<b>string</b>

## GotoTagVisibility – Visibility of the signal source

Block Symbol

Licence: [STANDARD](#)



### Function Description

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

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

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

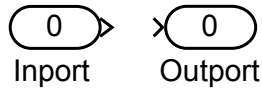
### Parameter

<b>GotoTag</b>	Reference to a <b>Goto</b> block with the <b>GotoTag</b> parameter, whose <b>string</b> visibility is defined by the position of this block ( <b>GotoTagVisibility</b> )
----------------	--

## 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 connection of **Inport** and **Outport** blocks in various tasks is checked and created by the **REXYGEN Compiler** compiler.

The ordering of the blocks to be connected is based on the **Port** parameter of the given block. The numberings of the input and output ports are independent on each other. The numbering is automatic in both the **REXYGEN Studio** and the Matlab-Simulink system, 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!

There are other functionalities of the port blocks in the Matlab-Simulink environment, but these are not used in the **REXYGEN** control system. Detailed description of the blocks for Matlab-Simulink can be found in [2].

### Input

value	Value going to the output pin or <b>Inport</b>	unknown
-------	--	---------

### Output

value	Value coming from the input pin or <b>Outport</b>	unknown
-------	---	---------



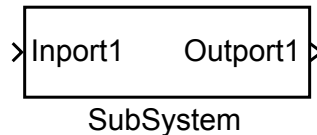
## Parameter

Port	Ordering of the Inport or Outport pins	long
------	--	------

## SubSystem – Subsystem block

Block Symbol

Licence: [STANDARD](#)



### Function Description

The **Subsystem** block is a cornerstone of hierarchical organization of a control (or simulation) algorithm. It allows embedding a subsystem into another system (or subsystem). The subsystem contains blocks and their connections. The subsystem is executed as ordered sequence of blocks during real-time operation of the REXYGEN system. Therefore it 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.

There are two possible ways of creating a subsystem in the REXYGEN Studio program:

- Copy the **Subsystem** block from the **INOUT** library to the given diagram (.mdl file). Blocks can be inserted into the subsystem upon its opening (including **Inport** and **Outport** blocks).
- Select a group of blocks and use the **Create subsystem** command (**Create subsystem** in the **Edit** menu). The selected blocks are then replaced by the subsystem block, which contains all the original blocks and **Inport** and **Outport** blocks for signals crossing the subsystem boundary. Ports for all unconnected inputs and outputs are created as well.

Once a subsystem is created, you can enter it with a double-click.

It is also possible to create the so-called subsystem mask and define parameters whose values can be used inside the subsystem. Select a subsystem block and go to menu **Edit**→**Subsystem Mask**. A dialog appears and you can define parameters and the corresponding descriptions.

Once a mask is created, the subsystem acts just like any other block – double-click it to open the *Block properties* dialog. You can see the dialog contains the parameters defined in the subsystem mask. If you need to edit the internals of a masked subsystem, select it and go to menu **Edit**→**Look under Mask**.

See Example 0101-02 demonstrating the use of subsystems. The example is contained in the development tools of the REXYGEN system.

### Inputs

The ordering and names of the inputs are given by the numbers and names of the `Inport` blocks contained within the subsystem.

## Outputs

The ordering and names of the outputs are given by the numbers and names of the `Outport` blocks contained within the subsystem.

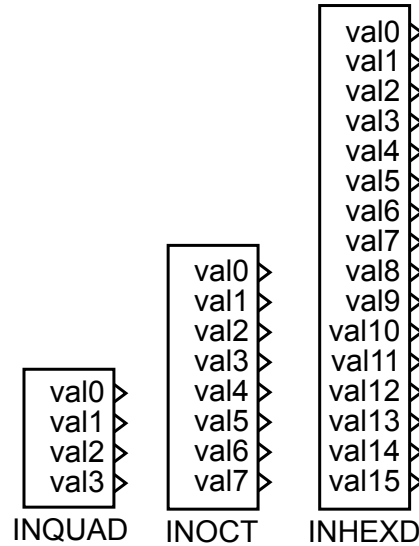
## Parameters

The parameters of the subsystem are defined by the subsystem mask.

## 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.

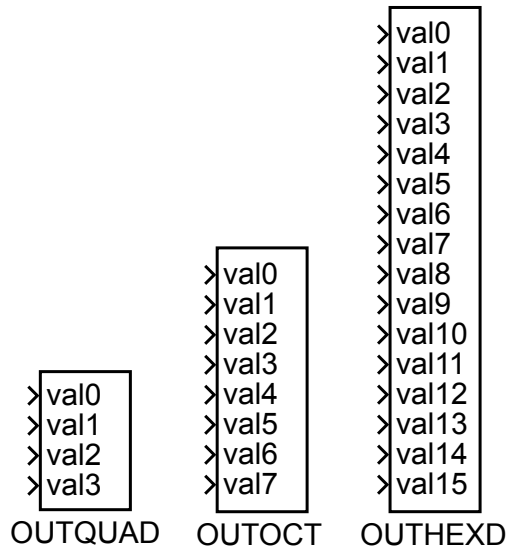
## Outputs

<code>vali</code>	Input signals fed into the control algorithm through input/output drivers. The type and location of individual signals is described in the user manual for the given driver.	<code>unknown</code>
-------------------	--	----------------------

## 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.

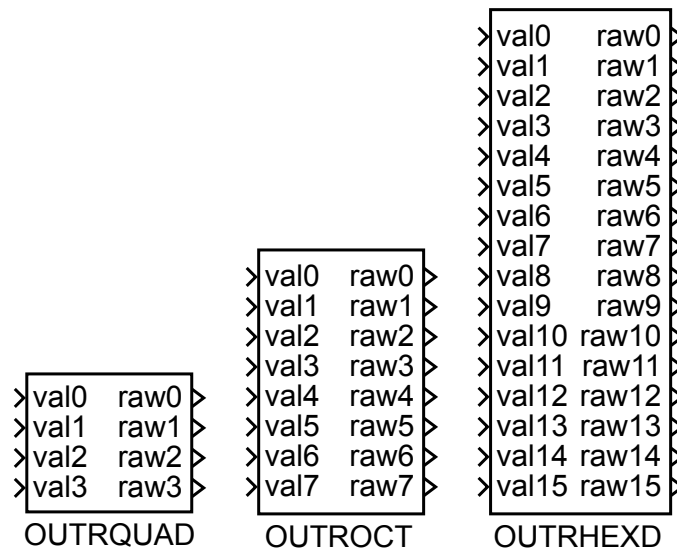
## Inputs

<code>vali</code>	Signals to be sent to the process via the input/output driver. The type and location of individual signals is described in the user manual for the given driver.	<code>unknown</code>
-------------------	--	----------------------

## 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 **raw<sub>i</sub>** output:

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

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

### Inputs

**val<sub>i</sub>**      Output signals defined by the control algorithm through the **unknown** input/output driver. The type and location of individual signals is described in the user manual for the given driver.



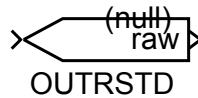
## Outputs

<code>rawi</code>	Feedback information about the write operation result. The type and meaning of individual signals is described in the user manual for the given driver.	<code>unknown</code>
-------------------	---	----------------------

## 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 D/A converter (thus the **raw** notation).
- Through reading the quality flags of the signal. This information can be separated from the signal by the [VIN](#) and [QFD](#) blocks.

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

### Input

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

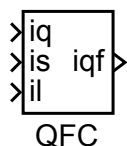
### Output

<b>raw</b>	Feedback information about the write operation result. The type and meaning of the signal is described in the user manual for the given driver.
------------	---

## QFC – Quality flags coding

Block Symbol

Licence: [ADVANCED](#)



### Function Description

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

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

### Inputs

<b>iq</b>	Basic quality type flags, see table <a href="#">1.2</a> , page <a href="#">17</a>	long
<b>is</b>	Substatus flags, see <a href="#">[1]</a>	long
<b>il</b>	Limits flags, see <a href="#">[1]</a>	long

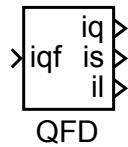
### Output

<b>iqf</b>	Bit combination of the <b>iq</b> , <b>is</b> and <b>il</b> input signals	long
------------	--	------

QFD – Quality flags decoding

Block Symbol

Licence: [ADVANCED](#)



Function Description

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

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

Input

<b>iqf</b>	Quality flags to be decomposed to <b>iq</b> , <b>is</b> and <b>il</b> components	<b>long</b>
------------	--	-------------

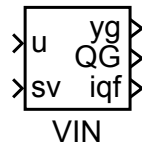
Outputs

<b>iq</b>	Basic quality type flags, see table 1.2, page 17	<b>long</b>
<b>is</b>	Substatus flags, see [1]	<b>long</b>
<b>il</b>	Limits flags, see [1]	<b>long</b>

## VIN – Validation of the input signal

### Block Symbol

Licence: [ADVANCED](#)



### 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 **RexLib** function block library for Matlab-Simulink does not use any quality flags.

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 = on** only in case of **BAD** quality.

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

### Inputs

<b>u</b>	Input signal whose quality is assessed. The type of the signal is determined upon the connected signal.	<b>unknown</b>
<b>sv</b>	Substitute value for an error case	<b>unknown</b>

### Outputs

<b>yg</b>	Validated output signal ( <b>yg = u</b> for <b>QG = on</b> or <b>yg = sv</b> for <b>QG = off</b> )	<b>unknown</b>
<b>QG</b>	Indicator of input signal acceptability	<b>bool</b>
<b>iqf</b>	Complete quality flag separated from the <b>u</b> input signal	<b>long</b>

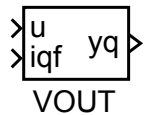
### Parameter

<b>GU</b>	Acceptability of <b>UNCERTAIN</b> quality	<b>bool</b>
	<b>off</b> ... Uncertain quality unacceptable	
	<b>on</b> .... Uncertain quality acceptable	

VOUT – Validation of the output signal

Block Symbol

Licence: [ADVANCED](#)



Function Description

It is possible to use the [VOUT](#) block to force arbitrary quality flags for a given signal. The desired quality flags are given by the input signal `iqf`. Further details about quality flags can be found in chapter [1.4](#) of this manual. The [RexLib](#) function block library for Matlab-Simulink does not use any quality flags.

Inputs

<code>u</code>	Input signal whose quality flags are being replaced. The type of the signal is determined upon the connected signal.	<code>unknown</code>
<code>iqf</code>	Desired quality flags	<code>long</code>

Output

<code>yq</code>	Resulting signal composed from input <code>u</code> and quality flags given by the <code>iqf</code> input	<code>unknown</code>
-----------------	---	----------------------

## Chapter 4

# MATH – Math blocks

### Contents

---

ABS_ – Absolute value . . . . .	69
ADD – Addition of two signals . . . . .	70
ADDQUAD, ADDOCT, ADDHEXD – Multi-input addition . . . . .	71
CNB – Boolean (logic) constant . . . . .	73
CNE – Enumeration constant . . . . .	74
CNI – Integer constant . . . . .	75
CNR – Real constant . . . . .	76
DIF_ – Difference . . . . .	77
DIV – Division of two signals . . . . .	78
EAS – Extended addition and subtraction . . . . .	79
EMD – Extended multiplication and division . . . . .	80
FNX – Evaluation of single-variable function . . . . .	81
FNXY – Evaluation of two-variables function . . . . .	83
GAIN – Multiplication by a constant . . . . .	85
GRADS – Gradient search optimization . . . . .	86
IADD – Integer addition . . . . .	88
ISUB – Integer subtraction . . . . .	90
IMUL – Integer multiplication . . . . .	92
IDIV – Integer division . . . . .	94
IMOD – Remainder after integer division . . . . .	95
LIN – Linear interpolation . . . . .	96
MUL – Multiplication of two signals . . . . .	97
POL – Polynomial evaluation . . . . .	98
REC – Reciprocal value . . . . .	99
REL – Relational operator . . . . .	100
RTOI – Real to integer number conversion . . . . .	101

SQR – Square value . . . . .	102
SQRT_ – Square root . . . . .	103
SUB – Subtraction of two signals . . . . .	104

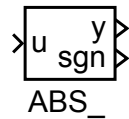
---



## 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.

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

### Input

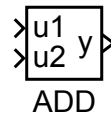
<b>u</b>	Analog input of the block	double
----------	---------------------------	--------

### Outputs

<b>y</b>	Absolute value of the input signal	double
<b>sgn</b>	Indication of the input signal sign	long

**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.

**Inputs**

<b>u1</b>	First analog input of the block	<b>double</b>
<b>u2</b>	Second analog input of the block	<b>double</b>

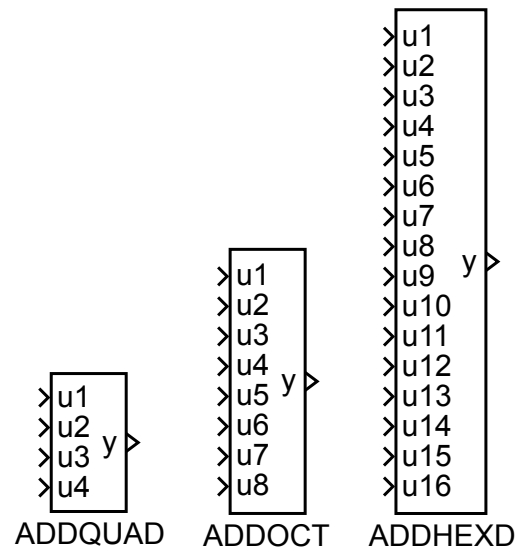
**Output**

<b>y</b>	Sum of the input signals	<b>double</b>
----------	--------------------------	---------------

## 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.

### Inputs

$u1..u16$	Analog input signals	double
-----------	----------------------	--------

### Output

$y$	Resulting value	double
-----	-----------------	--------

## Parameter

<b>n1</b>	List of signals to subtract instead of adding. The format of the list is e.g. <code>1,3..5,8</code> . 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.	<b>long</b>
-----------	--	-------------

## CNB – Boolean (logic) constant

Block Symbol

Licence: [STANDARD](#)



### Function Description

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

### Output

Y	Logical output of the block	bool
---	-----------------------------	------

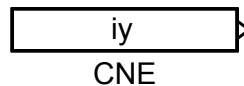
### Parameter

YCN	Boolean constant	⊙on bool
	off ... Disabled	
	on .... Enabled	

## CNE – Enumeration constant

Block Symbol

Licence: [STANDARD](#)



### Function Description

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

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

### Parameters

<b>yenum</b>	Enumeration constant	⊙1: option A	string
<b>pupstr</b>	Popup list definition	⊙1: option A 2: option B 3: option C	string

### Output

<b>iy</b>	Integer output of the block	long
-----------	-----------------------------	------

## CNI – Integer constant

Block Symbol



Licence: [STANDARD](#)

### Function Description

The **CNI** block stands for an integer constant.

### Output

<code>iy</code>	Integer output of the block	<code>long</code>
-----------------	-----------------------------	-------------------

### Parameter

<code>icn</code>	Integer constant	<code>⊙1</code> <code>long</code>
------------------	------------------	-----------------------------------

## CNR – Real constant

Block Symbol

Licence: [STANDARD](#)



### Function Description

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

### Output

y	Analog output of the block	double
---	----------------------------	--------

### Parameter

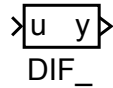
ycn	Real constant	⊕1.0 double
-----	---------------	-------------



## 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).

### Input

<b>u</b>	Analog input of the block	double
----------	---------------------------	--------

### Output

<b>y</b>	Difference of the input signal	double
----------	--------------------------------	--------

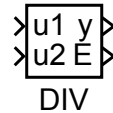
### Parameters

<b>ISSF</b>	Zero output at start-up	bool
	<div style="margin-left: 20px;"> <b>off</b> ... In the first cycle the output will be <math>y = u</math>.  <b>on</b> .... Zero output in the first cycle, <math>y = 0</math>.         </div>	

## 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  $y = yerr$ .

### Inputs

<b>u1</b>	First analog input of the block	<b>double</b>
<b>u2</b>	Second analog input of the block	<b>double</b>

### Outputs

<b>y</b>	Quotient of the inputs	<b>double</b>
<b>E</b>	Error flag – division by zero	<b>bool</b>

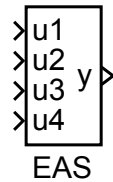
### Parameter

<b>yerr</b>	Substitute value for an error case	1.0 <b>double</b>
-------------	------------------------------------	-------------------

## 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

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

### Inputs

<b>u1</b>	First analog input of the block	double
<b>u2</b>	Second analog input of the block	double
<b>u3</b>	Third analog input of the block	double
<b>u4</b>	Fourth analog input of the block	double

### Output

<b>y</b>	Analog output of the block	double
----------	----------------------------	--------

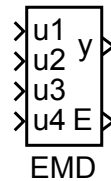
### Parameters

<b>a</b>	Weighting coefficient of the <b>u1</b> input	⊙1.0	double
<b>b</b>	Weighting coefficient of the <b>u2</b> input	⊙1.0	double
<b>c</b>	Weighting coefficient of the <b>u3</b> input	⊙1.0	double
<b>d</b>	Weighting coefficient of the <b>u4</b> input	⊙1.0	double
<b>y0</b>	Additive constant (bias)		double

## 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**.

### Inputs

<b>u1</b>	First analog input of the block	double
<b>u2</b>	Second analog input of the block	double
<b>u3</b>	Third analog input of the block	double
<b>u4</b>	Fourth analog input of the block	double

### Outputs

<b>y</b>	Analog output of the block	double
<b>E</b>	Error flag – division by zero	bool

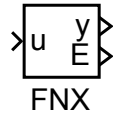
### Parameters

<b>a</b>	Weighting coefficient of the <b>u1</b> input	⊙1.0	double
<b>a0</b>	Additive constant for <b>u1</b> input		double
<b>b</b>	Weighting coefficient of the <b>u2</b> input	⊙1.0	double
<b>b0</b>	Additive constant for <b>u2</b> input		double
<b>c</b>	Weighting coefficient of the <b>u3</b> input	⊙1.0	double
<b>c0</b>	Additive constant for <b>u3</b> input		double
<b>d</b>	Weighting coefficient of the <b>u4</b> input	⊙1.0	double
<b>d0</b>	Additive constant for <b>u4</b> input		double
<b>yerr</b>	Substitute value for an error case	⊙1.0	double

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

Input

u	Analog input of the block	double
---	---------------------------	--------

Outputs

y	Result of the selected function	double
E	Error flag	bool

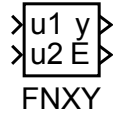
Parameters

ifn	Function type (see table above)	⊙1 long
yerr	Substitute value for an error case	double

## 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: <b>atan2</b>	arctangent $u1/u2$	–
2: <b>fmod</b>	remainder after division $u1/u2$	$u2 \neq 0.0$
3: <b>pow</b>	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**).

### Inputs

<b>u1</b>	First analog input of the block	double
<b>u2</b>	Second analog input of the block	double

### Outputs

<b>y</b>	Result of the selected function	double
<b>E</b>	Error flag	bool
	<b>off</b> ... No error	
	<b>on</b> ... An error occurred	

Parameters

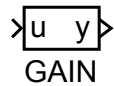
<code>ifn</code>	Function type (see the table above)	$\odot$ 1	<code>long</code>
	1 ..... <code>atan2</code>		
	2 ..... <code>fmod</code>		
	3 ..... <code>pow</code>		
<code>yerr</code>	Substitute value for an error case		<code>double</code>



## 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

$$y = ku.$$

### Input

u	Analog input of the block	double
---	---------------------------	--------

### Output

y	Analog output of the block	double
---	----------------------------	--------

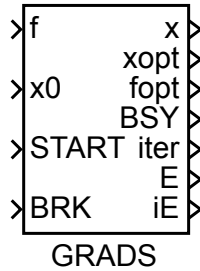
### Parameter

k	Gain	⊙1.0    double
---	------	----------------

## 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

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

### Inputs

<b>f</b>	Value of the optimized $f(\cdot)$ for given variable $\mathbf{x}$	double
<b>x0</b>	Optimization starting point	double
<b>START</b>	Starting signal (rising edge)	bool
<b>BRK</b>	Termination signal	bool

### Outputs

<b>x</b>	Current value of the optimized variable	double
<b>xopt</b>	Resulting optimal value of the $\mathbf{x}$ variable	double
<b>fopt</b>	Resulting optimal value of the function $f(\mathbf{x}, v)$	double
<b>BSY</b>	Indicator of running optimization	bool

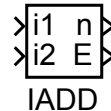
<code>iter</code>	Number of current iteration	long
<code>E</code>	Error flag	bool
<code>iE</code>	Error code	long
	1 ..... $x \notin \langle x_{\min}, x_{\max} \rangle$	
	2 ..... $x = x_{\min}$ or $x = x_{\max}$	

## Parameters

<code>xmin</code>	Lower limit for the $x$ variable		double
<code>xmax</code>	Upper limit for the $x$ variable	⊙10.0	double
<code>gamma</code>	Coefficient for determining the step size of the gradient optimization method	⊙0.3	double
<code>d0</code>	Initial step size	⊙0.05	double
<code>dmin</code>	Minimum step size	⊙0.01	double
<code>dmax</code>	Maximum step size	⊙1.0	double
<code>n</code>	Iteration period (in sampling periods $T_S$ )	⊙100	long
<code>itermax</code>	Maximum number of iterations	⊙20	long

## 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  $-32768 \dots +32767$ , 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  $30000 + 2770 = 32767$ ).

### Inputs

i1	First integer input of the block	$\downarrow -9,22E+18 \uparrow 9,22E+18$	long
i2	Second integer input of the block	$\downarrow -9,22E+18 \uparrow 9,22E+18$	long

### Outputs

n	Integer sum of the input signals	long
E	Error flag	bool
	off ... No error	
	on .... An error occurred	

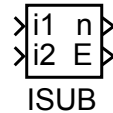
### Parameters

vtype	Numeric type	⊙4 long
	2 ..... Byte (range 0 ... 255)	
	3 ..... Short (range -32768 ... 32767)	
	4 ..... Long (range -2147483648 ... 2147483647)	
	5 ..... Word (range 0 ... 65536)	
	6 ..... DWord (range 0 ... 4294967295)	
	10 .... Large (range -9223372036854775808...9223372036854775807)	

SAT	Saturation (overflow) checking	bool
	off ... Overflow is not checked	
	on .... Overflow is checked	

## 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  $-32768 \dots +32767$ , 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  $30000 - -2770 = 32767$ ).

### Inputs

i1	First integer input of the block	$\downarrow -9,22E+18 \uparrow 9,22E+18$	long
i2	Second integer input of the block	$\downarrow -9,22E+18 \uparrow 9,22E+18$	long

### Parameters

vtype	Numeric type	⊙4	long
	2 . . . . . Byte (range 0 ... 255)		
	3 . . . . . Short (range -32768 ... 32767)		
	4 . . . . . Long (range -2147483648 ... 2147483647)		
	5 . . . . . Word (range 0 ... 65536)		
	6 . . . . . DWord (range 0 ... 4294967295)		
	10 . . . . Large (range -9223372036854775808...9223372036854775807)		
SAT	Saturation (overflow) checking		bool
	off ... Overflow is not checked		
	on ... Overflow is checked		

### Outputs

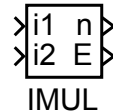
n	Integer difference between the input signals	long
---	--	------

E	Error flag	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  $-32768 \dots +32767$ , 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  $2000 * 20 = 32767$ ).

### Inputs

i1	First integer input of the block	$\downarrow -9,22E+18 \uparrow 9,22E+18$	long
i2	Second integer input of the block	$\downarrow -9,22E+18 \uparrow 9,22E+18$	long

### Parameters

vtype	Numeric type	$\odot 4$	long
	2 . . . . . Byte (range 0 ... 255)		
	3 . . . . . Short (range -32768 ... 32767)		
	4 . . . . . Long (range -2147483648 ... 2147483647)		
	5 . . . . . Word (range 0 ... 65536)		
	6 . . . . . DWord (range 0 ... 4294967295)		
	10 . . . . Large (range -9223372036854775808...9223372036854775807)		
SAT	Saturation (overflow) checking		bool
	off ... Overflow is not checked		
	on ... Overflow is checked		

### Outputs

n	Integer product of the input signals	long
---	--------------------------------------	------

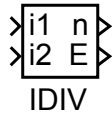


E	Error flag	bool
	off ...	No error
	on ....	An error occurred

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 **E** is set to **on** and the output **n** is substituted by  $n = \mathbf{nerr}$ .

Inputs

i1	First integer input of the block	$\downarrow -9,22E+18 \uparrow 9,22E+18$	long
i2	Second integer input of the block	$\downarrow -9,22E+18 \uparrow 9,22E+18$	long

Outputs

n	Integer quotient of the inputs	long
E	Error flag – division by zero	bool

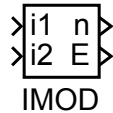
Parameters

vtype	Numeric type	$\odot 4$	long
	2 ..... Byte		
	3 ..... Short		
	4 ..... Long		
	5 ..... Word		
	6 ..... DWord		
	10 .... Large		
nerr	Substitute value for an error case	$\odot 1$	long

## 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  $n = nerr$ .

### Inputs

<b>i1</b>	First integer input of the block	$\downarrow -9,22E+18 \uparrow 9,22E+18$	long
<b>i2</b>	Second integer input of the block	$\downarrow -9,22E+18 \uparrow 9,22E+18$	long

### Outputs

<b>n</b>	Remainder after integer division	long
<b>E</b>	Error flag – division by zero	bool

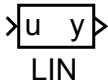
### Parameters

<b>vtype</b>	Numeric type	$\odot 4$	long
	2 ..... Byte		
	3 ..... Short		
	4 ..... Long		
	5 ..... Word		
	6 ..... DWord		
	10 ..... Large		
<b>nerr</b>	Substitute value for an error case	$\odot 1$	long

LIN – Linear interpolation

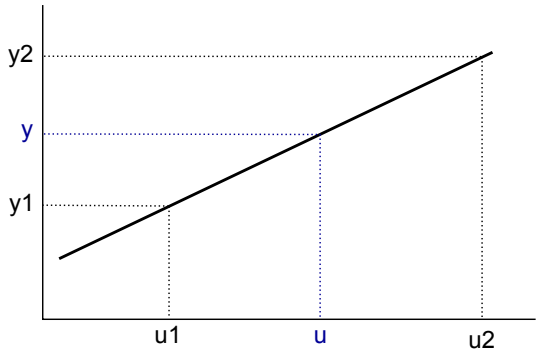
Block Symbol

Licence: [STANDARD](#)



Function Description

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



Input

$u$	Analog input of the block	double
-----	---------------------------	--------

Output

$y$	Analog output of the block	double
-----	----------------------------	--------

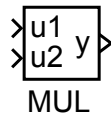
Parameters

$u_1$	x-coordinate of the 1st interpolation node	double
$y_1$	y-coordinate of the 1st interpolation node	double
$u_2$	x-coordinate of the 2nd interpolation node	⊙1.0 double
$y_2$	y-coordinate of the 2nd interpolation node	⊙1.0 double

## MUL – Multiplication of two signals

Block Symbol

Licence: [STANDARD](#)



### Function Description

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

### Inputs

u1	First analog input of the block	double
u2	Second analog input of the block	double

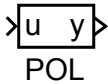
### Output

y	Product of the input signals	double
---	------------------------------	--------

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 improve the numerical robustness.

Input

u	Analog input of the block	double
---	---------------------------	--------

Output

y	Analog output of the block	double
---	----------------------------	--------

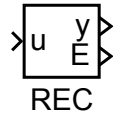
Parameters

a <i>i</i>	The <i>i</i> -th coefficient of the polynomial, <i>i</i> = 0, 1, ..., 8	double
------------	---	--------

## 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 substitutional value **y = yerr**.

### Input

<b>u</b>	Analog input of the block	<b>double</b>
----------	---------------------------	---------------

### Outputs

<b>y</b>	Analog output of the block	<b>double</b>
<b>E</b>	Error flag – division by zero	<b>bool</b>

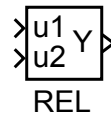
### Parameter

<b>yerr</b>	Substitute value for an error case	<b>⊙1.0   double</b>
-------------	------------------------------------	----------------------

## 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  $Y = \text{on}$  when relation holds, otherwise it is zero (relation does not hold). The binary operation codes are listed below.

### Inputs

<b>u1</b>	First analog input of the block	<b>double</b>
<b>u2</b>	Second analog input of the block	<b>double</b>

### Output

<b>Y</b>	Logical output indicating whether the relation holds	<b>bool</b>
----------	--	-------------

### Parameter

<b>irel</b>	Relation type	<b>⊙1</b>	<b>long</b>
1	..... equality (==)		
2	..... inequality (!=)		
3	..... less than (<)		
4	..... greater than (>)		
5	..... less than or equal to (<=)		
6	..... greater than or equal to (>=)		



## 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).

### Input

$r$	Analog input of the block	double
-----	---------------------------	--------

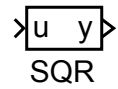
### Output

$i$	Rounded and converted input signal	long
-----	------------------------------------	------

## 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

$$y = u^2.$$

### Input

<b>u</b>	Analog input of the block	<b>double</b>
----------	---------------------------	---------------

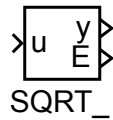
### Output

<b>y</b>	Square of the input signal	<b>double</b>
----------	----------------------------	---------------

## 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 ( $E = \text{on}$ ) and the output **y** is set to the substitute value **yerr**.

### Input

<b>u</b>	Analog input of the block	double
----------	---------------------------	--------

### Outputs

<b>y</b>	Square root of the input signal	double
<b>E</b>	Error flag	bool
	<b>off</b> ... No error	
	<b>on</b> .... Square root of negative number	

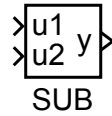
### Parameter

<b>yerr</b>	Substitute value for an error case	⊙1.0    double
-------------	------------------------------------	----------------

## 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.

### Inputs

u1	Analog input of the block	double
u2	Analog input of the block	double

### Output

y	Difference between the two input signals	double
---	--	--------

## Chapter 5

# ANALOG – Analog signal processing

### Contents

---

ABSR0T – Processing data from absolute position sensor . . . . .	107
ASW – Switch with automatic selection of input . . . . .	109
AVG – Moving average filter . . . . .	111
AVS – Motion control unit . . . . .	112
BPF – Band-pass filter . . . . .	113
CMP – Comparator with hysteresis . . . . .	114
CNDR – Nonlinear conditioner . . . . .	115
DEL – Delay with initialization . . . . .	117
DELM – Time delay . . . . .	118
DER – Derivation, filtering and prediction from the last n+1 samples	119
EVAR – Moving mean value and standard deviation . . . . .	121
INTE – Controlled integrator . . . . .	122
KDER – Derivation and filtering of the input signal . . . . .	124
LPF – Low-pass filter . . . . .	126
MINMAX – Running minimum and maximum . . . . .	127
NSCL – Nonlinear scaling factor . . . . .	128
RDFT – Running discrete Fourier transform . . . . .	129
RLIM – Rate limiter . . . . .	131
S10F2 – One of two analog signals selector . . . . .	132
SAI – Safety analog input . . . . .	135
SEL – Analog signal selector . . . . .	138
SELQUAD, SELOCT, SELHEXD – Analog signal selectors . . . . .	140
SHIFTOCT – Data shift register . . . . .	142
SHLD – Sample and hold . . . . .	144

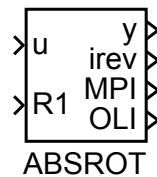
SINT – Simple integrator . . . . .	145
SPIKE – Spike filter . . . . .	146
SSW – Simple switch . . . . .	148
SWR – Selector with ramp . . . . .	149
VDEL – Variable time delay . . . . .	150
ZV4IS – Zero vibration input shaper . . . . .	151

---

## ABSROT – Processing data from absolute position sensor

### Block Symbol

Licence: [ADVANCED](#)



### Function Description

The **ABSROT** function block is intended for processing the data from absolute position sensor on rotary equipment, e.g. a shaft. The absolute sensor has a typical range of  $5^\circ$  to  $355^\circ$  (or  $-175^\circ$  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  $355^\circ$  to  $5^\circ$  in the input signal means that one revolution has been completed and the angle is in fact  $365^\circ$ .

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

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

### Inputs

<b>u</b>	Signal from the absolute position sensor	<b>double</b>
<b>R1</b>	Block reset	<b>bool</b>

### Outputs

<b>y</b>	Position output	<b>double</b>
<b>irev</b>	Number of finished revolutions	<b>long</b>
<b>MPI</b>	Mid-point indicator	<b>bool</b>
<b>OLI</b>	Off-limits indicator	<b>bool</b>

## Parameters

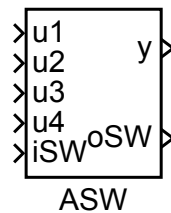
<code>lolim</code>	Lower limit of the sensor reading	$\ominus$ -3.141592654	double
<code>hilim</code>	Upper limit of the sensor reading	$\ominus$ 3.141592654	double
<code>tol</code>	Tolerance for the mid-point indicator	$\ominus$ 0.5	double
<code>hys</code>	Hysteresis for the mid-point indicator		double
<code>RESR</code>	Flag for resetting the revolutions counter		bool
	<code>off ...</code> Reset only the estimated position <code>y</code>		
	<code>on ....</code> Reset also the <code>irev</code> revolutions counter		



## 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.

### Inputs

$u1..u4$	Analog input signals to be selected from	double
$iSW$	Active signal or parameter selector	long

### Outputs

$y$	The selected analog signal or parameter	double
-----	---	--------

oSW	Identifier of the selected signal or parameter	long
-----	--	------

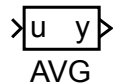
### Parameters

delta	Threshold for detecting a change	⊙0.000001	double
p1..p4	Parameters to be selected from		double

## AVG – Moving average filter

Block Symbol

Licence: [STANDARD](#)



### Function Description

The **AVG** block computes a moving average from the last **n** samples according to the formula

$$y_k = \frac{1}{n}(u_k + u_{k-1} + \cdots + u_{k-n+1}).$$

There is a limitation  $n < N$ , where  $N$  depends on the implementation.

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

### Input

<b>u</b>	Input signal to be filtered	<b>double</b>
----------	-----------------------------	---------------

### Output

<b>y</b>	Filtered output signal	<b>double</b>
----------	------------------------	---------------

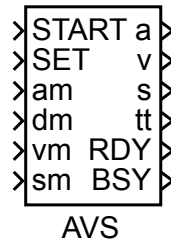
### Parameter

<b>n</b>	Number of samples to compute the average from	$\downarrow 1 \uparrow 10000000 \odot 10$	<b>long</b>
<b>nmax</b>	Limit for parameter <b>n</b> (used for internal memory allocation)	$\downarrow 10 \uparrow 10000000 \odot 100$	<b>long</b>

## 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**.

### Inputs

<b>START</b>	Starting signal (rising edge)	<b>bool</b>
<b>SET</b>	Initialize/compute the trajectory for the current inputs	<b>bool</b>
<b>am</b>	Maximal allowed acceleration [m/s <sup>2</sup> ]	<b>double</b>
<b>dm</b>	Maximal allowed deceleration [m/s <sup>2</sup> ]	<b>double</b>
<b>vm</b>	Maximum allowed velocity [m/s]	<b>double</b>
<b>sm</b>	Desired final position [m] (initial position is 0)	<b>double</b>

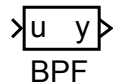
### Outputs

<b>a</b>	Acceleration [m/s <sup>2</sup> ]	<b>double</b>
<b>v</b>	Velocity [m/s]	<b>double</b>
<b>s</b>	Position [m]	<b>double</b>
<b>tt</b>	Time [s]	<b>double</b>
<b>RDY</b>	Flag indicating that the block is ready to generate the trajectory	<b>bool</b>
<b>BSY</b>	Flag indicating that the trajectory is being generated	<b>bool</b>

## BPF – Band-pass filter

Block Symbol

Licence: [STANDARD](#)



### Function Description

The BPF implements a second order filter in the form

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

where  $a$  and  $\xi$  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**.

### Input

<b>u</b>	Input signal to be filtered	double
----------	-----------------------------	--------

### Output

<b>y</b>	Filtered output signal	double
----------	------------------------	--------

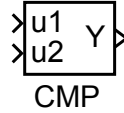
### Parameters

<b>fm</b>	Peak frequency, middle of the frequency transmission band [Hz]	double
	$\odot 1.0$	
<b>xi</b>	Relative damping coefficient (recommended value 0.5 to 1)	double
	$\odot 0.707$	
<b>ISSF</b>	Steady state at start-up flag	bool
	<b>off</b> ... Zero initial state	
	<b>on</b> ... Initial steady state	

## 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.

### Inputs

<b>u1</b>	First analog input of the block	<b>double</b>
<b>u2</b>	Second analog input of the block	<b>double</b>

### Output

<b>Y</b>	Logical output of the block	<b>bool</b>
----------	-----------------------------	-------------

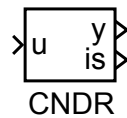
### Parameter

<b>hys</b>	Hysteresis	$\odot 0.5$ <b>double</b>
------------	------------	---------------------------

## CNDR – Nonlinear conditioner

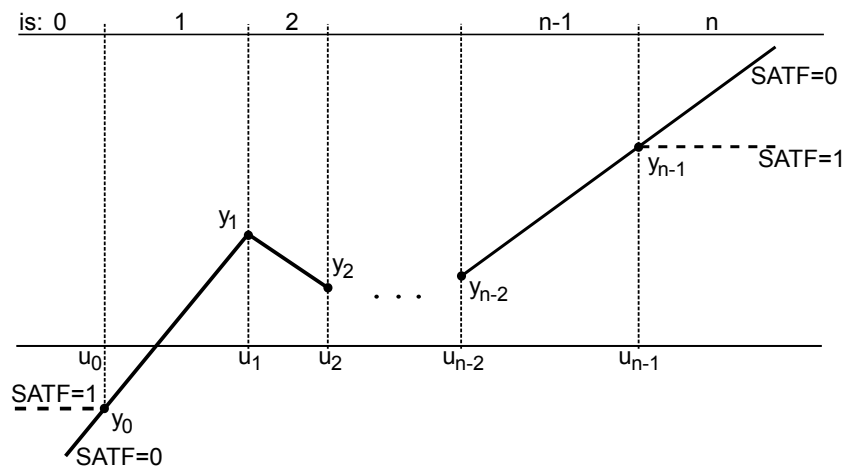
Block Symbol

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.

### Input

$u$	Analog input of the block	double
-----	---------------------------	--------

### Outputs

$y$	Analog output of the block	double
$is$	Active sector of nonlinearity (see the figure above)	long

### Parameters

$n$	Number of $(u, y)$ node pairs	$\odot 2$ long
-----	-------------------------------	----------------

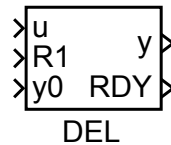
SATF	Saturation flag	⊙on	bool
	off ... Signal not limited   on .... Saturation limits active		
up	Vector of increasing $u$ -coordinates	⊙[-1 1]	double
yp	Vector of $y$ -coordinates	⊙[-1 1]	double



## 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.

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

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** = **off**.

### Inputs

<b>u</b>	Analog input of the block	double
<b>R1</b>	Block reset	bool
<b>y0</b>	Initial output value	double

### Outputs

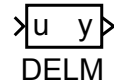
<b>y</b>	Delayed input signal	double
<b>RDY</b>	Ready flag indicating that the buffer is filled with the input signal samples	bool

### Parameter

<b>n</b>	Delay (number of samples). The resulting time delay is $n \cdot T_S$ , where $T_S$ is the block execution period.	long ↓0 ↑10000000 ⊖10
<b>nmax</b>	Limit for parameter <b>n</b> (used for internal memory allocation)	long ↓10 ↑10000000 ⊖100

**DELM – Time delay**

Block Symbol

Licence: [STANDARD](#)**Function Description**

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

**Input**

<b>u</b>	Analog input of the block	<b>double</b>
----------	---------------------------	---------------

**Output**

<b>y</b>	Delayed input signal	<b>double</b>
----------	----------------------	---------------

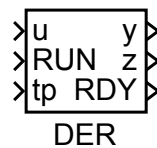
**Parameter**

<b>del</b>	Time delay [s]	⊙1.0 <b>double</b>
<b>nmax</b>	Size of delay buffer <b>del</b> (number of samples). Used for internal memory allocation.	<b>long</b> ↓10 ↑100000000 ⊙100

## 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  $y = 0$ ,  $z = u$ .

### Inputs

<b>u</b>	Analog output of the block	<b>double</b>
<b>RUN</b>	Enable execution	<b>bool</b>
	<b>off</b> ... tracking ( $z = u$ )	
	<b>on</b> ... filtering ( $y$ – estimate of the derivative, $z$ – estimate of $u$ at time $t_p$ )	
<b>tp</b>	Time instant for prediction/filtering ( $tp = 0$ corresponds with the current sampling instant)	<b>double</b>

### Outputs

<b>y</b>	Estimate of input signal derivative	<b>double</b>
<b>z</b>	Predicted/filtered input signal	<b>double</b>
<b>RDY</b>	Ready flag (all $n + 1$ samples are available)	<b>bool</b>

## Parameters

<b>n</b>	Number of samples for interpolation ( $n+1$ samples are used); $1 \leq n \leq$	<b>long</b>
	<i>nmax</i> $\downarrow 1 \uparrow 10000000 \odot 10$	
<b>nmax</b>	Limit for parameter <b>n</b> (used for internal memory allocation)	<b>long</b>
	$\downarrow 1 \uparrow 10000000 \odot 10$	

## EVAR – Moving mean value and standard deviation

Block Symbol

Licence: [STANDARD](#)



### Function Description

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

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

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

where  $k$  stands for the current sampling instant.

### Input

<b>u</b>	Analog input of the block	double
----------	---------------------------	--------

### Outputs

<b>mu</b>	Mean value of the input signal	double
<b>si</b>	Standard deviation of the input signal	double

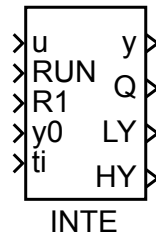
### Parameter

<b>n</b>	Number of samples to estimate the statistical properties from	long
	↓2 ↑100000000 ⊕100	
<b>nmax</b>	Limit value of parameter <b>n</b> (used for internal memory allocation)	long
	↓10 ↑100000000 ⊕200	

## 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.

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

### Inputs

<b>u</b>	Analog input of the block	double
<b>RUN</b>	Enable execution off ... Integration stopped on .... Integration running	bool
<b>R1</b>	Block reset, initialization of the integrator output to <b>y0</b>	bool
<b>y0</b>	Initial output value	double
<b>ti</b>	Integral time constant	double

### Outputs

<b>y</b>	Integrator output	double
<b>Q</b>	Running integration indicator	bool
<b>LY</b>	Lower level indicator ( $y < y_{min}$ )	bool

<code>HY</code>	Upper level indicator ( $y > y_{\max}$ )	<code>bool</code>
-----------------	--	-------------------

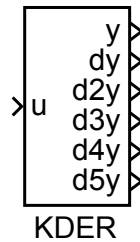
## Parameters

<code>ymin</code>	Lower level definition	$\odot -1.0$	<code>double</code>
<code>ymax</code>	Upper level definition	$\odot 1.0$	<code>double</code>

## 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  $\omega_0$ . But at the same time, the frequency spectrum of the noise should be as far away from the cutoff frequency  $\omega_0$  as possible. The cutoff frequency  $\omega_0$  and thus also the **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.

### Input

<b>u</b>	Input signal to be filtered	double
----------	-----------------------------	--------

### Outputs

<b>y</b>	Filtered input signal	double
<b>dy</b>	Estimated 1st order derivative	double
<b>d2y</b>	Estimated 2nd order derivative	double
<b>d3y</b>	Estimated 3rd order derivative	double
<b>d4y</b>	Estimated 4th order derivative	double
<b>d5y</b>	Estimated 5th order derivative	double



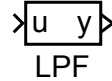
## Parameters

<code>norder</code>	Order of the derivative filter	$\downarrow 2 \uparrow 10 \odot 3$	<code>long</code>
<code>pbeta</code>	Bandwidth of the derivative filter	$\downarrow 0.0 \odot 0.1$	<code>double</code>

## 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** parameter defines the filter bandwidth and **xi** is the relative damping coefficient. The recommended value is **xi** = 0.71 for the Butterworth filter and **xi** = 0.87 for the Bessel filter.

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

### Input

<b>u</b>	Input signal to be filtered	double
----------	-----------------------------	--------

### Output

<b>y</b>	Filtered output signal	double
----------	------------------------	--------

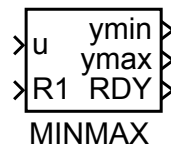
### Parameters

<b>fb</b>	Filter bandwidth [Hz]; the frequencies in the range $\langle 0, \mathbf{fb} \rangle$ pass through the filter, the attenuation at the frequency <b>fb</b> is 3 dB and approximately 40 dB at $10 \cdot \mathbf{fb}$ ; it must hold $f_b < \frac{1}{10T_s}$ for proper function of the filter, where $T_s$ is the block execution period $\odot 1.0$	double
<b>xi</b>	Relative damping coefficient (recommended value 0.5 to 1) $\odot 0.707$	double
<b>ISSF</b>	Steady state at start-up	bool
	<b>off</b> ... Zero initial state	
	<b>on</b> ... Initial steady state	

## 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.

### Inputs

<b>u</b>	Analog input of the block	double
<b>R1</b>	Block reset	bool

### Outputs

<b>ymin</b>	Minimal value found	double
<b>ymax</b>	Maximal value found	double
<b>RDY</b>	Ready flag (buffer filled)	bool

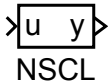
### Parameters

<b>n</b>	Number of samples for analysis (buffer length)	↓1 ↑100000000 ⊙100	long
<b>nmax</b>	Limit value of parameter <b>n</b> (used for internal memory allocation)	↓10 ↑100000000 ⊙200	long

NSCL – Nonlinear scaling factor

Block Symbol

Licence: [STANDARD](#)

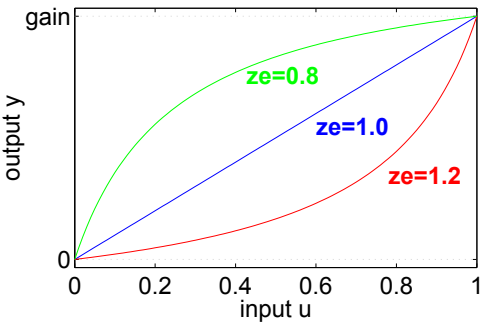


Function Description

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

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

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.



Input

u	Analog input of the block	double
---	---------------------------	--------

Output

y	Analog output of the block	double
---	----------------------------	--------

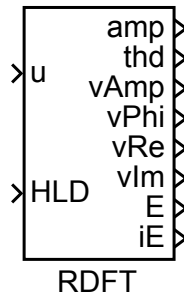
Parameters

gain	Signal gain	⊙1.0	double
ze	Shaping parameter	⊙1.0	double

## 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.

### Inputs

<b>u</b>	Analog input of the block	double
<b>HLD</b>	Hold	bool

### Outputs

<b>amp</b>	Amplitude of the fundamental frequency	double
<b>thd</b>	Total harmonic distortion (only for <b>nharm</b> ≥ 1)	double
<b>vAmp</b>	Vector of amplitudes at given frequencies	reference
<b>vPhi</b>	Vector of phase-shifts at given frequencies	reference
<b>vRe</b>	Vector of real parts at given frequencies	reference
<b>vIm</b>	Vector of imaginary parts at given frequencies	reference
<b>E</b>	Error flag	bool

iE	Error code	error
i	..... REXYGEN general error	

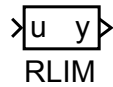
### Parameters

freq	Fundamental frequency	↓0.000000001 ↑1000000000.0 ⊙1.0	double
nper	Number of periods to calculate upon	↓1 ↑10000 ⊙10	long
nharm	Number of monitored harmonic frequencies	↓0 ↑16 ⊙3	long
ifrunit	Frequency units	↓1 ↑2 ⊙1	long
	1 ..... Hz		
	2 ..... rad/s		
iphunit	Phase shift units	↓0 ↑2 ⊙1	long
	1 ..... degrees		
	2 ..... radians		
nmax	Allocated size of array	↓10 ↑10000000 ⊙8192	long
freq2	Vector of user-defined monitored frequencies	⊙[2.0 3.0 4.0]	double

## 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**:

the steepest rise per second:  $1/\mathbf{tp}$   
 the steepest descent per second:  $-1/\mathbf{tn}$

### Input

<b>u</b>	Input signal to be filtered	double
----------	-----------------------------	--------

### Output

<b>y</b>	Filtered output signal	double
----------	------------------------	--------

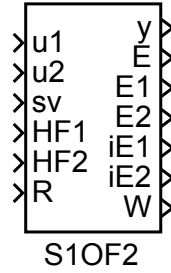
### Parameters

<b>tp</b>	Time constant defining the maximum allowed rise	⊙2.0	double
<b>tn</b>	Time constant defining the maximum allowed descent (note that <b>tn</b> > 0)	⊙2.0	double

## 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:

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

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.

## Inputs

<b>u1</b>	First analog input of the block	double
<b>u2</b>	Second analog input of the block	double
<b>sv</b>	Substitute value for an error case, i.e. <b>E</b> = <b>on</b>	double
<b>HF1</b>	Hardware error flag for signal <b>u1</b> <b>off</b> ... The input module of the signal works normally <b>on</b> .... Hardware error of the input module occurred	bool
<b>HF2</b>	Hardware error flag for signal <b>u2</b> <b>off</b> ... The input module of the signal works normally <b>on</b> .... Hardware error of the input module occurred	bool
<b>R</b>	Reset inner error flags of the input signals <b>u1</b> and <b>u2</b>	bool

## Outputs

<b>y</b>	Analog output of the block	double
<b>E</b>	Output signal invalidity indicator <b>off</b> ... Signal is valid <b>on</b> .... Signal is invalid	bool
<b>E1</b>	Invalidity indicator for input <b>u1</b> <b>off</b> ... Signal is valid <b>on</b> .... Signal is invalid, <b>y</b> = <b>u2</b>	bool
<b>E2</b>	Invalidity indicator for input <b>u2</b> <b>off</b> ... Signal is valid <b>on</b> .... Signal is invalid, <b>y</b> = <b>u1</b>	bool
<b>iE1</b>	Reason of input <b>u1</b> invalidity 0 ..... Signal valid 1 ..... Signal out of range 2 ..... Signal varies too little 3 ..... Signal varies too little and signal out of range 4 ..... Signal varies too much 5 ..... Signal varies too much 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
<b>iE2</b>	Reason of input <b>u2</b> invalidity, see the <b>iE1</b> output	long

<b>w</b>	Warning flag (invalid input signal)	<b>bool</b>
<b>off</b> ...	Both input signals <b>u1</b> and <b>u2</b> are valid	
<b>on</b> ....	At least one of the input signals is invalid	

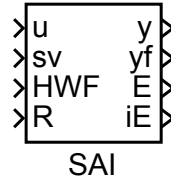
## Parameters

<b>nb</b>	Number of samples which are not included in the validity assessment of the signals <b>u1</b> and <b>u2</b> after initialization of the block	<b>long</b>
	⊙10	
<b>nc</b>	Number of samples for invariability testing (see the <b>SAI</b> block, condition F2)	<b>long</b>
	⊙10	
<b>nbits</b>	Number of A/D converter bits (source of the signals <b>u1</b> and <b>u2</b> )	<b>long</b>
	⊙12	
<b>nr</b>	Number of samples for variability testing (see the <b>SAI</b> block, condition F3)	<b>long</b>
	⊙10	
<b>prate</b>	Maximum allowed percentage change of the input <b>u1</b> ( <b>u2</b> ) within the last <b>nr</b> samples (with respect to the overall range of the input signals <b>vmax</b> – <b>vmin</b> , see the <b>SAI</b> block)	<b>double</b>
	⊙10.0	
<b>nv</b>	Number of samples for out-of-range testing (see the <b>SAI</b> block, condition F4)	<b>long</b>
	⊙1	
<b>vmin</b>	Lower limit for the input signals <b>u1</b> and <b>u2</b>	<b>double</b>
	⊙-1.0	
<b>vmax</b>	Upper limit for the input signals <b>u1</b> and <b>u2</b>	<b>double</b>
	⊙1.0	
<b>nd</b>	Number of samples for deviation testing (inner flag D; D is always <b>off</b> for <b>nd</b> = 0)	<b>long</b>
	⊙5	
<b>pdev</b>	Maximum allowed percentage deviation of the inputs <b>u1</b> and <b>u2</b> with respect to the overall range of the input signals <b>vmax</b> – <b>vmin</b>	<b>double</b>
	⊙10.0	
<b>mode</b>	Defines how to compute the output signal <b>y</b> when both input signals are valid ( <b>E1</b> = <b>off</b> , <b>E2</b> = <b>off</b> and <b>D</b> = <b>off</b> )	<b>long</b>
	⊙1	
	1 ..... Average, $y = \frac{u1+u2}{2}$	
	2 ..... Minimum, $y = \min(u1, u2)$	
	3 ..... Maximum, $y = \max(u1, u2)$	

## SAI – Safety analog input

Block Symbol

Licence: [ADVANCED](#)



### Function Description

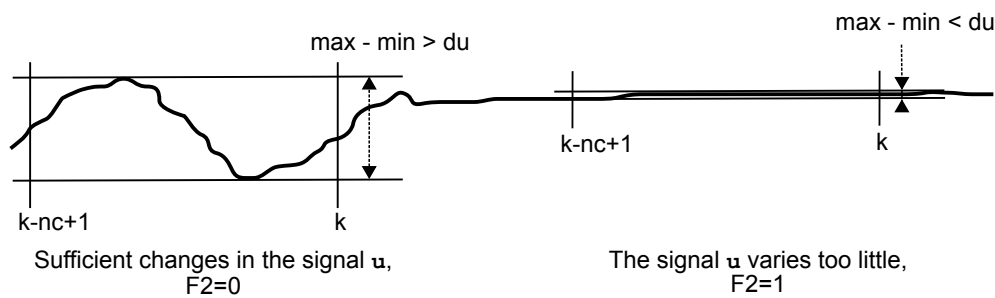
The **SAI** block tests the input signal  $u$  and assesses its validity. The input signal  $u$  is considered invalid (the output  $E = \text{on}$ ) in the following cases:

**F1:** Hardware error. The input signal  $\text{HWF} = \text{on}$ .

**F2:** The input signal  $u$  varies too little. The last  $nc$  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  $v_{\min}$  and  $v_{\max}$  are the lower and upper limits of the input  $u$ , respectively, and  $n_{\text{bits}}$  is the number of A/D converter bits. The situation when the input signal  $u$  varies too little is shown in the following picture:

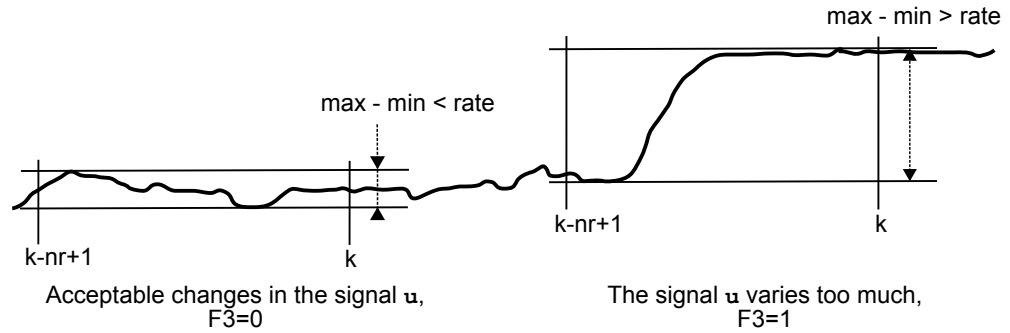


If the parameter  $nc$  is set to  $nc = 0$ , the condition **F2** is never fulfilled.

**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  $rate$ ,

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

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  $\text{nr} = 0$ , 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  $\text{nv} = 0$ , 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  $\text{R} = \text{off}$ , the output **E** is set to **on** and remains true until the reset (rising edge  $\text{R: off} \rightarrow \text{on}$ ).

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  $\text{nb} \geq 5$  to allow the **SPIKE** filter initial conditions to fade away.

## Inputs

<b>u</b>	Analog input of the block	double
<b>sv</b>	Substitute value to be used when the signal <b>u</b> is marked as invalid	double
<b>HWF</b>	Hardware error indicator	bool
	<b>off</b> ... The input module of the signal works normally	
	<b>on</b> .... Hardware error of the input module occurred	
<b>R</b>	Reset inner error flags F1–F4	bool

## Outputs

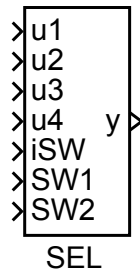
<b>y</b>	Analog output of the block	double
<b>yf</b>	Filtered analog output signal <b>y</b> , output of the <a href="#">SPIKE</a> filter	double
<b>E</b>	Output signal invalidity indicator	bool
	<b>off</b> ... Signal is valid	<b>sv</b>
	<b>on</b> .... Signal is invalid, $y = yf =$	
<b>iE</b>	Reason of invalidity	long
	0 ..... Signal valid	
	1 ..... Signal out of range	
	2 ..... Signal varies too little	
	3 ..... Signal varies too little and signal out of range	
	4 ..... Signal varies too much	
	5 ..... Signal varies too much 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	

## Parameters

<b>nb</b>	Number of samples which are not included in the validity assessment of the signal <b>u</b> after initialization of the block	$\odot 10$	long
<b>nc</b>	Number of samples for invariability testing (the F2 condition)	$\odot 10$	long
<b>nbits</b>	Number of A/D converter bits	$\odot 12$	long
<b>nr</b>	Number of samples for variability testing (the F3 condition)	$\odot 10$	long
<b>prate</b>	Maximum allowed percentage change of the input <b>u</b> within the last <b>nr</b> samples (with respect to the overall range of the input signal $v_{\max} - v_{\min}$ )	$\odot 10.0$	double
<b>nv</b>	Number of samples for out-of-range testing (the F4 condition)	$\odot 1$	long
<b>vmin</b>	Lower limit for the input signal <b>u</b>	$\odot -1.0$	double
<b>vmax</b>	Upper limit for the input signal <b>u</b>	$\odot 1.0$	double

## SEL – Analog signal selector

### Block Symbol

Licence: [STANDARD](#)

### Function Description

The **SEL** block is obsolete, replace it by the [SELQUAD](#) block. Note the difference in binary selector signals *SWn*.

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

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

### Inputs

<i>u1</i>	First analog input of the block	double
<i>u2</i>	Second analog input of the block	double
<i>u3</i>	Third analog input of the block	double
<i>u4</i>	Fourth analog input of the block	double
<i>iSW</i>	Active signal selector, active when <b>BINF</b> = off	long
<i>SW1</i>	Binary signal selector, active when <b>BINF</b> = on	bool
<i>SW2</i>	Binary signal selector, active when <b>BINF</b> = on	bool

### Output

<i>y</i>	The selected signal	double
----------	---------------------	--------

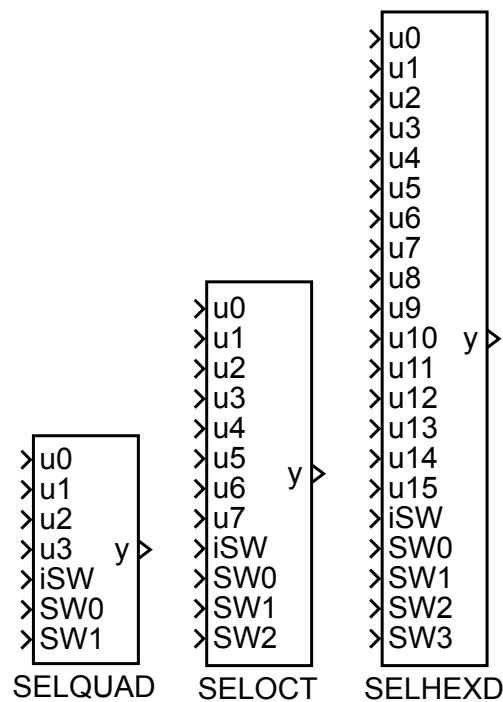
## Parameter

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

## SELQUAD, SELOCT, SELHEXD – Analog signal selectors

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**. 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

Please note that the only difference among the blocks is the number of inputs.

## Inputs

u0..15	Analog inputs of the block	double
iSW	Active signal selector	long
SW0..3	Binary signal selectors	bool

## Output

y	The selected input signal	double
---	---------------------------	--------

## Parameter

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

## 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 **y8** then have the same data type.

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

### Inputs

<b>u</b>	Data input of the register	unknown
<b>RUN</b>	Enables outputs shift	bool

### Outputs

<b>y0</b>	First output of the block	unknown
<b>y1</b>	Second output of the block	unknown
<b>y2</b>	Third output of the block	unknown
<b>y3</b>	Fourth output of the block	unknown
<b>y4</b>	Fifth output of the block	unknown

y5	Sixth output of the block	unknown
y6	Seventh output of the block	unknown
y7	Eighth output of the block	unknown

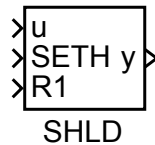
## Parameters

vtype	Output data type	⦿8	long
	1 . . . . . Bool		
	2 . . . . . Byte		
	3 . . . . . Short		
	4 . . . . . Long		
	5 . . . . . Word		
	6 . . . . . DWord		
	7 . . . . . Float		
	8 . . . . . Double		
	-- . . . . .		
	10 . . . . . Large		

## 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.

### Inputs

<b>u</b>	Analog input of the block	double
<b>SETH</b>	Trigger for the set and hold operation	bool
<b>R1</b>	Block reset, <b>R1</b> = <b>on</b> → <b>y</b> = <b>y0</b>	bool

### Output

<b>y</b>	Analog output of the block	double
----------	----------------------------	--------

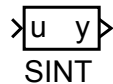
### Parameter

<b>y0</b>	Initial output value	double
<b>mode</b>	Sampling mode	⊙3 long
	1 . . . . . Triggered sampling	
	2 . . . . . Hold last value	
	3 . . . . . Hold current value	

## 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  $y_k$  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.

### Input

u	Analog input of the block	double
---	---------------------------	--------

### Output

y	Analog output of the block	double
---	----------------------------	--------

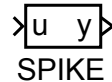
### Parameters

ti	Integral time constant $T_i$	⊙1.0	double
y0	Initial output value		double
ymax	Upper limit of the output signal	⊙1.0	double
ymin	Lower limit of the output signal	⊙-1.0	double

## 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.

### Input

$u$	Input signal to be filtered	double
-----	-----------------------------	--------

## Output

y	Filtered output signal	double
---	------------------------	--------

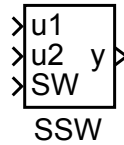
## Parameters

mingap	Minimum size of the tolerance window	⊙0.01	double
q	Tolerance window adaptation speed	↓1.0 ⊙2.0	double

## 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**).

### Inputs

<b>u1</b>	First analog input of the block	<b>double</b>
<b>u2</b>	Second analog input of the block	<b>double</b>
<b>SW</b>	Signal selector	<b>bool</b>
	<b>off</b> ... The <b>u1</b> signal is selected, $y = u1$	
	<b>on</b> .... The <b>u2</b> signal is selected, $y = u2$	

### Output

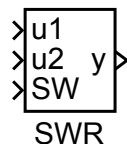
<b>y</b>	Analog output of the block	<b>double</b>
----------	----------------------------	---------------



## 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.

### Inputs

<b>u1</b>	First analog input of the block	<b>double</b>
<b>u2</b>	Second analog input of the block	<b>double</b>
<b>SW</b>	Signal selector	<b>bool</b>
	<b>off ...</b> The <b>u1</b> signal is selected	
	<b>on ....</b> The <b>u2</b> signal is selected	

### Parameters

<b>t1</b>	Rate limiter time constant for switching from <b>u2</b> to <b>u1</b>	$\odot 1.0$	<b>double</b>
<b>t2</b>	Rate limiter time constant for switching from <b>u1</b> to <b>u2</b>	$\odot 1.0$	<b>double</b>
<b>y0</b>	Initial output value to start the tracking from (before the first switching of signals occurs)		<b>double</b>

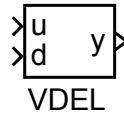
### Output

<b>y</b>	Analog output of the block	<b>double</b>
----------	----------------------------	---------------

## 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.

### Inputs

<b>u</b>	Analog input of the block	double
<b>d</b>	Time delay [s]	double

### Output

<b>y</b>	Delayed input signal	double
----------	----------------------	--------

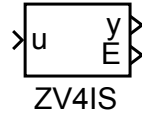
### Parameter

<b>y0</b>	Initial/substitute output value	double
<b>nmax</b>	Size of delay buffer (number of samples) for the time delay <b>d</b> . Used for internal memory allocation.	long
		↓10 ↑100000000 ⊙1000

## ZV4IS – Zero vibration input shaper

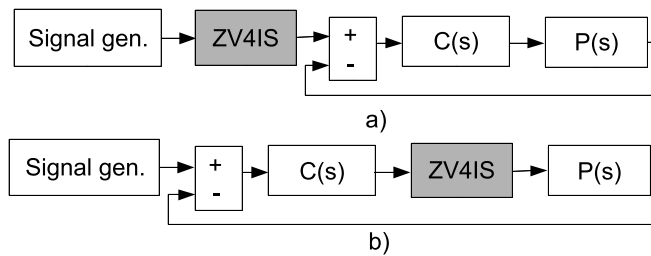
Block Symbol

Licence: [ADVANCED](#)



### Function Description

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



The input shaping filter can be used in two different ways. By using an *open loop connection*, the input reference signal for an feedback loop coming from human operator or higher level of control structure is properly shaped in order to attenuate any unwanted oscillations. The internal dynamics of the filter does not influence a behaviour of the 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.

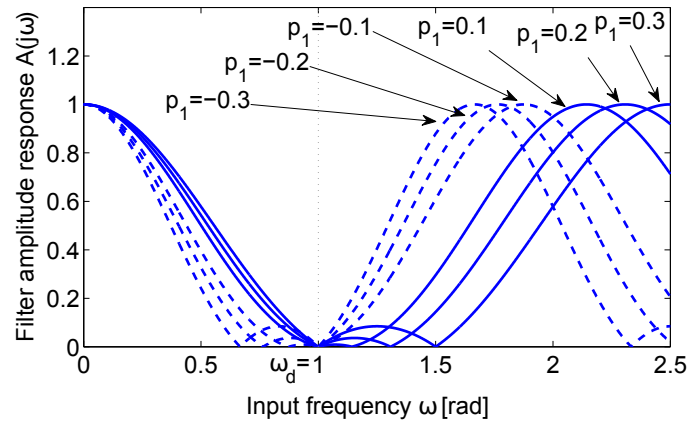
The algorithm of input shaper can be described in time domain

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

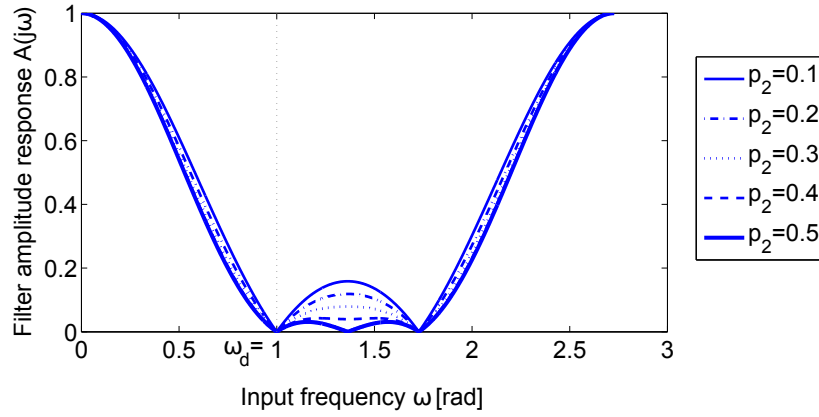
Thus, the filter has a structure of sum of weighted time delays of an input signal. The gains  $A_1..A_4$  and time delay values  $t_1..t_4$  depend on a choice of filter type, natural frequency and damping of controlled oscillatory mode of the system. The main advantage of this structure compared to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

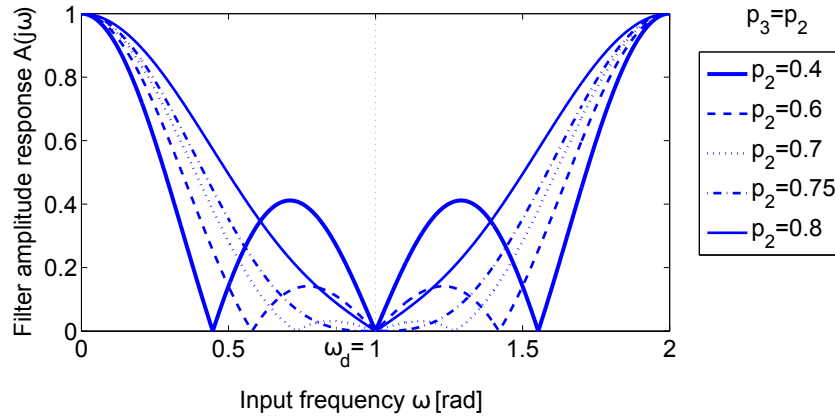
For precise tuning of the filter, complete parameterization  $\text{ipar} = 2$  can be selected. For this choice, three parameters  $p\_alpha$ ,  $p\_a2$  and  $p\_a3$  which affect the shape of the filter frequency response can freely be assigned. These parameters can be used for finding of optimal 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.

## Input

$u$

Input signal to be filtered

double

## Outputs

y	Filtered output signal	double
E	Error flag	bool
	off ... No error      on .... An error occurred	

## Parameters

omega	Natural frequency	⊙1.0	double
xi	Relative damping coefficient		double
ipar	Specification	⊙1	long
	1 ..... Basic types of IS		
	2 ..... Complete parametrization		
istype	Type	⊙2	long
	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
p_a2	Insensitivity parameter	⊙0.5	double
p_a3	Additional parameter (only for p_alpha = 0)	⊙0.5	double
nmax	Size of data buffer (number of samples). Used for internal memory allocation.		long
		↓10 ↑100000000 ⊙1000	

## Chapter 6

# GEN – Signal generators

### Contents

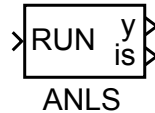
---

ANLS – Controlled generator of piecewise linear function . . . . .	156
BINS – Controlled binary sequence generator . . . . .	158
BIS – Binary sequence generator . . . . .	160
MP – Manual pulse generator . . . . .	161
PRBS – Pseudo-random binary sequence generator . . . . .	162
SG, SGI – Signal generators . . . . .	164

---

## 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.

### Input

<b>RUN</b>	Enable execution, run the analog sequence generation	<b>bool</b>
------------	--	-------------

### Outputs

<b>y</b>	Analog output of the block	<b>double</b>
<b>is</b>	Index of the active time interval	<b>long</b>

### Parameters

<b>y0</b>	Initial output value		<b>double</b>
<b>t1</b>	Node 1 time	⊙1.0	<b>double</b>
<b>y1</b>	Node 1 value		<b>double</b>
<b>t2</b>	Node 2 time	⊙1.0	<b>double</b>



y2	Node 2 value	⊙1.0	double
t3	Node 3 time	⊙2.0	double
y3	Node 3 value	⊙1.0	double
t4	Node 4 time	⊙2.0	double
y4	Node 4 value		double
RPT	Repeating sequence		bool
	off ... Disabled		
	on .... Enabled		

## 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. The difference between **BINS** and [BIS](#) blocks is that the internal timer of the **BINS** block is set to 0 and the output **Y** is set to **Y0** whenever a rising edge occurs at the **START** input (even when a binary sequence is being generated). The output value is inverted at time instants **t1**, **t2**, ..., **t8** (**off**→**on**, **on**→**off**). The last switching of the output occurs at time  $t_i$ , where  $t_{i+1} < t_i$  and the output holds its value afterwards. But for the **RPT** parameter set to **on**, instead of switching the output for the last time, the block returns to its initial state, the internal block timer is set to 0 and the binary sequence is generated repeatedly. On the contrary to the [BIS](#) block the changes in parameters **t1**...**t8** are accepted only when 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.

### Input

<b>START</b>	Starting signal (rising edge)	<b>bool</b>
--------------	-------------------------------	-------------

### Outputs

<b>Y</b>	Logical output of the block	<b>bool</b>
<b>is</b>	Index of the active time interval	<b>long</b>

### Parameters

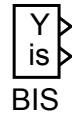
<b>Y0</b>	Initial output value	<b>bool</b>
	<b>off</b> ... Disabled/false <b>on</b> .... Enabled/true	
<b>t1</b>	Switching time 1 [s]	⊙1.0 <b>double</b>
<b>t2</b>	Switching time 2 [s]	⊙2.0 <b>double</b>
<b>t3</b>	Switching time 3 [s]	⊙3.0 <b>double</b>

t4	Switching time 4 [s]	⊙4.0	double
t5	Switching time 5 [s]	⊙5.0	double
t6	Switching time 6 [s]	⊙6.0	double
t7	Switching time 7 [s]	⊙7.0	double
t8	Switching time 8 [s]	⊙8.0	double
RPT	Repeating sequence		bool
	off ... Disabled	on .... Enabled	

## 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 output value is inverted at time instants **t1**, **t2**, ..., **t8** (**off**→**on**, **on**→**off**). The last switching of the output occurs at time  $t_i$ , where  $t_{i+1} < t_i$  and the output then holds its value. But for the **RPT** parameter set to **on**, instead of switching the output for the last time, the block returns to its initial state, the internal block timer is set to 0 and the binary sequence is generated repeatedly.

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.

### Outputs

<b>Y</b>	Logical output of the block	<b>bool</b>
<b>is</b>	Index of the active time interval	<b>long</b>

### Parameters

<b>Y0</b>	Initial output value		<b>bool</b>
	<b>off</b> ... Disabled/false	<b>on</b> .... Enabled/true	
<b>t1</b>	Switching time 1 [s]	⊙1.0	<b>double</b>
<b>t2</b>	Switching time 2 [s]	⊙2.0	<b>double</b>
<b>t3</b>	Switching time 3 [s]	⊙3.0	<b>double</b>
<b>t4</b>	Switching time 4 [s]	⊙4.0	<b>double</b>
<b>t5</b>	Switching time 5 [s]	⊙5.0	<b>double</b>
<b>t6</b>	Switching time 6 [s]	⊙6.0	<b>double</b>
<b>t7</b>	Switching time 7 [s]	⊙7.0	<b>double</b>
<b>t8</b>	Switching time 8 [s]	⊙8.0	<b>double</b>
<b>RPT</b>	Repeating sequence		<b>bool</b>
	<b>off</b> ... Disabled	<b>on</b> .... Enabled	

## 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.

### Output

Y	Logical output of the block	bool
---	-----------------------------	------

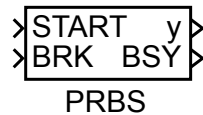
### Parameters

pwidth	Pulse width [s]	⊙1.0	double
BSTATE	Output pulse activation		bool
	off ... No action		
	on .... Generate output pulse		
RPTF	Allow pulse extension		bool
	off ... Disabled		
	on .... Enabled		

## PRBS – Pseudo-random binary sequence generator

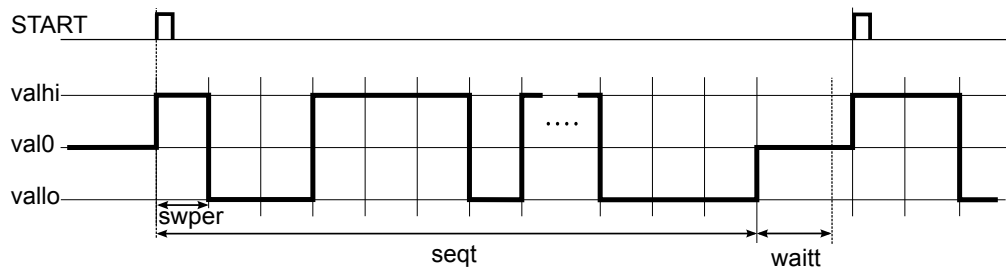
Block Symbol

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.

### Inputs

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

### Outputs

<code>y</code>	Generated pseudo-random binary sequence	<code>double</code>
<code>BSY</code>	Busy flag	<code>bool</code>

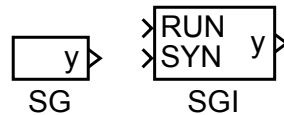
### Parameters

<code>val0</code>	Initial and final value	<code>double</code>
-------------------	-------------------------	---------------------

<b>valhi</b>	Upper level of the y output	$\odot 1.0$	double
<b>vallo</b>	Lower level of the y output	$\odot -1.0$	double
<b>swper</b>	Period of random output switching [s]	$\odot 1.0$	double
<b>swprob</b>	Probability of switching	$\downarrow 0.0 \uparrow 1.0 \odot 0.2$	double
<b>seqt</b>	Length of the sequence [s]	$\odot 10.0$	double
<b>waitt</b>	Settling period [s]	$\odot 2.0$	double

## SG, SGI – Signal generators

### Block Symbols

 Licence: [STANDARD](#)


### Function Description

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

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

### Inputs

<b>RUN</b>	Enable execution, run the binary sequence generation	<b>bool</b>
<b>SYN</b>	Synchronization signal	<b>bool</b>

### Output

<b>y</b>	Analog output of the block	<b>double</b>
----------	----------------------------	---------------

### Parameters

<b>isig</b>	Generated signal type	⊙1	<b>long</b>
	1 ..... Sine wave		
	2 ..... Symmetrical rectangular signal		
	3 ..... Sawtooth signal		
	4 ..... White noise with uniform distribution		
<b>amp</b>	Amplitude of the generated signal	⊙1.0	<b>double</b>
<b>freq</b>	Frequency of the generated signal	⊙1.0	<b>double</b>
<b>phase</b>	Phase shift of the generated signal		<b>double</b>
<b>offset</b>	Value added to the generated signal	⊙1.0	<b>double</b>
<b>ifrunit</b>	Frequency units	⊙1	<b>long</b>
	1 ..... Hz		
	2 ..... rad/s		



iphunit      Phase shift units  
               1 ..... degrees  
               2 ..... radians

⊙1    long



## Chapter 7

# REG – Function blocks for control

### Contents

---

ARLY – Advance relay . . . . .	169
FLCU – Fuzzy logic controller unit . . . . .	170
FRID – * Frequency response identification . . . . .	173
I3PM – Identification of a three parameter model . . . . .	176
LC – Lead compensator . . . . .	178
LLC – Lead-lag compensator . . . . .	179
MCU – Manual control unit . . . . .	180
PIDAT – PID controller with relay autotuner . . . . .	182
PIDE – PID controller with defined static error . . . . .	185
PIDGS – PID controller with gain scheduling . . . . .	188
PIDMA – PID controller with moment autotuner . . . . .	190
PIDU – PID controller unit . . . . .	196
PIDUI – PID controller unit with variable parameters . . . . .	199
POUT – Pulse output . . . . .	201
PRGM – Setpoint programmer . . . . .	202
PSMPC – Pulse-step model predictive controller . . . . .	204
PWM – Pulse width modulation . . . . .	208
RLY – Relay with hysteresis . . . . .	210
SAT – Saturation with variable limits . . . . .	211
SC2FA – State controller for 2nd order system with frequency autotuner . . . . .	213
SCU – Step controller with position feedback . . . . .	220
SCUV – Step controller unit with velocity input . . . . .	223
SELU – Controller selector unit . . . . .	227
SMHCC – Sliding mode heating/cooling controller . . . . .	229
SMHCCA – Sliding mode heating/cooling controller with autotuner . . . . .	233
SWU – Switch unit . . . . .	240

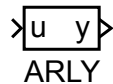
TSE – <b>Three-state element</b> . . . . .	<b>241</b>
--	------------

---

## ARLY – Advance relay

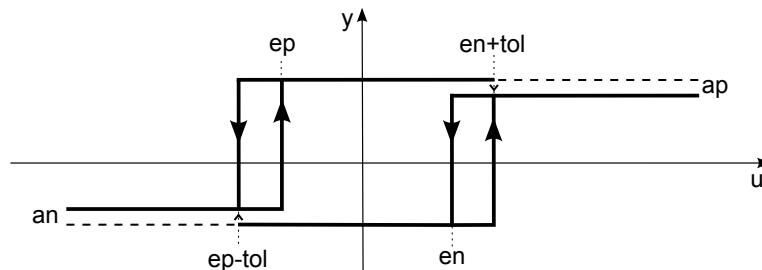
### Block Symbol

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.



### Input

$u$	Analog input of the block	double
-----	---------------------------	--------

### Output

$y$	Analog output of the block	double
-----	----------------------------	--------

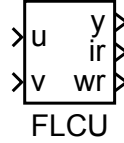
### Parameters

$ep$	Value for switching the output to the "On" state	$\odot -1.0$	double
$en$	Value for switching the output to the "Off" state	$\odot 1.0$	double
$tol$	Tolerance limit for the superposed noise of the input signal $u$	$\downarrow 0.0 \odot 0.5$	double
$ap$	Value of the $y$ output in the "On" state	$\odot 1.0$	double
$an$	Value of the $y$ output in the "Off" state	$\odot -1.0$	double
$y0$	Initial output value		double

## FLCU – Fuzzy logic controller unit

Block Symbol

Licence: [ADVANCED](#)



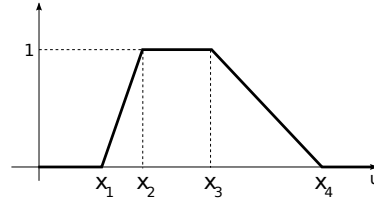
### Function Description

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

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

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

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

The set of inference rules is also a matrix whose rows are  $[i_l, j_l, k_l, w_l], l = 1, \dots, nr$ , 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.

## Inputs

<b>u</b>	First analog input of the block	double
<b>v</b>	Second analog input of the block	double

## Outputs

<b>y</b>	Analog output of the block	double
<b>ir</b>	Dominant rule	long
<b>wr</b>	Degree of truth of the dominant rule	double

## Parameters

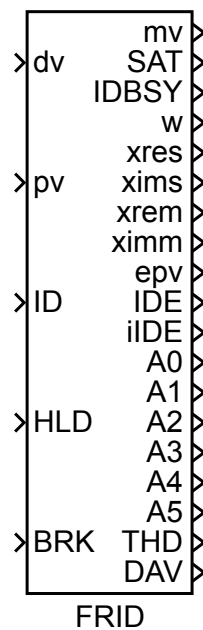
<b>umax</b>	Upper limit of the <b>u</b> input	$\odot 1.0$	double
<b>umin</b>	Lower limit of the <b>u</b> input	$\odot -1.0$	double
<b>nu</b>	Number of membership functions of the input <b>u</b>	$\downarrow 1 \uparrow 25 \odot 3$	long
<b>vmax</b>	Upper limit of the <b>v</b> input	$\odot 1.0$	double
<b>vmin</b>	Lower limit of the <b>v</b> input	$\odot -1.0$	double
<b>nv</b>	Number of membership functions of the input <b>v</b>	$\downarrow 1 \uparrow 25 \odot 3$	long
<b>ny</b>	Number of membership functions of the output <b>y</b>	$\downarrow 1 \uparrow 100 \odot 3$	long
<b>nr</b>	Number of inference rules	$\downarrow 1 \uparrow 25 \odot 3$	long
<b>mfu</b>	Matrix of membership functions of the input <b>u</b>		double
		$\odot [-1 \ -1 \ -1 \ 0; \ -1 \ 0 \ 0 \ 1; \ 0 \ 1 \ 1 \ 1]$	
<b>mfv</b>	Matrix of membership functions of the input <b>v</b>		double
		$\odot [-1 \ -1 \ -1 \ 0; \ -1 \ 0 \ 0 \ 1; \ 0 \ 1 \ 1 \ 1]$	
<b>sty</b>	Matrix of membership functions of the output <b>y</b>		double
		$\odot [-1 \ 0 \ 0; \ 0 \ 0 \ 0; \ 1 \ 0 \ 0]$	
<b>rls</b>	Matrix of inference rules	$\odot [1 \ 2 \ 3 \ 100; \ 1 \ 1 \ 1 \ 100; \ 1 \ 0 \ 3 \ 100]$	byte



## FRID — \* Frequency response identification

Block Symbol

Licence: [ADVANCED](#)



### Function Description

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

### Inputs

dv	Feedforward control variable	double
pv	Process variable	double
ID	Start the tuning experiment	bool
HLD	Hold	bool
BRK	Stop the tuning experiment	bool

### Parameters

ubias	Static component of the exciting signal	double
uamp	Amplitude of the exciting signal	⊙1.0 double
wb	Frequency interval lower limit [rad/s]	⊙1.0 double
wf	Frequency interval higher limit [rad/s]	⊙10.0 double

<b>isweep</b>	Frequency sweeping mode	⊙1	long
	1 ..... Logarithmic		
	2 ..... Linear		
<b>cp</b>	Sweeping Rate	⊙0.995	double
<b>iavg</b>	Number of values for averaging	⊙10	long
<b>obw</b>	Observer bandwidth	⊙2	long
	1 ..... LOW		
	2 ..... NORMAL		
	3 ..... HIGH		
<b>stime</b>	Settling period [s]	⊙10.0	double
<b>umax</b>	Maximum generator amplitude	⊙1.0	double
<b>thdmin</b>	Minimum demanded THD threshold	⊙0.1	double
<b>adapt_rc</b>	Maximum rate of amplitude variation	⊙0.001	double
<b>pv_max</b>	Maximum desired process value	⊙1.0	double
<b>pv_sat</b>	Maximum allowed process value	⊙2.0	double
<b>ADAPT_EN</b>	Enable automatic amplitude adaptation	⊙on	bool
<b>immode</b>	Mesurement mode	⊙1	long
	1 ..... Manual specification of frequency points		
	2 ..... Linear series of nmw points in the interval <wb;wf>		
	3 ..... Logarithmic series of nmw points in the interval <wb;wf>		
	4 ..... Automatic detection of important frequencies (N/A)		
<b>nwm</b>	Number of frequency response point for automatic mode		long
<b>wm</b>	Frequency measurement points for manual meas. mode [array of rad/s]	⊙[2.0 4.0 6.0 8.0]	double

## Outputs

<b>mv</b>	Manipulated variable (controller output)	double
<b>SAT</b>	Saturation flag	bool
<b>IDBSY</b>	Tuner busy flag	bool
<b>w</b>	Actual frequency [rad/s]	double
<b>xres</b>	real part of frequency response (sweeping)	double
<b>xims</b>	imaginary part of frequency response (sweeping)	double
<b>xrem</b>	real part of frequency response (measurement)	double
<b>ximm</b>	imaginary part of frequency response (measurement)	double
<b>epv</b>	Estimated process value	double
<b>IDE</b>	Error indicator	bool
<b>iIDE</b>	Error code	long
<b>A0</b>	Estimated DC value	double
<b>A1</b>	Estimated 1st harmonics amlitude	double
<b>A2</b>	Estimated 2nd harmonics amlitude	double
<b>A3</b>	Estimated 3rd harmonics amlitude	double
<b>A4</b>	Estimated 4th harmonics amlitude	double
<b>A5</b>	Estimated 5th harmonics amlitude	double
<b>THD</b>	Total harmonic distorsion	double

DAV

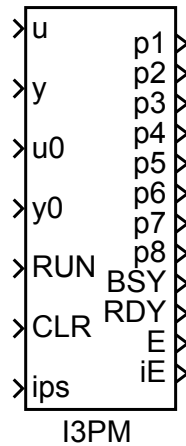
Data Valid

bool

## I3PM – Identification of a three parameter model

Block Symbol

Licence: [ADVANCED](#)



### Function Description

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

### Inputs

u	Input of the identified system	double
y	Output of the identified system	double
u0	Input steady state	double
y0	Output steady state	double
RUN	Execute identification	bool
CLR	Block reset	bool
ips	Meaning of the output signals	long
	0 ..... FOPDT model	
	p1 ... gain	
	p2 ... time delay	
	p3 ... time constant	
	1 ..... moments of input and output	
	p1 ... parameter $mu0$	
	p2 ... parameter $mu1$	
	p3 ... parameter $mu2$	
	p4 ... parameter $my0$	
	p5 ... parameter $my1$	
	p6 ... parameter $my2$	

2 ..... process moments  
     p1 ... parameter  $mp0$   
     p2 ... parameter  $mp1$   
     p3 ... parameter  $mp2$   
 3 ..... characteristic numbers  
     p1 ... parameter  $\kappa$   
     p2 ... parameter  $\mu$   
     p3 ... parameter  $\sigma^2$   
     p4 ... parameter  $\sigma$

## Outputs

p <i>i</i>	Identified parameters with respect to <b>ips</b> , $i = 1, \dots, 8$	double
BSY	Busy flag	bool
RDY	Ready flag	bool
E	Error flag	bool
iE	Error code	long
1 .....	Premature termination ( <b>RUN = off</b> )	
2 .....	$\mu0 = 0$	
3 .....	$mp0 = 0$	
4 .....	$\sigma^2 < 0$	

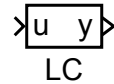
## Parameters

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

## LC – Lead compensator

Block Symbol

Licence: [STANDARD](#)



### Function Description

The LC block is a discrete simulator of derivative element

$$C(s) = \frac{\text{td} * s}{\frac{\text{td}}{\text{nd}} * s + 1},$$

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

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

### Input

<b>u</b>	Analog input of the block	<b>double</b>
----------	---------------------------	---------------

### Output

<b>y</b>	Analog output of the block	<b>double</b>
----------	----------------------------	---------------

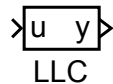
### Parameters

<b>td</b>	Derivative time constant	⊙1.0	<b>double</b>
<b>nd</b>	Derivative filtering parameter	⊙10.0	<b>double</b>
<b>ISSF</b>	Steady state at start-up		<b>bool</b>
	<b>off</b> ... Zero initial state		
	<b>on</b> .... Initial steady state		

## LLC – Lead-lag compensator

Block Symbol

Licence: [STANDARD](#)



### Function Description

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

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

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**.

This block is ideal for simulation of first order plus dead time systems (FOPDT). Just set the **a** parameter to zero.

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

### Input

<b>u</b>	Analog input of the block	double
----------	---------------------------	--------

### Output

<b>y</b>	Analog output of the block	double
----------	----------------------------	--------

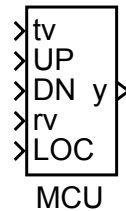
### Parameters

<b>tau</b>	Time constant	⊙1.0	double
<b>a</b>	Numerator time constant coefficient		double
<b>ISSF</b>	Steady state at start-up		bool
	<b>off</b> ... Zero initial state		
	<b>on</b> .... Initial steady state		

## MCU – Manual control unit

Block Symbol

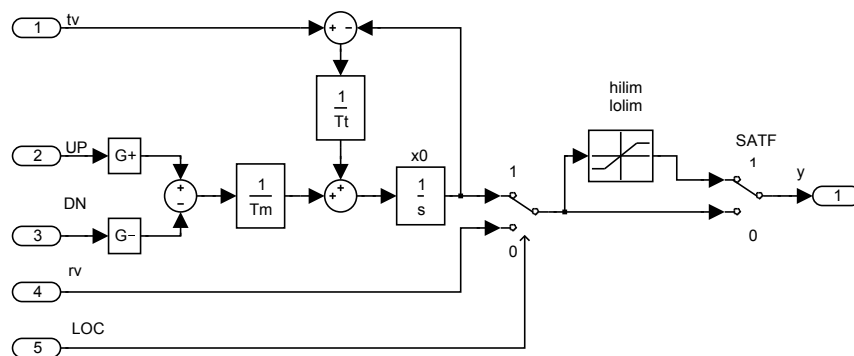
Licence: [STANDARD](#)



### Function Description

The MCU block is suitable for manual setting of the numerical output value  $y$ , e.g. a setpoint. In the local mode ( $LOC = on$ ) the value is set using the buttons  $UP$  and  $DN$ . The rate of increasing/decreasing of the output  $y$  from the initial value  $y_0$  is determined by the integration time constant  $t_m$  and pushing time of the buttons. After 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 ( $SATF = on$ ) by saturation limits  $lolim$  and  $hilim$ . If none of the buttons is pushed ( $UP = off$  and  $DN = off$ ), the output  $y$  tracks the input value  $tv$ . The tracking speed is controlled by the integration time constant  $t_t$ .

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



### Inputs

$tv$	Tracking variable	double
$UP$	The "up" signal	bool



DN	The "down" signal	bool
rv	Remote output value in the mode LOC = off	double
LOC	Local or remote mode	bool

## Output

y	Analog output of the block	double
---	----------------------------	--------

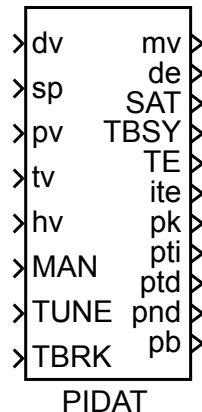
## Parameters

tt	Tracking time constant of the input tv	⊙1.0	double
tm	Initial value of integration time constant	⊙100.0	double
y0	Initial output value		double
q	Multiplication quotient	⊙5.0	double
ta	Interval after which the rate is changed [s]	⊙4.0	double
tf	Interval after which the rate changes no more [s]	⊙8.0	double
SATF	Saturation flag		bool
	off ... Signal not limited		
	on .... Saturation limits active		
hilim	Upper limit of the output signal	⊙1.0	double
lolim	Lower limit of the output signal	⊙-1.0	double

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

### Inputs

<b>dv</b>	Feedforward control variable	double
<b>sp</b>	Setpoint variable	double

pv	Process variable	double
tv	Tracking variable	double
hv	Manual value	double
MAN	Manual or automatic mode	bool
	off ... Automatic mode	
	on .... Manual mode	
TUNE	Start the tuning experiment	bool
TBRK	Stop the tuning experiment	bool

## Outputs

mv	Manipulated variable (controller output)	double
de	Deviation error	double
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; expected time (in seconds) to finishing the tuning experiment while the tuning experiment is active	long
	1000 .. Signal/noise ratio too low	
	1001 .. Hysteresis too high	
	1002 .. Too tight termination rule	
	1003 .. Phase out of interval	
pk	Proposed controller gain	double
pti	Proposed integral time constant	double
ptd	Proposed derivative time constant	double
pnd	Proposed derivative component filtering	double
pb	Proposed weighting factor – proportional component	double

## Parameters

irtype	Controller type (control law)	⊙6	long
	1 ..... D            4 ..... P            7 ..... PID		
	2 ..... I            5 ..... PD		
	3 ..... ID          6 ..... PI		
RACT	Reverse action flag		bool
	off ... Higher mv → higher pv		
	on .... Higher mv → lower pv		
k	Controller gain $K$	⊙1.0	double
ti	Integral time constant $T_i$	⊙4.0	double
td	Derivative time constant $T_d$	⊙1.0	double
nd	Derivative filtering parameter $N$	⊙10.0	double
b	Setpoint weighting – proportional part	⊙1.0	double

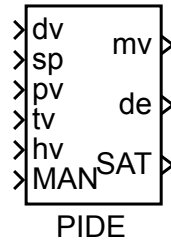
<b>c</b>	Setpoint weighting – derivative part		<b>double</b>
<b>tt</b>	Tracking time constant. No meaning for controllers without integrator.	$\odot 1.0$	<b>double</b>
<b>hilim</b>	Upper limit of the controller output	$\odot 1.0$	<b>double</b>
<b>lolim</b>	Lower limit of the controller output	$\odot -1.0$	<b>double</b>
<b>iainf</b>	Type of apriori information	$\odot 1$	<b>long</b>
	1 ..... No apriori information		
	2 ..... Astatic process (process with integration)		
	3 ..... Low order process		
	4 ..... Static process + slow closed loop step response		
	5 ..... Static process + middle fast (normal) closed loop step response		
	6 ..... Static process + fast closed loop step response		
<b>k0</b>	Static gain of the process (must be provided in case of <b>iainf</b> = 3, 4, 5)	$\odot 1.0$	<b>double</b>
<b>n1</b>	Maximum number of half-periods for estimation of frequency response point	$\odot 20$	<b>long</b>
<b>mm</b>	Maximum number of half-periods for averaging	$\odot 4$	<b>long</b>
<b>amp</b>	Relay controller amplitude	$\odot 0.1$	<b>double</b>
<b>uhys</b>	Relay controller hysteresis		<b>double</b>
<b>ntime</b>	Length of noise amplitude estimation period at the beginning of the tuning experiment [s]	$\odot 5.0$	<b>double</b>
<b>rerrap</b>	Termination value of the oscillation amplitude relative error	$\odot 0.1$	<b>double</b>
<b>aerrph</b>	Termination value of the absolute error in oscillation phase	$\odot 10.0$	<b>double</b>
<b>maxdev</b>	Maximal admissible deviation error from the initial steady state	$\odot 1.0$	<b>double</b>

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

## PIDE – PID controller with defined static error

Block Symbol

Licence: [ADVANCED](#)



### Function Description

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

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

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

where

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

$U(s)$  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  $K$ ,  $T_i$ ,  $T_d$ ,  $N$ ,  $\varepsilon$  ( $= b_p/100$ ),  $b$  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** of the controller to the controller input **tv** and properly setting the tracking time constant **tt** we obtain the bumpless operation of the controller in the case of the mode switching (manual, automatic) and also the correct operation of the controller when saturation of the output **mv** occurs (antiwindup).

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  $\varepsilon > 0$ .

## Inputs

dv	Feedforward control variable	double
sp	Setpoint variable	double
pv	Process variable	double
tv	Tracking variable	double
hv	Manual value	double
MAN	Manual or automatic mode	bool
	off ... Automatic mode	
	on .... Manual mode	

## Outputs

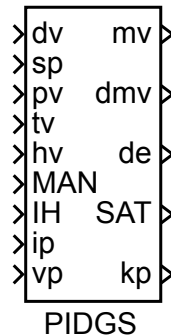
mv	Manipulated variable (controller output)	double
de	Deviation error	double
SAT	Saturation flag	bool
	off ... The controller implements a linear control law	
	on .... The controller output is saturated	

## Parameters

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

## 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  $thrsha(1) < vp < thrsha(2)$  etc. till the set 5 for  $thrsha(5) < vp$ . The index of the active parameter set is available at the **kp** output.

### Inputs

<b>dv</b>	Feedforward control variable	double
<b>sp</b>	Setpoint variable	double
<b>pv</b>	Process variable	double
<b>tv</b>	Tracking variable	double
<b>hv</b>	Manual value	double
<b>MAN</b>	Manual or automatic mode	bool
	<b>off</b> ... Automatic mode	
	<b>on</b> .... Manual mode	
<b>IH</b>	Integrator hold	bool
	<b>off</b> ... Integration enabled	
	<b>on</b> .... Integration disabled	
<b>ip</b>	Parameter set index	↓0 ↑5 long
<b>vp</b>	Switching analog signal	double

### Outputs

<b>mv</b>	Manipulated variable (controller output)	double
-----------	--	--------



dmv	Controller velocity output (difference)	double
de	Deviation error	double
SAT	Saturation flag	bool
	off ... The controller implements a linear control law	
	on .... The controller output is saturated	
kp	Active parameter set index	long

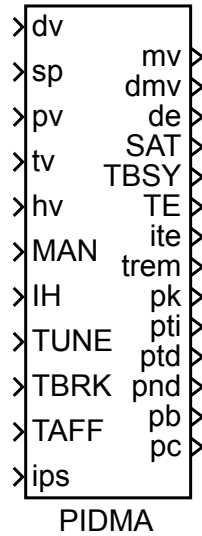
## Parameters

hilim	Upper limit of the controller output	⊙1.0	double
lolim	Lower limit of the controller output	⊙-1.0	double
dz	Dead zone		double
icotype	Controller output type	⊙1	long
	1 ..... Analog output		
	2 ..... Pulse width modulation (PWM)		
	3 ..... Step controller unit with position feedback (SCU)		
	4 ..... Step controller unit without position feedback (SCUV)		
npars	Number of controller parameter sets	⊙6	long
GSCF	Switch parameters by analog signal vp		bool
	off ... Index-based switching		
	on .... Analog signal based switching		
hys	Hysteresis for controller parameters switching		double
irtypea	Vector of controller types (control laws)	⊙[6 6 6 6 6 6]	byte
	1 ..... D            4 ..... P            7 ..... PID		
	2 ..... I            5 ..... PD		
	3 ..... ID          6 ..... PI		
RACTA	Vector of reverse action flags	⊙[0 0 0 0 0 0]	bool
	0 ..... Higher mv → higher pv		
	1 ..... Higher mv → lower pv		
ka	Vector of controller gains $K$	⊙[1.0 1.0 1.0 1.0 1.0 1.0]	double
tia	Vector of integral time constants $T_i$	⊙[4.0 4.0 4.0 4.0 4.0 4.0]	double
tda	Vector of derivative time constants $T_d$	⊙[1.0 1.0 1.0 1.0 1.0 1.0]	double
nda	Vector of derivative filtering parameters $N$	⊙[10.0 10.0 10.0 10.0 10.0 10.0]	double
ba	Setpoint weighting factors – proportional part	⊙[1.0 1.0 1.0 1.0 1.0 1.0]	double
ca	Setpoint weighting factors – derivative part	⊙[0.0 0.0 0.0 0.0 0.0 0.0]	double
tta	Vector of tracking time constants. No meaning for controllers without integrator.	⊙[1.0 1.0 1.0 1.0 1.0 1.0]	double
thrsha	Vector of thresholds for switching the parameters	⊙[0.1 0.2 0.3 0.4 0.5 0]	double

## PIDMA – PID controller with moment autotuner

Block Symbol

Licence: [AUTOTUNING](#)



### Function Description

The PIDMA block has the same control function as the [PIDU](#) block. Additionally it is equipped with the moment autotuning function.

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

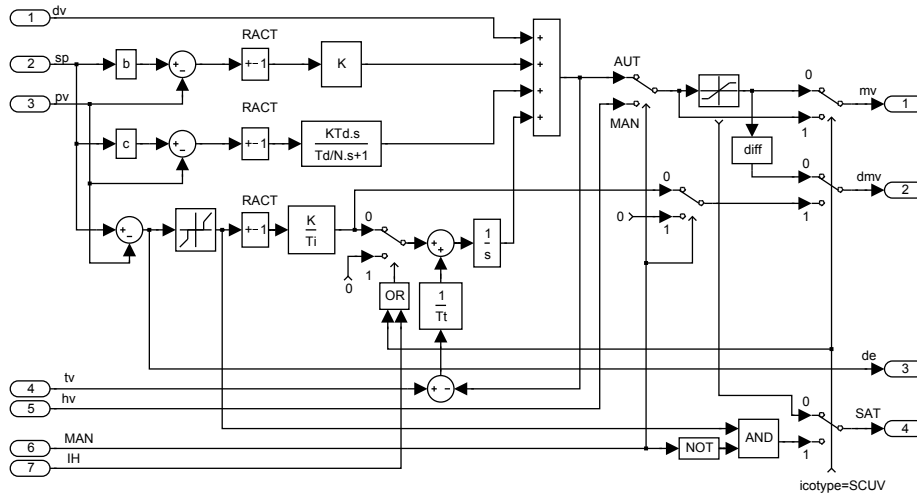
$$U(s) = \pm K \left\{ bW(s) - Y(s) + \frac{1}{T_i s} [W(s) - Y(s)] + \frac{T_d s}{\frac{T_d}{N}s + 1} [cW(s) - Y(s)] \right\} + Z(s)$$

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** = **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 rule of thumb is  $tt \approx \sqrt{T_i T_d}$  or  $tt \approx 2 \cdot \sqrt{T_i}$  in the case of a PI controller). In this way we obtain the bumpless operation of the controller in the case of the mode switching (manual, automatic) and also the correct operation of the controller in the saturation of the output

**mv** (antiwindup). The additional outputs **dmv**, **de** and **SAT** generate the velocity output (difference of **mv**), deviation error and saturation flag, respectively.

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

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

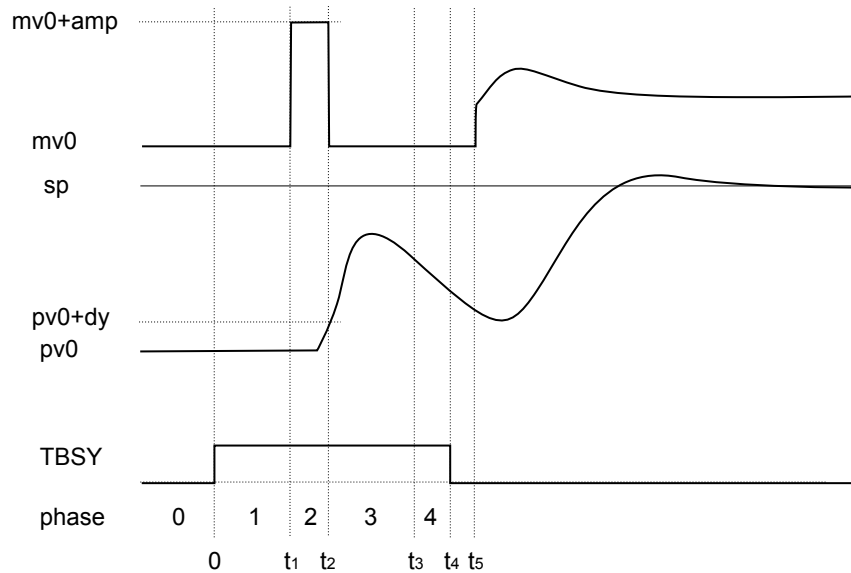


The block **PIDMA** extends the control function of the standard PID controller by the built in autotuning feature. Before start of the autotuner the operator have to reach the steady state of the process at a suitable working point (in manual or automatic mode) and specify the required type of the controller **itttype** (PI or PID) and other tuning parameters (**iainf**, **DGC**, **tdg**, **tn**, **amp**, **dy** and **ispeed**). The identification experiment is started by 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 identification experiment is properly finished (**TE = off**) and the input **ips** is equal to zero, then the optimal parameters immediately appear on the block outputs **pk**, **pti**, **ptd**, **pnd**, **pb**, **pc**. In the opposite case (**TE = on**) the output **ite** specifies the

experiment error more closely. Other values of the `ips` input are reserved for custom specific purposes.

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`.

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.

## Inputs

<code>dv</code>	Feedforward control variable	double
<code>sp</code>	Setpoint variable	double
<code>pv</code>	Process variable	double
<code>tv</code>	Tracking variable	double
<code>hv</code>	Manual value	double
<code>MAN</code>	Manual or automatic mode	bool
	<code>off</code> ... Automatic mode	
	<code>on</code> .... Manual mode	
<code>IH</code>	Integrator hold	bool
	<code>off</code> ... Integration enabled	
	<code>on</code> .... Integration disabled	
<code>TUNE</code>	Start the tuning experiment ( <code>off→on</code> ) or force transition to the next tuning phase (see the description of the <code>ite</code> output)	bool
<code>TBRK</code>	Stop the tuning experiment	bool

TAFF	Tuning affirmation; determines the way the computed parameters are handled	bool
	off ... Parameters are only computed	
	on ... Parameters are set into the control law	
ips	Meaning of the output signals <b>pk</b> , <b>pti</b> , <b>ptd</b> , <b>pnd</b> , <b>pb</b> and <b>pc</b>	long
	0 ..... Designed parameters <b>k</b> , <b>ti</b> , <b>td</b> , <b>nd</b> , <b>b</b> and <b>c</b> of the PID control law	
	1 ..... Process moments: static gain ( <b>pk</b> ), resident time constant ( <b>pti</b> ), measure of the system response length ( <b>ptd</b> )	
	2 ..... Three-parameter first-order plus dead-time model: static gain ( <b>pk</b> ), dead-time ( <b>pti</b> ), time constant ( <b>ptd</b> )	
	3 ..... Three-parameter second-order plus dead-time model with double time constant: static gain ( <b>pk</b> ), dead-time ( <b>pti</b> ), time constant ( <b>ptd</b> )	
	4 ..... Estimated boundaries for manual fine-tuning of the PID controller ( <b>irtype</b> = 7) gain <b>k</b> : upper boundary <b>k<sub>hi</sub></b> ( <b>pk</b> ), lower boundary <b>k<sub>lo</sub></b> ( <b>pti</b> )	
	>99 ... Reserved for diagnostic purposes	

## Outputs

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

*Tuning phases codes (during the experiment):*

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

*Remark about terminating the tuning phases*

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

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

## Parameters

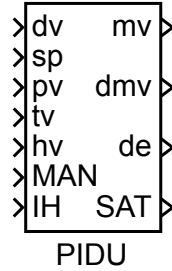
irtype	Controller type (control law)	⊙6	long
	1 ..... D      4 ..... P      7 ..... PID		
	2 ..... I      5 ..... PD		
	3 ..... ID      6 ..... PI		
RACT	Reverse action flag		bool
	off ... Higher mv → higher pv		
	on ... Higher mv → lower pv		
k	Controller gain $K$	⊙1.0	double
ti	Integral time constant $T_i$	⊙4.0	double
td	Derivative time constant $T_d$	⊙1.0	double
nd	Derivative filtering parameter $N$	⊙10.0	double
b	Setpoint weighting – proportional part	⊙1.0	double
c	Setpoint weighting – derivative part		double
tt	Tracking time constant. No meaning for controllers without integrator.	⊙1.0	double
hilim	Upper limit of the controller output	⊙1.0	double
lolim	Lower limit of the controller output	⊙-1.0	double
dz	Dead zone		double
icotype	Controller output type	⊙1	long
	1 ..... Analog output		
	2 ..... Pulse width modulation (PWM)		
	3 ..... Step controller unit with position feedback (SCU)		
	4 ..... Step controller unit without position feedback (SCUV)		

<b>itttype</b>	Controller type to be designed	⊙6	long
	6 ..... PI controller		
	7 ..... PID controller		
<b>iainf</b>	Type of apriori information	⊙1	long
	1 ..... Static process		
	2 ..... Astatic process		
<b>DGC</b>	Drift gradient compensation	⊙on	bool
	off ... Disabled		
	on .... Enabled		
<b>tdg</b>	Drift gradient estimation time [s]	⊙60.0	double
<b>tn</b>	Length of noise estimation period [s]	⊙5.0	double
<b>amp</b>	Tuning pulse amplitude	⊙0.5	double
<b>dy</b>	Tuning pulse get down threshold (absolute difference from the steady pv value)	↓0.0 ⊙0.1	double
<b>ispeed</b>	Desired closed loop speed	⊙2	long
	1 ..... Slow closed loop		
	2 ..... Normal (middle fast) closed loop		
	3 ..... Fast closed loop		
<b>ipid</b>	PID controller form	⊙1	long
	1 ..... Parallel form		
	2 ..... Series form		

## PIDU – PID controller unit

Block Symbol

Licence: [STANDARD](#)



### Function Description

The **PIDU** block is a basic block for creating a complete PID controller (or P, I, PI, PD, PID, PI+S). In the most simple case it works as a standalone unit with the standard PID controller functionality with two degrees of freedom. It can operate in automatic mode (**MAN** = **off**) or manual mode (**MAN** = **on**).

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

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

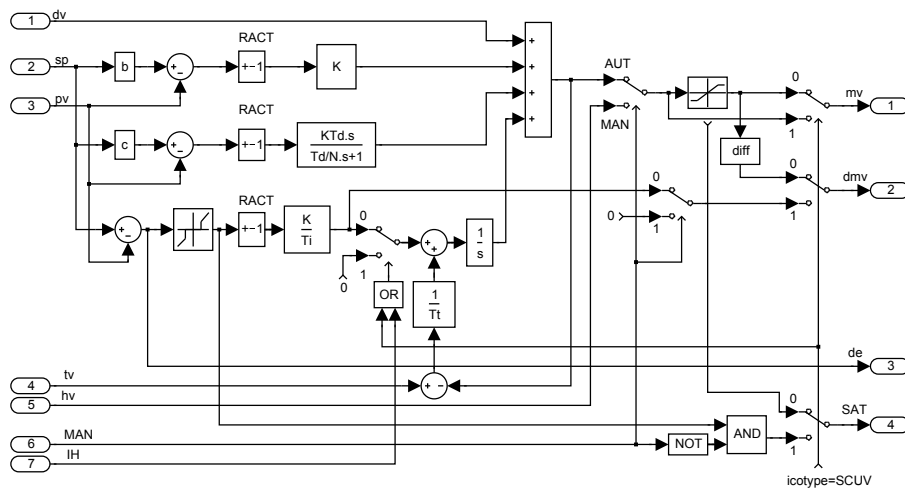
where  $U(s)$  is Laplace transform of the manipulated variable **mv**,  $W(s)$  is Laplace transform of the setpoint variable **sp**,  $Y(s)$  is Laplace transform of the process variable **pv**,  $Z(s)$  is Laplace transform of the feedforward control variable **dv** and  $K$ ,  $T_i$ ,  $T_d$ ,  $N$ ,  $b$  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 rule of thumb is  $tt \approx \sqrt{T_i T_d}$  or  $tt \approx 2 \cdot \sqrt{T_i}$  in the case of a PI controller). In this way we obtain the bumpless operation of the controller in the case of the mode switching (manual, automatic) and also the correct operation of the controller when saturation of the output **mv** occurs (antiwindup). The additional outputs **dmv**, **de** and **SAT** generate the velocity output (difference of **mv**), deviation error and saturation flag, respectively.

If the **PIDU** block is connected with the **SCUV** block to configure the 3-point step controller without the positional feedback, then the parameter **icotype** must be set to 4 and the meaning of the outputs **mv** and **dmv** and **SAT** is modified in the following way:



$mv$  and  $dmv$  give the PD part and difference of I part of the control law, respectively, and  $SAT$  provides the information for the **SCUV** block whether the deviation error is less than the dead zone  $dz$  in the automatic mode. In this case, the setpoint weighting factor  $c$  should be zero.

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



## Inputs

$dv$	Feedforward control variable	double
$sp$	Setpoint variable	double
$pv$	Process variable	double
$tv$	Tracking variable	double
$hv$	Manual value	double
$MAN$	Manual or automatic mode off ... Automatic mode on .... Manual mode	bool
$IH$	Integrator hold off ... Integration enabled on .... Integration disabled	bool

## Outputs

$mv$	Manipulated variable (controller output)	double
$dmv$	Controller velocity output (difference)	double
$de$	Deviation error	double
$SAT$	Saturation flag off ... The controller implements a linear control law on .... The controller output is saturated	bool

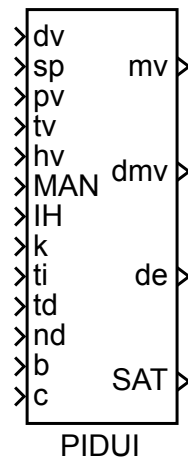
## Parameters

irtype	Controller type (control law)	⊙6	long
	1 ..... D            4 ..... P            7 ..... PID		
	2 ..... I            5 ..... PD		
	3 ..... ID          6 ..... PI		
RACT	Reverse action flag		bool
	off ... Higher mv → higher pv		
	on ... Higher mv → lower pv		
k	Controller gain $K$	⊙1.0	double
ti	Integral time constant $T_i$	⊙4.0	double
td	Derivative time constant $T_d$	⊙1.0	double
nd	Derivative filtering parameter $N$	⊙10.0	double
b	Setpoint weighting – proportional part	⊙1.0	double
c	Setpoint weighting – derivative part		double
tt	Tracking time constant. No meaning for controllers without integrator.	double	
		⊙1.0	
hilim	Upper limit of the controller output	⊙1.0	double
lolim	Lower limit of the controller output	⊙-1.0	double
dz	Dead zone		double
icotype	Controller output type	⊙1	long
	1 ..... Analog output		
	2 ..... Pulse width modulation (PWM)		
	3 ..... Step controller unit with position feedback (SCU)		
	4 ..... Step controller unit without position feedback (SCUV)		

## PIDUI – PID controller unit with variable parameters

### 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.

### Inputs

dv	Feedforward control variable	double
sp	Setpoint variable	double
pv	Process variable	double
tv	Tracking variable	double
hv	Manual value	double
MAN	Manual or automatic mode off ... Automatic mode on .... Manual mode	bool
IH	Integrator hold off ... Integration enabled on .... Integration disabled	bool
k	Controller gain $K$	double
ti	Integral time constant $T_i$	double
td	Derivative time constant $T_d$	double
nd	Derivative filtering parameter $N$	double
b	Setpoint weighting – proportional part	double

c	Setpoint weighting – derivative part	double
---	--------------------------------------	--------

## Outputs

mv	Manipulated variable (controller output)	double
dmv	Controller velocity output (difference)	double
de	Deviation error	double
SAT	Saturation flag	bool
	off ... The controller implements a linear control law	
	on .... The controller output is saturated	

## Parameters

irtype	Controller type (control law)	⊙6 long
	1 ..... D                      4 ..... P                      7 ..... PID	
	2 ..... I                      5 ..... PD	
	3 ..... ID                      6 ..... PI	
RACT	Reverse action flag	bool
	off ... Higher mv → higher pv	
	on .... Higher mv → lower pv	
tt	Tracking time constant. No meaning for controllers without integrator.	double ⊙1.0
hilim	Upper limit of the controller output	⊙1.0 double
lolim	Lower limit of the controller output	⊙-1.0 double
dz	Dead zone	double
icotype	Controller output type	⊙1 long
	1 ..... Analog output	
	2 ..... Pulse width modulation ( <a href="#">PWM</a> )	
	3 ..... Step controller unit with position feedback ( <a href="#">SCU</a> )	
	4 ..... Step controller unit without position feedback ( <a href="#">SCUV</a> )	

## 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 occuring sooner than the period of **btime** seconds since the last falling edge of the output signal elapses has no effect on the output signal **Y**.

### Input

<b>U</b>	Logical input of the block	<b>bool</b>
----------	----------------------------	-------------

### Output

<b>Y</b>	Logical output of the block	<b>bool</b>
----------	-----------------------------	-------------

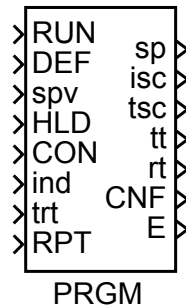
### Parameters

<b>dtime</b>	Minimum width of the output pulse [s]	⊙1.0	<b>double</b>
<b>btime</b>	Minimum delay between two successive output pulses [s]	⊙1.0	<b>double</b>

## PRGM – Setpoint programmer

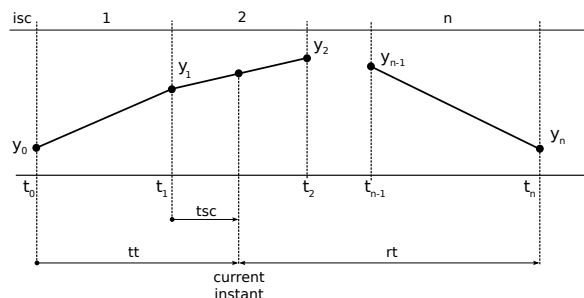
Block Symbol

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.



## Inputs

RUN	Enable execution	bool
DEF	Initialize <b>sp</b> to the value of <b>spv</b>	bool
spv	Initializing constant	double
HLD	Output and timer freezing	bool
CON	Continue from defined node	bool
ind	Index of the node to continue from	long
trt	Time to reach the defined node with index <b>ind</b>	double
RPT	Repetition flag	bool

## Outputs

<b>sp</b>	Setpoint variable (function value of the time function at given time)	double
<b>isc</b>	Current function sector	long
<b>tsc</b>	Time elapsed since the start of current sector	double
<b>tt</b>	Time elapsed since the start of program generation	double
<b>rt</b>	Remaining time till the end of program	double
CNF	Flag indicating that the configured curve is being followed	bool
E	Error flag – the node times are not ascending	bool

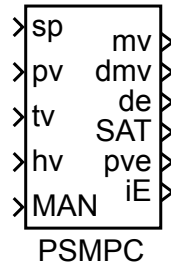
## Parameters

<b>n</b>	Number of sectors	$\downarrow 1 \uparrow 10000000$	$\odot 2$	long
<b>tmunits</b>	Time units		$\odot 1$	long
	1 ..... seconds			
	2 ..... minutes			
	3 ..... hours			
<b>tm</b>	( <b>n</b> + 1)-dimensional vector of ascending node times	$\odot [0 \ 1 \ 2]$		double
<b>y</b>	( <b>n</b> + 1)-dimensional vector of node values (values of the time function)	$\odot [0 \ 1 \ 0]$		double

## PSMPC – Pulse-step model predictive controller

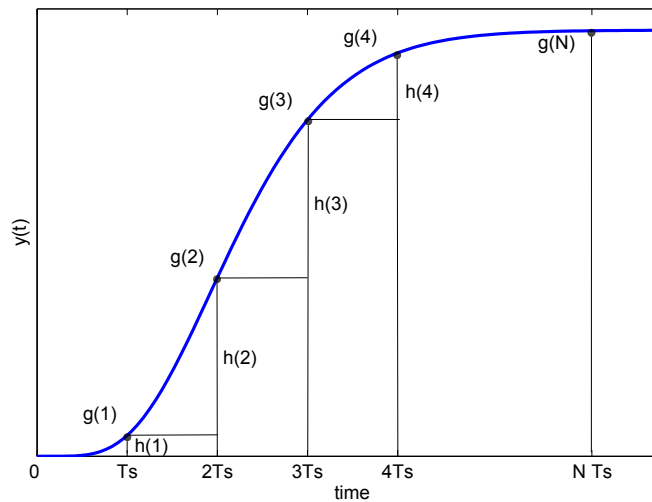
Block Symbol

Licence: [ADVANCED](#)



### Function Description

The **PSMPC** block can be used for control of hardly controllable linear time-invariant systems with manipulated value constraints (e.g. time delay or non-minimum phase systems). It is especially well suited for the case when fast transition without overshoot from one level of controlled variable to another is required. In general, the **PSMPC** block can be used where the PID controllers are commonly used.



The **PSMPC** block is a predictive controller with explicitly defined constraints on the amplitude of manipulated variable.

The prediction is based on the discrete step response  $g(j)$ ,  $j = 1, \dots, N$  is used. The figure above shows how to obtain the discrete step response  $g(j)$ ,  $j = 0, 1, \dots, N$  and the discrete impulse response  $h(j)$ ,  $j = 0, 1, \dots, N$  with sampling period  $T_s$  from continuous step response. Note that  $N$  must be chosen such that  $N \cdot T_s > t_{95}$ , where  $t_{95}$  is the time



to reach 95 % of the final steady state value.

For stable, linear and t-invariant systems with monotonous step response it is also possible to use the moment model set approach [4] and describe the system by only 3 characteristic numbers  $\kappa$ ,  $\mu$ , and  $\sigma^2$ , which can be obtained easily from a very short and simple experiment. The controlled system can be approximated by first order plus dead-time system

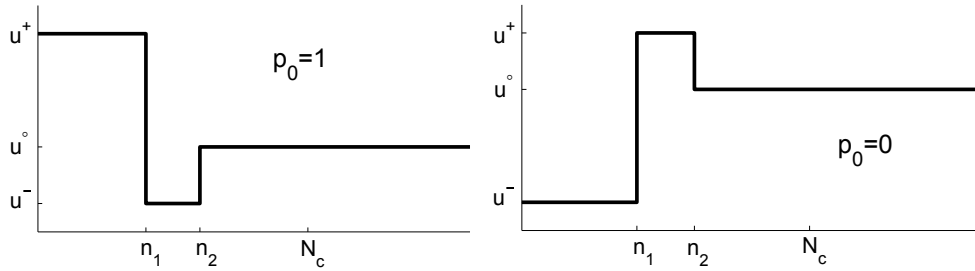
$$F_{FOPDT}(s) = \frac{K}{\tau s + 1} \cdot e^{-Ds}, \quad \kappa = K, \quad \mu = \tau + D, \quad \sigma^2 = \tau^2 \quad (7.1)$$

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  $u^\infty$ ) minimizes the criterion

$$I = \sum_{i=N_1}^{N_2} \hat{e}(k+i|k)^2 + \lambda \sum_{i=0}^{N_C} \Delta \hat{u}(k+i|k)^2 \rightarrow \min, \quad (7.3)$$

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  $\lambda$  penalizes the changes in the control signal. The algorithm used for solving the optimization task (7.3) combines brute force and the least squares method. The value  $u^\infty$  is determined using the least squares method for all admissible combinations of  $p_0, n_1$  and  $n_2$  and the optimal control sequence is selected afterwards. The selected sequence in the pulse-step shape is optimal in the open-loop sense. To convert from open-loop to closed-loop control strategy, only the first element of the computed control sequence is applied and the whole optimization procedure is repeated in the next sampling instant.

The parameters  $N_1$ ,  $N_2$ ,  $H_C$ , and  $\lambda$  in the criterion (7.3) take the role of design parameters. Only the last parameter  $\lambda$  is meant for manual tuning of the controller. In the case the model in the form (7.1) or (7.2) is used, the parameters  $N_1$  and  $N_2$  are determined automatically with respect to the  $\mu$  and  $\sigma^2$  characteristic numbers. The controller can be then effectively tuned by adjusting the characteristic numbers  $\kappa$ ,  $\mu$  and  $\sigma^2$ .

### Warning

It is necessary to set the **nsr** parameter to sufficiently large number to avoid Matlab/Simulink crash when using the **PSMPC** block for simulation purposes. Especially when using FOPDT or SOPDT model, the **nsr** parameter must be greater than the length of the internally computed discrete step response.

### Inputs

<b>sp</b>	Setpoint variable	double
<b>pv</b>	Process variable	double
<b>tv</b>	Tracking variable (applied control signal)	double
<b>hv</b>	Manual value	double
<b>MAN</b>	Manual or automatic mode	bool
	<b>off</b> ... Automatic mode	
	<b>on</b> .... Manual mode	

### Outputs

<b>mv</b>	Manipulated variable (controller output)	double
<b>dmv</b>	Controller velocity output (difference)	double
<b>de</b>	Deviation error	double
<b>SAT</b>	Saturation flag	bool
	<b>off</b> ... The controller implements a linear control law	
	<b>on</b> .... The controller output is saturated	
<b>pve</b>	Predicted process variable based on the controlled process model	double
<b>iE</b>	Error code	long
	0 ..... No error	
	1 ..... Incorrect FOPDT model	
	2 ..... Incorrect SOPDT model	
	3 ..... Invalid step response sequence	

### Parameters

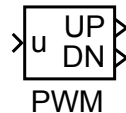
<b>nc</b>	Control horizon length ( $N_C$ )	⊙5	long
<b>np1</b>	Start of coincidence interval ( $N_1$ )	⊙1	long
<b>np2</b>	End of coincidence interval ( $N_2$ )	⊙10	long
<b>lambda</b>	Control signal penalization coefficient ( $\lambda$ )	⊙0.05	double
<b>umax</b>	Upper limit of the controller output ( $u^+$ )	⊙1.0	double
<b>umin</b>	Lower limit of the controller output ( $u^-$ )	⊙-1.0	double

<b>imtype</b>	Controlled process model type	⊙3	long
	1 ..... FOPDT model (7.1)		
	2 ..... SOPDT model (7.2)		
	3 ..... Discrete step response		
<b>kappa</b>	Static gain ( $\kappa$ )	⊙1.0	double
<b>mu</b>	Resident time constant ( $\mu$ )	⊙20.0	double
<b>sigma</b>	Measure of the system response length ( $\sqrt{\sigma^2}$ )	⊙10.0	double
<b>nsr</b>	Length of the discrete step response ( $N$ ), see the warning above		long
		↓10 ↑10000000 ⊙11	
<b>sr</b>	Discrete step response sequence ( $[g(1), \dots, g(N)]$ )		double
		⊙[0 0.2642 0.5940 0.8009 0.9084 0.9596 0.9826 0.9927 0.9970 0.9988 0.9995]	

## PWM – Pulse width modulation

Block Symbol

Licence: [STANDARD](#)



### Function Description

The **PWM** block implements a pulse width modulation algorithm for proportional actuators. In the general, it is assumed the input signal **u** ranges in the interval from -1 to +1. The width  $L$  of the output pulse is computed by the expression:

$$L = \text{pertm} * |u|,$$

where **pertm** is the modulation time period. If  $u > 0$  ( $u < 0$ ), the pulse is generated in the output **UP** (**DN**). However, the width of the generated pulses are affected by other parameters of the block. The asymmetry factor **asyfac** determines the ratio of negative pulses duration to positive pulses duration. The modified pulse widths are given by:

$$\begin{aligned} \text{if } u > 0 \text{ then } L(\text{UP}) &:= \begin{cases} L & \text{for } \text{asyfac} \leq 1.0 \\ L/\text{asyfac} & \text{for } \text{asyfac} > 1.0 \end{cases} \\ \text{if } u < 0 \text{ then } L(\text{DN}) &:= \begin{cases} L * \text{asyfac} & \text{for } \text{asyfac} \leq 1.0 \\ L & \text{for } \text{asyfac} > 1.0 \end{cases} \end{aligned}$$

Further, if the computed width is less than minimum pulse duration **dtime** the width is set to zero. If the pulse width differs from the modulation period **pertm** less than minimum pulse break time **btime** then width of the pulse is set to **pertm**. In the case the positive pulse is succeeded by the negative one (or vice versa) the latter pulse is possibly shifted in such a way that the distance between these pulses is at least equal to the minimum off time **offtime**. If **SYNCH** = on, then the change of the input value **u** causes the immediate recalculation of the current pulse widths if a synchronization condition is violated.

### Input

<b>u</b>	Analog input of the block	<b>double</b>
----------	---------------------------	---------------

### Outputs

<b>UP</b>	The "up" signal	<b>bool</b>
<b>DN</b>	The "down" signal	<b>bool</b>

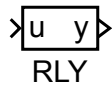
## Parameters

<code>pertm</code>	Modulation period length [s]	⊙10.0	double
<code>dtime</code>	Minimum width of the output pulse [s]	⊙0.1	double
<code>btime</code>	Minimum delay between output pulses [s]	⊙0.1	double
<code>offtime</code>	Minimum delay when altering direction [s]	⊙1.0	double
<code>asyfac</code>	Asymmetry factor	⊙1.0	double
<code>SYNCH</code>	Synchronization flag of the period start		bool
	<code>off</code> ... Synchronization disabled		
	<code>on</code> .... Synchronization enabled		

RLY – Relay with hysteresis

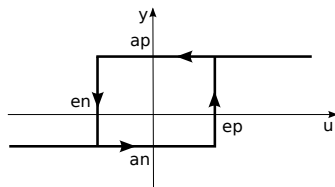
Block Symbol

Licence: [STANDARD](#)



Function Description

The RLY block transforms the input signal  $u$  to the output signal  $y$  according to the figure below.



Input

$u$	Analog input of the block	double
-----	---------------------------	--------

Output

$y$	Analog output of the block	double
-----	----------------------------	--------

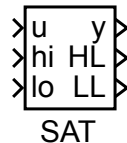
Parameters

$ep$	The value $u > ep$ causes $y = ap$ ("On")	$\odot 1.0$	double
$en$	The value $u < en$ causes $y = an$ ("Off")	$\odot -1.0$	double
$ap$	Output value $y$ in the "On" state	$\odot 1.0$	double
$an$	Output value $y$ in the "Off" state	$\odot -1.0$	double
$y0$	Initial output value at start-up		double

## 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  $\text{lolim}$  and  $\text{hilim}$  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  $\text{hilim0}$  and  $\text{lolim0}$  respectively or input-driven variables ( $\text{HLD} = \text{off}$ ,  $\text{hi}$  and  $\text{lo}$  inputs). The maximal rate at which the active limits may vary is given by time constants  $\text{tp}$  (positive slope) and  $\text{tn}$  (negative slope). These rates are active even if the saturation limits are changed manually ( $\text{HLD} = \text{on}$ ) using the  $\text{hilim0}$  and  $\text{lolim0}$  parameters. To allow immediate changes of the saturation limits, set  $\text{tp} = 0$  and  $\text{tn} = 0$ . The  $\text{HL}$  and  $\text{LL}$  outputs indicate the upper and lower saturation respectively.

If necessary, the  $\text{hilim0}$  and  $\text{lolim0}$  parameters are used as initial values for the input-driven saturation limits.

### Inputs

$u$	Analog input of the block	double
$hi$	Upper limit of the output signal (for the case $\text{HLD} = \text{off}$ )	double
$lo$	Lower limit of the output signal (for the case $\text{HLD} = \text{off}$ )	double

### Outputs

$y$	Analog output of the block	double
$\text{HL}$	Upper limit saturation indicator	bool
$\text{LL}$	Lower limit saturation indicator	bool

### Parameters

$\text{tp}$	Time constant defining the maximal positive slope of active limit changes	double	$\odot 1.0$
$\text{tn}$	Time constant defining the maximum negative slope of active limit changes	double	$\odot 1.0$
$\text{hilim0}$	Upper limit of the output (valid for $\text{HLD} = \text{on}$ )	double	$\odot 1.0$
$\text{lolim0}$	Lower limit of the output (valid for $\text{HLD} = \text{on}$ )	double	$\odot -1.0$

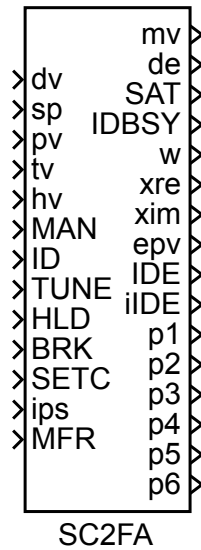
HLD      Fixed saturation limits       $\odot$ on    bool  
          off ... Variable limits      on .... Fixed limits



## SC2FA – State controller for 2nd order system with frequency autotuner

Block Symbol

Licence: [AUTOTUNING](#)



### Function Description

The **SC2FA** block implements a state controller for 2nd order system (7.4) with frequency autotuner. It is well suited especially for control (active damping) of lightly damped systems ( $\xi < 0.1$ ). But it can be used as an autotuning controller for arbitrary system which can be described with sufficient precision by the transfer function

$$F(s) = \frac{b_1 s + b_0}{s^2 + 2\xi\Omega s + \Omega^2}, \quad (7.4)$$

where  $\Omega > 0$  is the natural (undamped) frequency,  $\xi$ ,  $0 < \xi < 1$ , is the damping coefficient and  $b_1, b_0$  are arbitrary real numbers. The block has two operating modes: "Identification and design mode" and "Controller mode".

The "Identification and design mode" is activated by the binary input **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 of the **RUN** input. A harmonic signal with amplitude **uamp**, frequency  $\omega$  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

$T_b = \frac{2\pi}{\omega b}$  of the exciting sine wave in 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 usually lies within the interval  $\mathbf{cp} \in (0,95; 1)$ . The lower the damping coefficient  $\xi$  of the controlled system is, the closer to one the **cp** parameter must be.

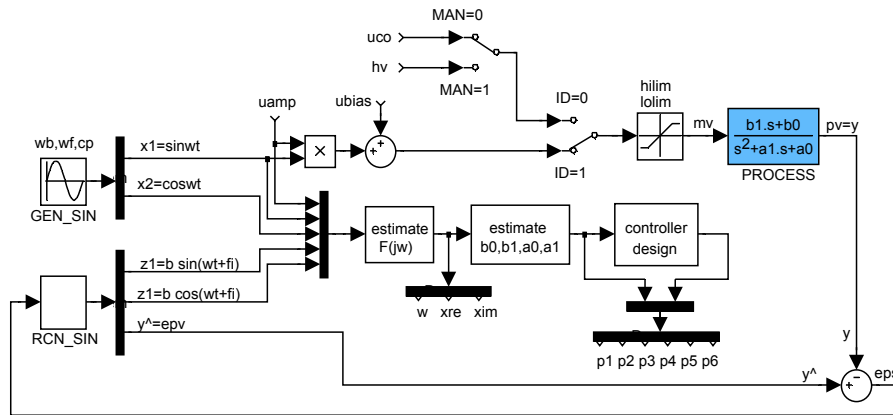
At the beginning of the identification period the exciting signal has a frequency of  $\omega = \omega b$ . After a period of **stime** seconds the estimation of current frequency response point starts. Its real and imaginary parts are available at the **xre** and **xim** outputs. If the **MANF** parameter is set to 0, then the frequency sweeping is stopped two times during the identification period. This happens when points with phase delay of **ph1** and **ph2** are reached for the first time. The breaks are **stime** seconds long. Default phase delay values are  $-60^\circ$  and  $-120^\circ$ , respectively, but these can be changed to arbitrary values within the interval  $(-360^\circ, 0^\circ)$ , where **ph1** > **ph2**. At the end of each break an arithmetic average is computed from the last **iavg** frequency point estimates. Thus we get two points of frequency response which are successively used to compute the controlled process model in the form of (7.4). If the **MANF** parameter is set to 1, then the selection of two frequency response points is manual. To select the frequency, set the input **HLD** = **on**, which stops the frequency sweeping. The identification experiment continues after returning the input **HLD** to 0. The remaining functionality is unchanged.

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**.

The **IDBSY** output is set to 1 during the "identification and design" phase. It is set back to 0 after the identification experiment finishes. A successful controller design is indicated by the output **IDE** = **off**. During the identification experiment the output **iIDE** displays the individual phases of the identification: **iIDE** = -1 means approaching the first point, **iIDE** = 1 means the break at the first point, **iIDE** = -2 means approaching the second point, **iIDE** = 2 means the break at the second point and **iIDE** = -3 means the last phase after leaving the second frequency response point. An error during the identification phase is indicated by the output **IDE** = **on** and the output **iIDE** provides more information about the error.

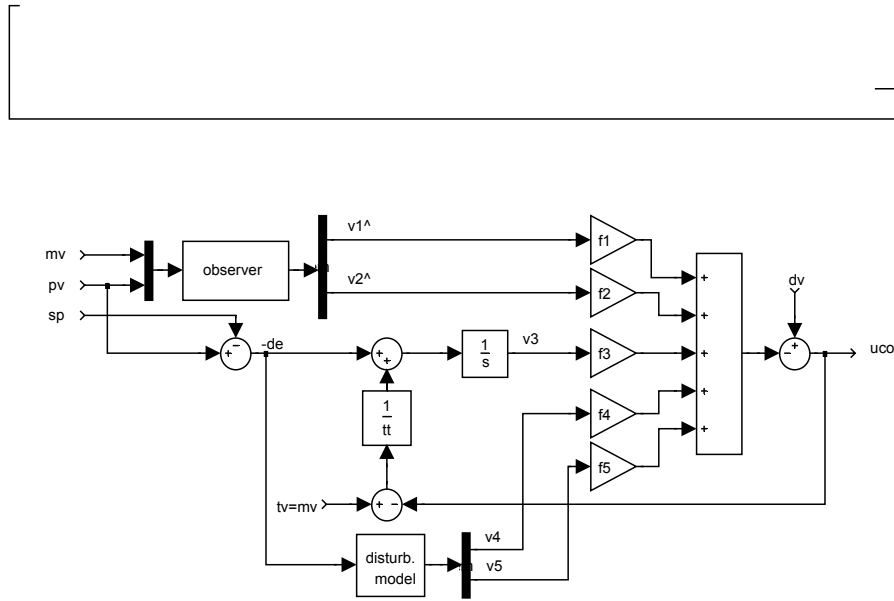
The computed state controller parameters are taken over by the control algorithm as soon as the **SETC** input is set to 1 (i.e. immediately if **SETC** is constantly set to **on**). The identified model and controller parameters can be obtained from the **p1**, **p2**, ..., **p6** outputs after setting the **ips** input to the appropriate value. 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.

The "Controller mode" (binary input  $ID = \text{off}$ ) has manual ( $MAN = \text{on}$ ) and automatic ( $MAN = \text{off}$ ) submodes. After a cold start of the block with the input  $ID = \text{off}$  it is assumed that the block parameters  $mb0$ ,  $mb1$ ,  $ma0$  and  $ma1$  reflect formerly identified coefficients  $b_0$ ,  $b_1$ ,  $a_0$  and  $a_1$  of the controlled system transfer function and the state controller design is performed automatically. Moreover if the controller is in the automatic mode and  $SETC = \text{on}$ , 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  $f_1, \dots, f_5$  after identification experiment (and  $SETC = \text{on}$ ).

~  
|



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**,...,**p6** after the identification experiment (**IDBSY** = **off**). The output signals meaning is switched when a change occurs at the **ips** input.

### Inputs

<b>dv</b>	Feedforward control variable	double
<b>sp</b>	Setpoint variable	double
<b>pv</b>	Process variable	double
<b>tv</b>	Tracking variable	double
<b>hv</b>	Manual value	double

MAN	Manual or automatic mode off ... Automatic mode    on .... Manual mode	bool
ID	Identification or controller operating mode off ... Controller mode    mode on .... Identification and design	bool
TUNE	Start the tuning experiment (off→on), the exciting harmonic signal is generated	bool
HLD	Stop frequency sweeping	bool
BRK	Termination signal	bool
SETC	Flag for accepting the new controller parameters and updating the control law off ... Parameters are only computed on .... Parameters are accepted as soon as computed off→on One-shot confirmation of the computed parameters	bool
ips	Switch for changing the meaning of the output signals 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 ... $\tau$ parameter p3 ... $\Omega$ parameter in rad/s p4 ... $\xi$ parameter p5 ... $\Omega$ 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	long
MFR	Generation of the parametric model frequency response at the w, xre and xim outputs (off→on triggers the generator)	bool

## Outputs

mv	Manipulated variable (controller output)	double
de	Deviation error	double

SAT	Saturation flag	bool
	off ... The controller implements a linear control law	
	on .... The controller output is saturated	
IDBSY	Identification running	bool
	off ... Identification not running	
	on .... Identification in progress	
w	Frequency response point estimate - frequency in rad/s	double
xre	Frequency response point estimate - real part	double
xim	Frequency response point estimate - imaginary part	double
epv	Reconstructed pv signal	double
IDE	Identification error indicator	bool
	off ... Successful identification experiment	
	on .... Identification error occurred	
iIDE	Error code	long
	101 ... Sampling period too low	
	102 ... Error identifying one or both frequency response point(s)	
	103 ... Manipulated variable saturation occurred during the identification experiment	
	104 ... Invalid process model	
p1..p6	Results of identification and design phase	double

## Parameters

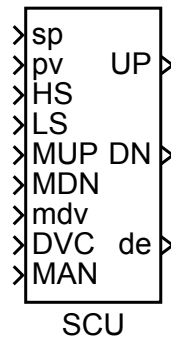
ubias	Static component of the exciting harmonic signal	double
uamp	Amplitude of the exciting harmonic signal	⊙1.0 double
wb	Frequency interval lower limit [rad/s]	⊙1.0 double
wf	Frequency interval upper limit [rad/s]	⊙10.0 double
isweep	Frequency sweeping mode	⊙1 long
	1 ..... Logarithmic	
	2 ..... Linear (not implemented yet)	
cp	Sweeping rate	↓0.5 ↑1.0 ⊙0.995 double
iavg	Number of values for averaging	⊙10 long
alpha	Relative positioning of the observer poles (in identification phase)	⊙2.0 double
xi	Observer damping coefficient (in identification phase)	⊙0.707 double
MANF	Manual frequency response points selection	bool
	off ... Disabled	
	on .... Enabled	
ph1	Phase delay of the 1st point in degrees	⊙-60.0 double
ph2	Phase delay of the 2nd point in degrees	⊙-120.0 double
stime	Settling period [s]	⊙10.0 double
ralpha	Relative positioning of the observer poles	⊙4.0 double
rx1	Observer damping coefficient	⊙0.707 double
ac11	Relative positioning of the 1st closed-loop poles couple	⊙1.0 double
xic11	Damping of the 1st closed-loop poles couple	⊙0.707 double

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
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	⊙1.0	double
dxi	Disturbance model damping coefficient		double
ac12	Relative positioning of the 2nd closed-loop poles couple	⊙2.0	double
xic12	Damping of the 2nd closed-loop poles couple	⊙0.707	double
tt	Tracking time constant	⊙1.0	double
hilim	Upper limit of the controller output	⊙1.0	double
lolim	Lower limit of the controller output	⊙-1.0	double
mb1p	Controlled system transfer function coefficient $b_1$		double
mb0p	Controlled system transfer function coefficient $b_0$	⊙1.0	double
ma1p	Controlled system transfer function coefficient $a_1$	⊙0.2	double
ma0p	Controlled system transfer function coefficient $a_0$	⊙1.0	double

## SCU – Step controller with position feedback

Block Symbol

Licence: [STANDARD](#)



### Function Description

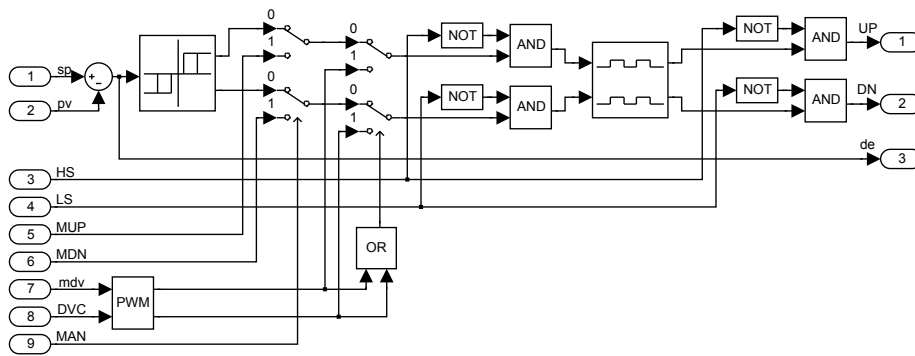
The **SCU** block implements the secondary (inner) position controller of the step controller loop. PIDU function block or some of the derived function blocks (**PIDMA**, etc.) is assumed as the primary controller.

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 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.

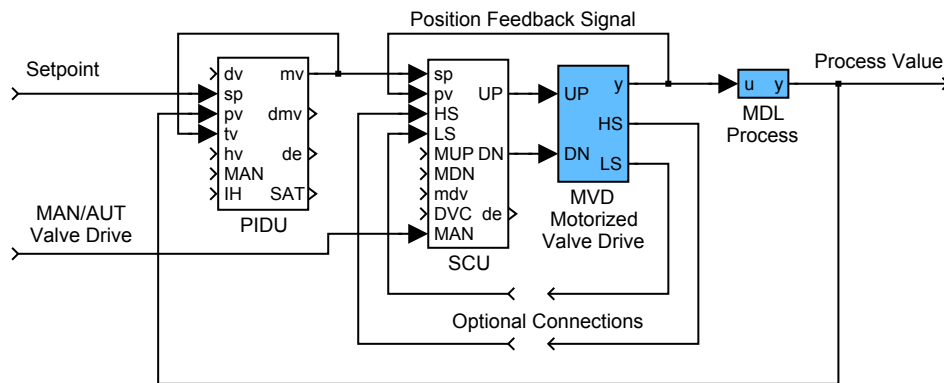
There is also a group of input signals for manual control available. The manual mode is activated by the **MAN = on** input signal. Then it is possible to move the motor back and forth by the **MUP** and **MDN** input signals. It is also possible to specify a position increment/decrement request by the **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.



## Inputs

sp	Setpoint (output of the primary controller)	double
pv	Controlled variable (position of the motorized valve drive)	double
HS	Upper end switch (detects the upper limit position of the valve)	bool
LS	Lower end switch (detects the lower limit position of the valve)	bool
MUP	Manual UP signal	bool
MDN	Manual DN signal	bool
mdv	Manual differential value (requested position increment/decrement with higher priority than direct signals MUP/MDN)	double
DVC	Differential value change command (off→on)	bool
MAN	Manual or automatic mode	bool
	off ... Automatic mode    on .... Manual mode	

## Outputs

UP	The "up" signal	bool
DN	The "down" signal	bool

<b>de</b>	Deviation error	<b>double</b>
-----------	-----------------	---------------

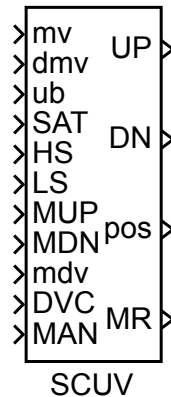
### Parameters

<b>thron</b>	Switch-on value	↓0.0 ⊙0.02	<b>double</b>
<b>throff</b>	Switch-off value	↓0.0 ⊙0.01	<b>double</b>
<b>dtime</b>	Minimum width of the output pulse [s]	↓0.0 ⊙0.1	<b>double</b>
<b>btime</b>	Minimum delay between two subsequent output pulses [s] to do	↓0.0 ⊙0.1	<b>double</b>
<b>RACT</b>	Reverse action flag		<b>bool</b>
	<b>off</b> ... Higher mv → higher pv		
	<b>on</b> ... Higher mv → lower pv		
<b>trun</b>	Motor time constant (determines the time during which the motor position changes by one unit)	↓0.0 ⊙10.0	<b>double</b>

## 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.

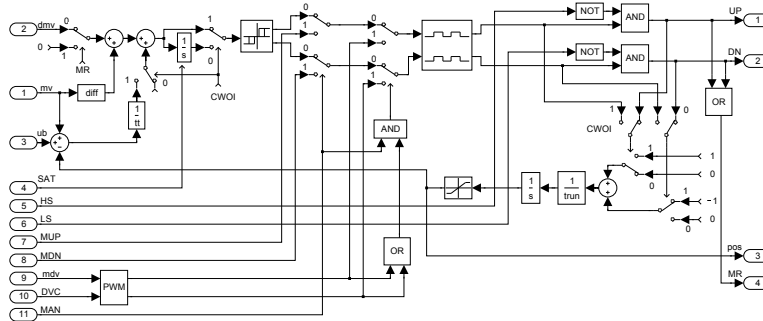
The position **pos** of the valve is estimated by an integrator with the time constant **trun**. If signals from high and low limit switches of the valve are available, they should be connected to the inputs **HS** and **LS**.

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.

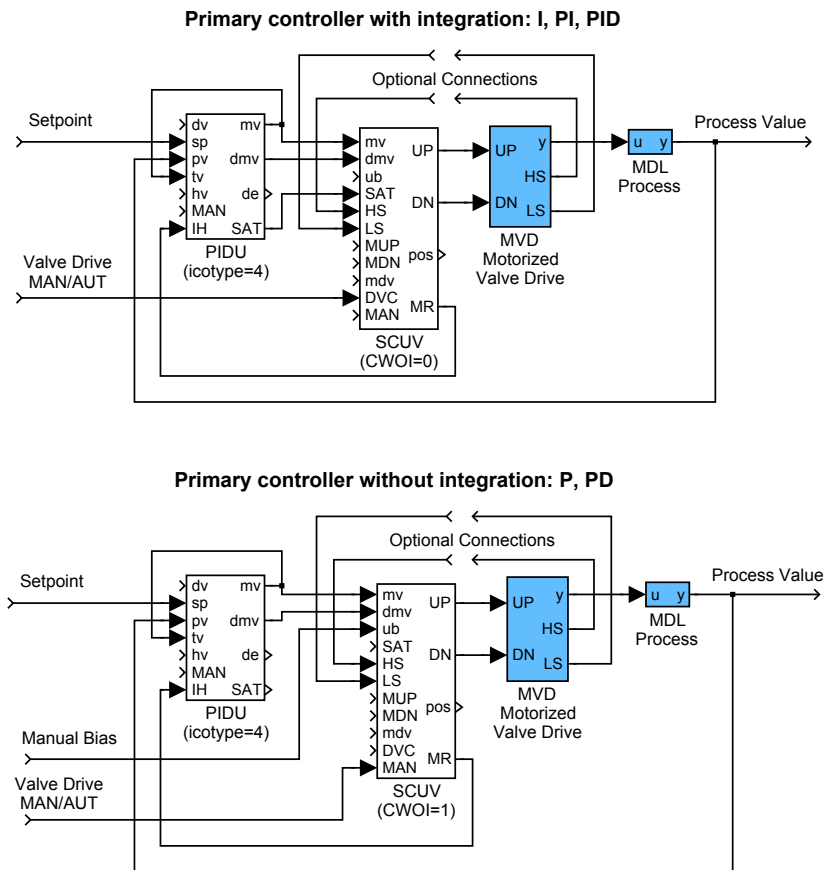
There is also a group of input signals for manual control available. The manual

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:



## Inputs

mv	Manipulated variable (controller output)	double
dmv	Controller velocity output (difference)	double

<b>ub</b>	Bias (only for P or PD primary controller)	<b>double</b>
<b>SAT</b>	Internal integrator reset (connected to the SAT output of the primary controller)	<b>bool</b>
<b>HS</b>	Upper end switch (detects the upper limit position of the valve)	<b>bool</b>
<b>LS</b>	Lower end switch (detects the lower limit position of the valve)	<b>bool</b>
<b>MUP</b>	Manual UP signal	<b>bool</b>
<b>MDN</b>	Manual DN signal	<b>bool</b>
<b>mdv</b>	Manual differential value (requested position increment/decrement with higher priority than direct signals MUP/MDN)	<b>double</b>
<b>DVC</b>	Differential value change command ( <b>off</b> → <b>on</b> )	<b>bool</b>
<b>MAN</b>	Manual or automatic mode	<b>bool</b>
	<b>off</b> ... Automatic mode <b>on</b> .... Manual mode	

## Outputs

<b>UP</b>	The "up" signal	<b>bool</b>
<b>DN</b>	The "down" signal	<b>bool</b>
<b>pos</b>	Position output of motor simulator	<b>double</b>
<b>MR</b>	Request to move the motor	<b>bool</b>
	<b>off</b> ... Motor idle (UP = <b>off</b> and DN = <b>off</b> )	
	<b>on</b> .... Request to move (UP = <b>on</b> or DN = <b>on</b> )	

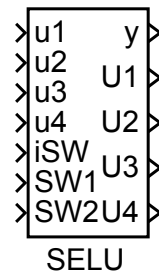
## Parameters

<b>thron</b>	Switch-on value	↓0.0 ⊕0.02	<b>double</b>
<b>throff</b>	Switch-off value	↓0.0 ⊕0.01	<b>double</b>
<b>dtime</b>	Minimum width of the output pulse [s]	↓0.0 ⊕0.1	<b>double</b>
<b>btime</b>	Minimum delay between two subsequent output pulses [s]	↓0.0 ⊕0.1	<b>double</b>
<b>RACT</b>	Reverse action flag		<b>bool</b>
	<b>off</b> ... Higher mv → higher pv		
	<b>on</b> .... Higher mv → lower pv		
<b>trun</b>	Motor time constant (determines the time during which the motor position changes by one unit)	↓0.0 ⊕10.0	<b>double</b>
<b>CW0I</b>	Controller without integration flag		<b>bool</b>
	<b>off</b> ... The primary controller has an integrator (I, PI, PID)		
	<b>on</b> .... The primary controller does not have an integrator (P, PD)		
<b>tt</b>	Tracking time constant	↓0.0 ⊕1.0	<b>double</b>

## 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).

### Inputs

<b>u1..u4</b>	Signals to be selected from	double
<b>iSW</b>	Active signal selector in case of <b>BINF = off</b>	long
<b>SW1</b>	Binary signal selector, used when <b>BINF = on</b>	bool
<b>SW2</b>	Binary signal selector, used when <b>BINF = on</b>	bool

### Outputs

<b>y</b>	Analog output of the block	double
<b>U1..U4</b>	Binary output signal for selector control	bool

## Parameter

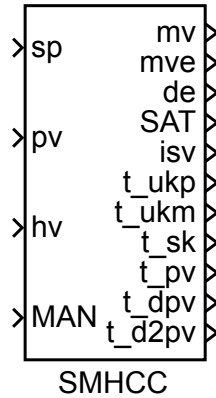
BINF	Enable the binary selectors	bool
	off ... Disabled (analog selector)	
	on .... Enabled (binary selectors)	



## SMHCC – Sliding mode heating/cooling controller

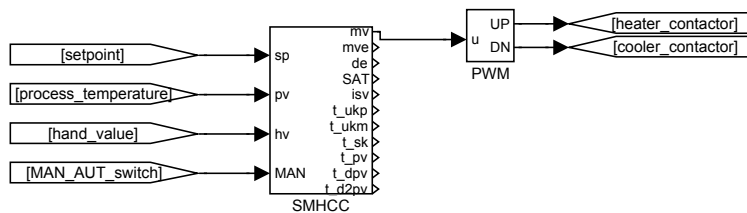
Block Symbol

Licence: [ADVANCED](#)



### Function Description

The sliding mode heating/cooling controller **SMHCC** is a novel high quality control algorithm intended for temperature control of heating-cooling (possibly asymmetrical) processes with ON-OFF heaters and/or ON-OFF coolers. The plastic extruder is a typical example of such process. However, it can also be applied to many similar cases, for example in thermal systems where a conventional thermostat is employed. To provide the proper control function the block **SMHCC** must be combined with the block **PWM** (Pulse Width Modulation) as depicted in the following figure.



It is important to note that the block **SMHCC** works with two time periods. The first period  $T_S$  is the sampling time of the process temperature, and this period is equal to the period with which the block **SMHCC** itself is executed. The second period  $T_C = i_{pwm} T_S$  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 ( $i_{pwmc}$  as maximal as possible), the ratio  $T_C/T_S$  as maximal as possible but  $T_C$  should be sufficiently small with respect to the process dynamics. An example of reasonable values for an extruder temperature control is as follows:

$$T_S = 0.1, i_{pwmc} = 100, T_C = 10s, T_R = 0.01s.$$

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 (**pv** – **sp**) and its first and second derivatives in the control period  $k$ , respectively, and  $\xi, \Omega$  are the control parameters described below. In the second phase,  $s_k$  is 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  $\xi, \Omega$ . For stable behavior, it must hold  $\xi > 0, \Omega > 0$ . A typical optimal value of  $\xi$  ranges in the interval  $[4, 8]$  and  $\xi$  about 6 is often a satisfactory value. The optimal value of  $\Omega$  strongly depends on the controlled process. The slower processes the lower optimal  $\Omega$ . The recommended value of  $\Omega$  for start of tuning is  $\pi/(5T_C)$ .

The manipulated variable **mv** usually ranges in the interval  $[-1, 1]$ . The positive (negative) value corresponds to heating (cooling). For example, **mv** = 1 means the full heating. The limits of **mv** can be reduced when needed by the controller parameters **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 **mv** is determined from the action amplitudes **uk\_p**, **uk\_m** by the following expression

$$\text{if } (s_k < 0.0) \text{ then } mv = uk\_p \text{ else } mv = -uk\_m.$$

Further, it is important to note that quasi sliding is seldom achievable because of a process dead time or disturbances. The suitable indicator of the quality of sliding is again the output **isv**. If the extraordinary fine tuning is required then it may be tried to find the better value for the bandwidth parameter **beta** of derivative filter, otherwise the default value 0.1 is preferred. In the manual mode (**MAN = on**) the controller input **hv** is (after limitation to the range  $[-hilim\_m, hilim\_p]$ ) copied to the manipulated variable **mv**.

## Inputs

<b>sp</b>	setpoint variable	double
<b>pv</b>	process variable	double
<b>hv</b>	manual value	double
<b>MAN</b>	controller mode	bool
	0 ..... automatic mode    1 ..... manual mode	

## Outputs

<b>mv</b>	manipulated variable (position controller output)	double
<b>mve</b>	equivalent manipulated variable	double
<b>de</b>	deviation error	double
<b>SAT</b>	saturation flag	bool
	0 ..... the controller implements a linear control law	
	1 ..... the controller output is saturated, $mv \geq hilim\_p$ or $mv \leq -hilim\_m$	
<b>isv</b>	number of the positive (+) or negative (−) sliding variable steps	long
<b>t_ukp</b>	current amplitude of heating	double
<b>t_ukm</b>	current amplitude of cooling	double
<b>t_sk</b>	discrete dynamic sliding variable $s_k$	double
<b>t_pv</b>	filtered control error <b>-de</b>	double
<b>t_dpv</b>	filtered first derivative of the control error <b>t_ek</b>	double
<b>t_d2pv</b>	filtered second derivative of the control error <b>t_ek</b>	double

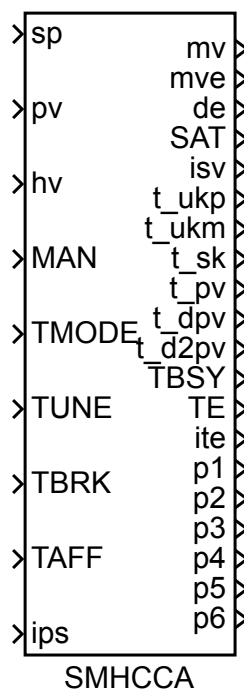
## Parameters

<code>ipwmc</code>	PWM cycle in the sampling periods of SMHCC ( $T_C/T_S$ )		long
<code>xi</code>	relative damping $\xi$ of sliding zero dynamics $\xi \geq 0$		double
<code>om</code>	natural frequency $\Omega$ of sliding zero dynamics	$\downarrow(0.0)$	double
<code>taup</code>	time constant for adaptation of heating action amplitude in seconds		double
<code>taum</code>	time constant for adaptation of cooling action amplitude in seconds		double
<code>beta</code>	bandwidth parameter of the derivative filter	$\downarrow 0$	double
<code>hilim_p</code>	high limit of the heating action amplitude	$\downarrow 0.0 \uparrow 1.0$	double
<code>hilim_m</code>	high limit of the cooling action amplitude	$\downarrow 0.0 \uparrow 1.0$	double
<code>u0_p</code>	initial value of the heating action amplitude after setpoint change and start of the block		double
<code>u0_m</code>	initial value of the cooling action amplitude after setpoint change and start of the block		double
<code>sp_dif</code>	Setpoint difference threshold	$\odot 10.0$	double
<code>tauf</code>	Equivalent manipulated variable filter time constant	$\odot 400.0$	double

## SMHCCA – Sliding mode heating/cooling controller with auto-tuner

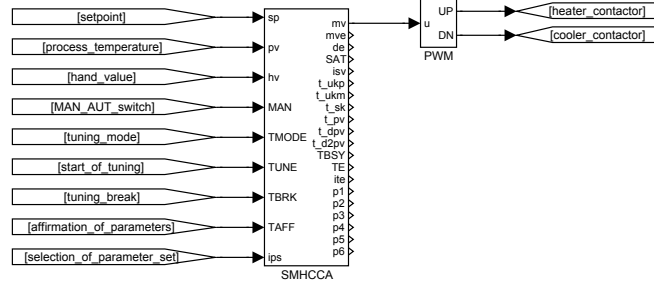
Block Symbol

Licence: [AUTOTUNING](#)



### Function Description

The sliding mode heating/cooling controller (SMHCCA) is a novel high quality control algorithm with a built-in autotuner for automatic tuning of the controller parameters. The controller is mainly intended for temperature control of heating-cooling (possibly asymmetrical) processes with ON-OFF heaters and/or ON-OFF coolers. The plastic extruder heating/cooling system is a typical example of such process. However, it can also be applied to many similar cases, for example, to thermal systems where a conventional thermostat is normally employed. To provide the proper control function, the SMHCCA block must be combined with the [PWM](#) block (Pulse Width Modulation) as depicted in the following figure.



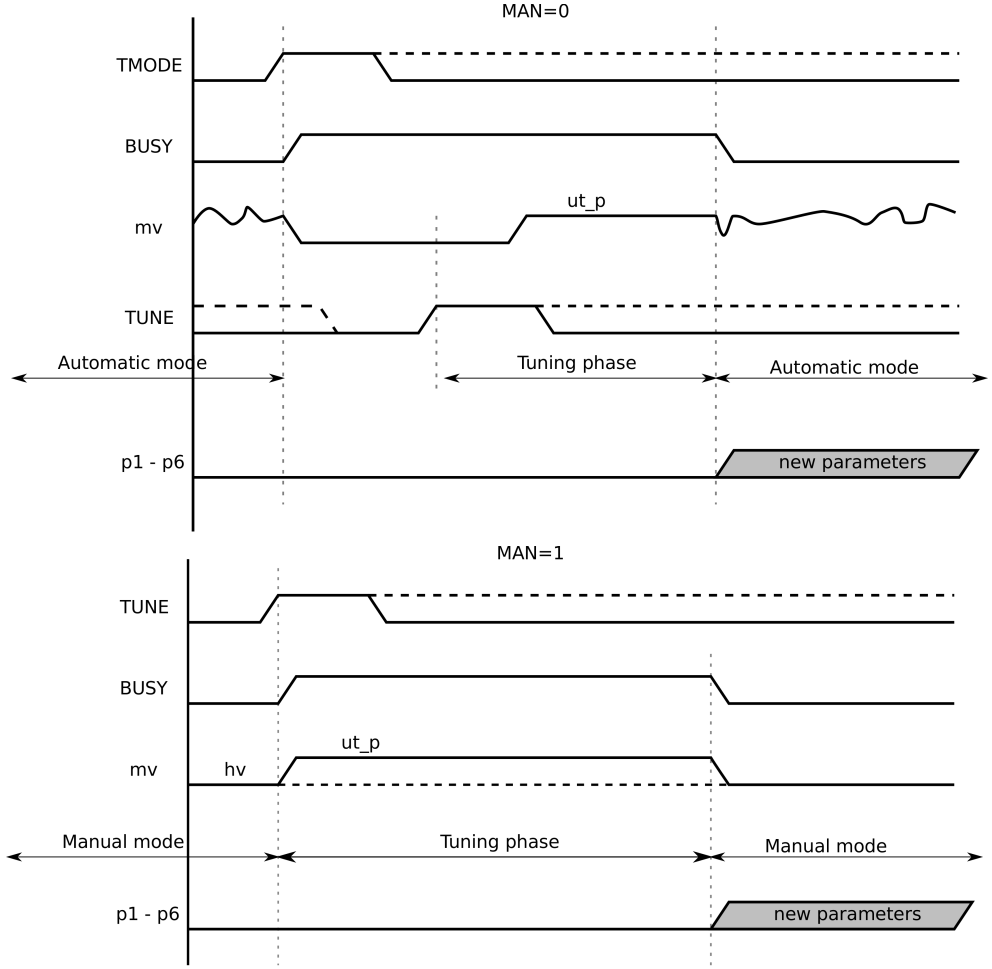
It is important to note that the block **SMHCCA** works with two time periods. The first period  $T_S$  is the sampling time of the process temperature, and this period is equal to the period with which the block **SMHCCA** itself is executed. The other period  $T_C = i_{pwm}T_S$  is the control period with which the block **SMHCCA** generates the manipulated variable. This period  $T_C$  is equal to the cycle time of **PWM** block. At every instant when the manipulated variable **mv** is changed by **SMHCCA** the PWM algorithm recalculates the width of the output pulse and starts a new PWM cycle. The time resolution  $T_R$  of the **PWM** block is third time period involved in. This period is equal to the period with which the block **PWM** is executed and generally may be different from  $T_S$ . To achieve the high quality of control it is recommended to choose  $T_S$  as minimal as possible ( $i_{pwm}$  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:

$$T_S = 0.1, i_{pwm} = 50, T_C = 5s, T_R = 0.1s.$$

Notice however that for a faster controlled system the sampling periods  $T_S$ ,  $T_C$  and  $T_R$  must be shortened! More precisely, the three minimal time constant of the process are important for selection of these time periods (all real thermal process has at least three time constants). For example, the sampling period  $T_S = 0.1$  is sufficiently short for such processes that have at least three time constants, the minimal of them is greater than 10s and the maximal is greater than 100s. For the proper function of the controller it is necessary that these time parameters are suitably chosen by the user according to the actual dynamics of the process! If **SMHCCA** is implemented on a processor with floating point arithmetic then the accurate setting of the sampling periods  $T_S$ ,  $T_C$ ,  $T_R$  and the parameter **beta** is critical for correct function of the controller. Also, some other parameters with the clear meaning described below have to be chosen manually. All the remaining parameters (**xi**, **om**, **taup**, **taum**, **tauf**) can be set by the built-in autotuner automatically. The autotuner uses the two methods for this purpose.

- The first one is dedicated to situations where the asymmetry of the process is not enormous (approximately, it means that the gain ratio of heating/cooling or cooling/heating is less than 5).
- The second method provides the tuning support for the strong asymmetric processes and is not implemented yet (So far, this method has been developed and tested in Simulink only).

Despite the fact that the first method of the tuning is based only on the heating regime, the resulting parameters are usually satisfactory for both heating and cooling regimes because of the strong robustness of sliding mode control. The tuning procedure is very quick and can be accomplished during the normal rise time period of the process temperature from cold state to the setpoint usually without any temporization or degradation of control performance. Thus the tuning procedure can be included in every start up from cold state to the working point specified by the sufficiently high temperature setpoint. Now the implemented procedure will be described in detail. The tuning procedure starts in the tuning mode or in the manual mode. If the tuning mode (**TMODE** = **on**) is selected the manipulated variable **mv** is automatically set to zero and the output **TBSY** is set to 1 for indication of the tuning stage of the controller. The cold state of the process is preserved until the initialization pulse is applied to the input **TUNE** ( $0 \rightarrow 1$ ). After some time (depending on **beta**), when the noise amplitude is estimated, the heating is switched on with the amplitude given by the parameter **ut\_p**. The process temperature **pv** and its two derivatives (outputs **t\_pv**, **t\_dpv**, **t\_d2pv**) are observed to obtain the optimal parameters of the controller. If the tuning procedure ends without errors, then **TBSY** is set to 0 and the controller begins to work in manual or automatic mode according to the input **MAN**. If **MAN** = **off** and affirmation input **TAFF** is set to 1, then the controller starts to work in automatic mode with the new parameter set provided by the tuner (if **TAFF** = **off**, then the new parameters are only displayed on the outputs **p1** . . . **p6**). If some error occurs during the tuning, then the tuning procedure stops immediately or stops after the condition **pv**>**sp** is fulfilled, the output **TE** is set to 1 and **ite** indicate the type of error. Also in this case, the controller starts to work in the mode determined by the input **MAN**. If **MAN** = **off** then works in automatic mode with the initial parameters before tuning! The tuning errors are usually caused either by an inappropriate setting of the parameter **beta** or by the too low value of **sp**. The suitable value of **beta** ranges in the interval (0.001,0.1). If a drift and noise in **pv** are large the small **beta** must be chosen especially for the tuning phase. The default value (**beta**=0.01) should work well for extruder applications. The correct value gives properly filtered signal of the second derivative of the process temperature **t\_d2pv**. This well-filtered signal (corresponding to the low value of **beta**) is mainly necessary for proper tuning. For control, the parameter **beta** may be sometimes slightly increased. The tuning procedure may be also started from manual mode (**MAN** = **off**) with any constant value of the input **hv**. However, the steady state must be provided in this case. Again, the tuning is started by the initialization pulse at the input **TUNE** ( $0 \rightarrow 1$ ) and after the stop of tuning the controller continues in the manual mode. In both cases the resulting parameters appear on the outputs **p1**, . . . , **p6**.



The control law of the block SMHCCA in automatic mode ( $MAN = \text{off}$ ) 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 ( $\text{pv} - \text{sp}$ ) and its first and second derivatives in the control period  $k$ , respectively, and  $\xi, \Omega$  are the control parameters described below. In the second phase,  $s_k$  is hold at the zero value (*the sliding phase*) by the proper control "bangs". Here, the heating action is alternated by cooling action and *vice versa* rapidly. The amplitudes of control actions are adapted appropriately to guarantee  $s_k = 0$  approximately. Thus, the hypothetical continuous dynamic sliding variable

$$s \triangleq \ddot{e} + 2\xi\Omega\dot{e} + \Omega^2 e$$



is approximately equal to zero at any time. Therefore the control deviation behaves according to the second order differential equation

$$s \triangleq \ddot{e} + 2\xi\Omega\dot{e} + \Omega^2e = 0$$

describing so called *zero sliding dynamics*. From it follows that the evolution of  $e$  can be prescribed by the parameters  $\xi, \Omega$ . For stable behavior, it must hold  $\xi > 0, \Omega > 0$ . A typical optimal value of  $\xi$  ranges in the interval  $[4, 8]$  and  $\xi$  about 6 is often a satisfactory value. The optimal value of  $\Omega$  strongly depends on the controlled process. The slower processes the lower optimal  $\Omega$ . The recommended value of  $\Omega$  for start of tuning is  $\pi/(5T_C)$ .

The manipulated variable  $mv$  usually ranges in the interval  $[-1, 1]$ . The positive (negative) value corresponds to heating (cooling). For example,  $mv = 1$  means the full heating. The limits of  $mv$  can be reduced when needed by the controller parameters `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 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  $mv$  is determined from the action amplitudes `uk_p`, `uk_m` by the following expression

$$\text{if } (s_k < 0.0) \text{ then } mv = uk\_p \text{ else } mv = -uk\_m.$$

Further, it is important to note that quasi sliding is seldom achievable because of a process dead time or disturbances. The suitable indicator of the quality of sliding is again the output `isv`. If the extraordinary fine tuning is required then it may be tried to find the better value for the bandwidth parameter `beta` of derivative filter, otherwise the default value 0.1 is preferred.

In the manual mode (`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  $mv$ .

## Inputs

<code>sp</code>	Setpoint variable	<code>double</code>
-----------------	-------------------	---------------------

<b>pv</b>	Process variable	double
<b>hv</b>	Manual value	double
<b>MAN</b>	Manual or automatic mode	bool
	0 ..... Automatic mode    1 ..... Manual mode	
<b>TMODE</b>	Tuning mode	bool
<b>TUNE</b>	Start the tuning experiment: TUNE off→on	bool
<b>TBRK</b>	Stop the tuning experiment: TBRK off→on	bool
<b>TAFF</b>	Affirmation of the parameter set provided by the tuning procedure: TAFF = on	bool
<b>ips</b>	Meaning of the output signals p1,...,p6	long
	0 ..... Controller parameters	
	p1 ... recommended control period $T_C$	
	p2 ... $\xi$	
	p3 ... $\omega_m$	
	p4 ... $\tau_{ap}$	
	p5 ... $\tau_{aum}$	
	p6 ... $\tau_{auf}$	
	1 ..... Auxiliary parameters	
	p1 ... $t_{p2}$ – time of the peak in the second derivative of pv	
	p2 ... $h_{peak2}$ – peak value in the second derivative of pv	
	p3 ... $d2$ – peak to peak amplitude of $t_{d2pv}$	
	p4 ... $t_{gain}$	

## Outputs

<b>mv</b>	Manipulated variable (controller output)	double
<b>mve</b>	Equivalent manipulated variable	double
<b>de</b>	Deviation error	double
<b>SAT</b>	Saturation flag	bool
	0 ..... Signal not limited	
	1 ..... Saturation limits active, $mv \geq hilim_p$ or $mv \leq -hilim_m$	
<b>isv</b>	Number of the positive (+) or negative (–) sliding variable steps	long
<b>t_ukp</b>	Current amplitude of heating	double
<b>t_ukm</b>	Current amplitude of cooling	double
<b>t_sk</b>	Discrete dynamic sliding variable	double
<b>t_pv</b>	Filtered process variable pv by 3rd order filter	double
<b>t_dpv</b>	Filtered first derivative of pv by 3rd order filter	double
<b>t_d2pv</b>	Filtered second derivative of pv by 3rd order filter	double
<b>TBSY</b>	Tuner busy flag (TBSY = on)	bool
<b>TE</b>	Tuning error	bool
	off ... Autotuning successful	
	on ... An error occurred during the experiment	

<b>ite</b>	Error code	<b>long</b>
0	..... No error	
1	..... Noise level in <b>pv</b> too high, check the temperature input	
2	..... Incorrect parameter <b>ut_p</b>	
3	..... Setpoint <b>sp</b> too low	
4	..... The two minimal process time constants are probably too small with respect to the sampling period $T_S$ OR too high level of noise in the second derivative of <b>pv</b> (try to decrease the <b>beta</b> parameter)	
5	..... Premature termination of the tuning procedure (TBRK)	
<b>pi</b>	Identified parameters with respect to <b>ips</b> , $i = 1, \dots, 6$	<b>double</b>

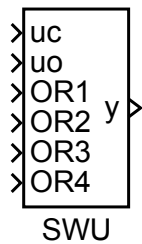
## Parameters

<b>ipwmc</b>	PWM cycle (in sampling periods of the block, $T_C/T_S$ )	$\odot 100$	<b>long</b>
<b>xi</b>	Relative damping of sliding zero dynamics	$\downarrow 0.5 \uparrow 8.0 \odot 1.0$	<b>double</b>
<b>om</b>	Natural frequency $\omega$ of sliding zero dynamics	$\downarrow 0.0 \odot 0.01$	<b>double</b>
<b>taup</b>	Time constant for adaptation of heating action amplitude [s]	$\odot 700.0$	<b>double</b>
<b>taum</b>	Time constant for adaptation of cooling action amplitude [s]	$\odot 400.0$	<b>double</b>
<b>beta</b>	Bandwidth parameter of the derivative filter	$\odot 0.01$	<b>double</b>
<b>hilim_p</b>	Upper limit of the heating action amplitude	$\downarrow 0.0 \uparrow 1.0 \odot 1.0$	<b>double</b>
<b>hilim_m</b>	Upper limit of the cooling action amplitude	$\downarrow 0.0 \uparrow 1.0 \odot 1.0$	<b>double</b>
<b>u0_p</b>	Initial amplitude of the heating action	$\odot 1.0$	<b>double</b>
<b>u0_m</b>	Initial amplitude of the cooling action	$\odot 1.0$	<b>double</b>
<b>sp_dif</b>	Setpoint difference threshold for blocking of heating/cooling amplitudes reset	$\odot 10.0$	<b>double</b>
<b>tauf</b>	Time constant of the filter for obtaining the equivalent manipulated variable	$\odot 400.0$	<b>double</b>
<b>itm</b>	Tuning method	$\odot 1$	<b>long</b>
	1 ..... Restricted to symmetrical processes		
	2 ..... Asymmetrical processes (not implemented yet)		
<b>ut_p</b>	Amplitude of heating for tuning experiment	$\downarrow 0.0 \uparrow 1.0 \odot 1.0$	<b>double</b>
<b>ut_m</b>	Amplitude of cooling for tuning experiment	$\downarrow 0.0 \uparrow 1.0 \odot 1.0$	<b>double</b>

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**, **OR2**, **OR3** and **OR4** are **off**, otherwise the output **y** takes over the **uo** input signal.

Inputs

uc	This input is copied to output <b>y</b> when all the binary inputs <b>OR1</b> , <b>OR2</b> , <b>OR3</b> and <b>OR4</b> are <b>off</b>	double
uo	This input is copied to output <b>y</b> when any of the binary inputs <b>OR1</b> , <b>OR2</b> , <b>OR3</b> , <b>OR4</b> is <b>on</b>	double
OR1	First logical output of the block	bool
OR2	Second logical output of the block	bool
OR3	Third logical output of the block	bool
OR4	Fourth logical output of the block	bool

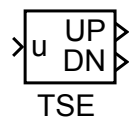
Output

y	Analog output of the block	double
---	----------------------------	--------

TSE – **Three-state element**

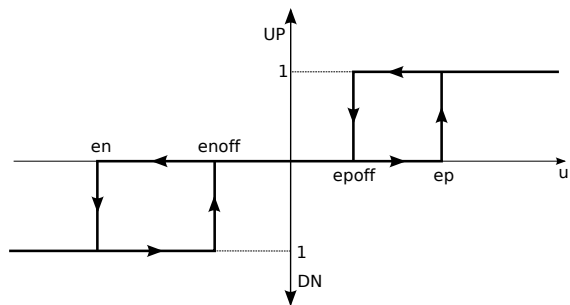
Block Symbol

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.



Input

u	Analog input of the block	double
---	---------------------------	--------

Outputs

UP	The "up" signal	bool
DN	The "down" signal	bool

Parameters

ep	The input value $u > ep$ results in $UP = on$ and $DN = off$	$\odot 1.0$	double
en	The input value $u < en$ results in $UP = off$ and $DN = on$	$\odot -1.0$	double
epoff	UP switch off value; if $UP = on$ and $u < epoff$ then $UP = off$	$\odot 0.5$	double
enoff	DN switch off value; if $DN = on$ and $u > enoff$ then $DN = off$	$\odot -0.5$	double



## Chapter 8

# LOGIC – Logic control

### Contents

---

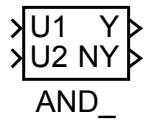
AND_ – Logical product of two signals . . . . .	244
ANDQUAD, ANDOCT, ANDHEXD – Logical product of multiple signals . .	245
ATMT – Finite-state automaton . . . . .	247
BDOCT, BDHEXD – Bitwise demultiplexers . . . . .	250
BITOP – Bitwise operation . . . . .	251
BMOCT, BMHEXD – Bitwise multiplexers . . . . .	252
COUNT – Controlled counter . . . . .	253
EATMT – Extended finite-state automaton . . . . .	255
EDGE_ – Falling/rising edge detection in a binary signal . . . . .	258
INTSM – Integer number bit shift and mask . . . . .	259
ISSW – Simple switch for integer signals . . . . .	260
INTSM – Integer number bit shift and mask . . . . .	261
ITOI – Transformation of integer and binary numbers . . . . .	262
NOT_ – Boolean complementation . . . . .	264
OR_ – Logical sum of two signals . . . . .	265
ORQUAD, OROCT, ORHEXD – Logical sum of multiple signals . . . . .	266
RS – Reset-set flip-flop circuit . . . . .	268
SR – Set-reset flip-flop circuit . . . . .	269
TIMER_ – Multipurpose timer . . . . .	270

---

**AND\_ – Logical product of two signals**

Block Symbol

Licence: [STANDARD](#)



**Function Description**

The **AND\_** block computes the logical product of two input signals **U1** and **U2**.  
If you need to work with more input signals, use the [ANDOCT](#) block.

**Inputs**

U1	First logical input of the block	bool
U2	Second logical input of the block	bool

**Outputs**

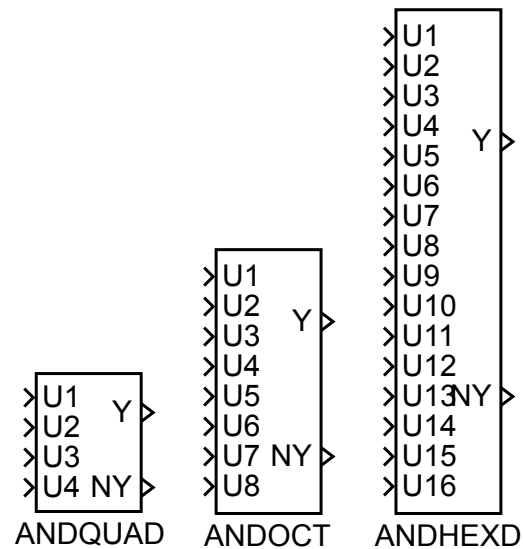
Y	Output signal, logical product ( $U1 \wedge U2$ )	bool
NY	Boolean complementation of Y ( $NY = \neg Y$ )	bool



## ANDQUAD, ANDOCT, ANDHEXD – Logical product of multiple signals

Block Symbols

Licence: [STANDARD](#)



### Function Description

The **ANDQUAD**, **ANDOCT** and **ANDHEXD** blocks compute the logical product of up to sixteen input signals  $U1, U2, \dots, U16$ . 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.

### Inputs

<b>U1..U16</b>	Logical inputs of the block	<b>bool</b>
----------------	-----------------------------	-------------

### Outputs

<b>Y</b>	Result of the logical operation	<b>bool</b>
<b>NY</b>	Boolean complementation of <b>Y</b>	<b>bool</b>

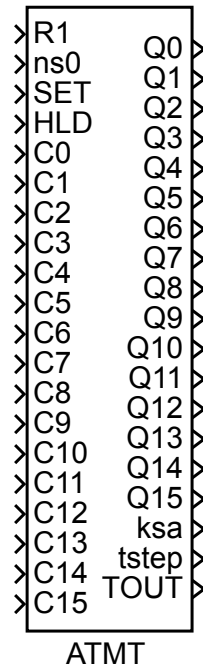
## Parameter

**n1**      List of signals to negate. The format of the list is e.g. 1,3..5,8.      **long**  
Third-party programs (Simulink, OPC clients etc.) work with an  
integer number, which is a binary mask, i.e. 157 (binary 10011101)  
in the mentioned case.

## ATMT – Finite-state automaton

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  $R1 = \text{on}$  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  $HLD = \text{on}$  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.

## Inputs

<b>R1</b>	Reset signal, $R1 = \text{on}$ brings the automat to the initial state $S0$ ; the $R1$ input overpowers the <b>SET</b> input	<b>bool</b>
<b>ns0</b>	This state is reached when rising edge occurs at the the <b>SET</b> input	<b>long</b>
<b>SET</b>	The rising edge of this signal forces the transition to the $ns0$ state	<b>bool</b>
<b>HLD</b>	The $HLD = \text{on}$ freezes the automat, no transitions occur regardless of the input signals, <b>tstep</b> is not increasing	<b>bool</b>
<b>C0...C15</b>	The transition conditions; $Ci = \text{on}$ means that the $i$ -th condition was fulfilled and the corresponding transition rule can be executed	<b>bool</b>

## Outputs

<b>Q0...Q15</b>	Output signals indicating the current state of the automat; the current state $i$ is indicated by $Qi = \text{on}$	<b>bool</b>
<b>ksa</b>	Integer code of the active state	<b>long</b>
<b>tstep</b>	Time elapsed since the current state was reached; the timer is set to 0 whenever a state transition occurs	<b>double</b>
<b>TOUT</b>	Flag indicating that the time limit for the current state was exceeded	<b>bool</b>

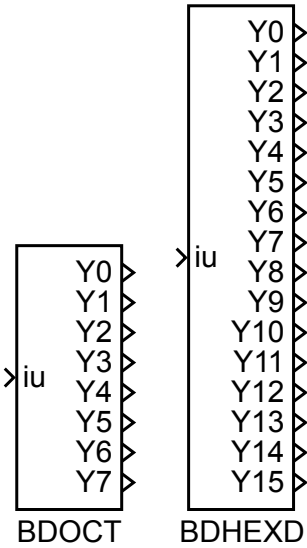
## Parameters

<b>moresteps</b>	Allow multiple transitions in one cycle of the automat off ... Disabled on .... Enabled	bool
<b>ntr</b>	Number of state transition table rows	$\downarrow 0 \uparrow 64 \odot 4$ long
<b>sfcname</b>	Filename of block configurator data file (filename is generated by system if parameter is empty)	string
<b>STT</b>	State transition table (matrix)	$\odot [0 \ 0 \ 1; \ 1 \ 1 \ 2; \ 2 \ 2 \ 3; \ 3 \ 3 \ 0]$ byte
<b>touts</b>	Vector of timeouts $TO0, TO1, \dots, TO15$ for the states $S0, S1, \dots, S15$	$\odot [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \ 16]$ double

BDOCT, BDHEXD – Bitwise demultiplexers

Block Symbols

Licence: [STANDARD](#)



Function Description

Both **BDOCT** and **BDHEXD** are bitwise demultiplexers for easy decomposition of the input signal to individual bits. The only difference is the number of outputs, the **BDOCT** block has 8 Boolean outputs while the **BDHEXD** block offers 16-bit decomposition. The output signals  $Y_i$  correspond with the individual bits of the input signal **iu**, the **Y0** output is the least significant bit.

Input

<b>iu</b>	Input signal to be decomposed	<b>long</b>
-----------	-------------------------------	-------------

Outputs

<b>Y0...Y15</b>	Individual bits of the input signal	<b>bool</b>
-----------------	-------------------------------------	-------------

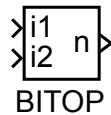
Parameter

<b>shift</b>	Bit shift of the input signal	$\downarrow 0 \uparrow 31$ <b>long</b>
--------------	-------------------------------	--

## 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).

### Inputs

<b>i1</b>	First integer input of the block	long
<b>i2</b>	Second integer input of the block	long

### Output

<b>n</b>	Result of the bitwise operation <b>iop</b>	long
----------	--	------

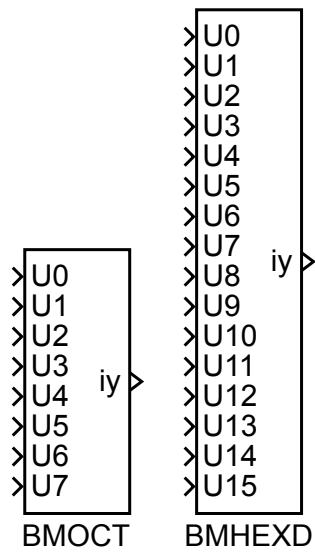
### Parameter

<b>iop</b>	Bitwise operation	⊙1	long
1	..... Bitwise negation (Bit NOT)		
2	..... Bitwise logical sum (Bit OR)		
3	..... Bitwise logical product (Bit AND)		
4	..... Bitwise logical exclusive sum (Bit XOR)		
5	..... Shift of the <b>i1</b> signal by <b>i2</b> bits to the left (Shift Left)		
6	..... Shift of the <b>i1</b> signal by <b>i2</b> bits to the right (Shift Right)		
7	..... 2's complement of the <b>i1</b> signal on 8 bits (2's Complement - Byte)		
8	..... 2's complement of the <b>i1</b> signal on 16 bits (2's Complement - Word)		
9	..... 2's complement of the <b>i1</b> signal on 32 bits (2's Complement - Long)		

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. The input signals  $U_i$  correspond with the individual bits of the output signal  $iy$ , the  $U_0$  input is the least significant bit.

Inputs

$U_0 \dots U_{15}$	Individual bits of the output signal	bool
--------------------	--------------------------------------	------

Output

$iy$	Composed output signal	long
------	------------------------	------

Parameter

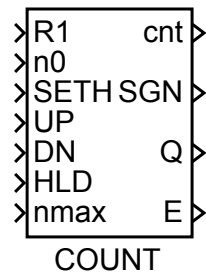
shift	Bit shift of the output signal	↓0 ↑31   long
-------	--------------------------------	---------------



## 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**.

### Inputs

<b>R1</b>	Block reset ( <b>R1</b> = <b>on</b> )	<b>bool</b>
<b>n0</b>	Value to set the counter to (using the <b>SETH</b> input)	<b>long</b>
<b>SETH</b>	Set the counter value to <b>n0</b> ( <b>SETH</b> = <b>on</b> )	<b>bool</b>
<b>UP</b>	Incrementing input signal	<b>bool</b>
<b>DN</b>	Decrementing input signal	<b>bool</b>
<b>HLD</b>	Counter freeze	<b>bool</b>
	<b>off</b> ... Counter is running	
	<b>on</b> .... Counter is locked	
<b>nmax</b>	Counter target value	<b>long</b>

### Outputs

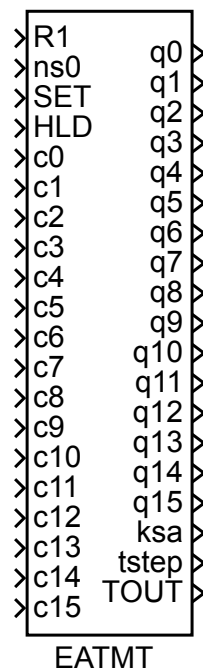
<b>cnt</b>	Total number of pulses	<b>long</b>
<b>SGN</b>	Sign of the <b>cnt</b> output	<b>bool</b>
	<b>off</b> ... for $\text{cnt} \leq 0$	
	<b>on</b> .... for $\text{cnt} > 0$	

Q	Target value indicator	bool
	off ... for cnt < nmax	
	on .... for cnt ≥ nmax	
E	Indicator of simultaneous occurrence of rising edges at both inputs UP and DN	bool

## 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 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 **EATMT** block.

## Inputs

<b>R1</b>	Reset signal, <b>R1 = on</b> brings the automat to the initial state $S0$ ; the <b>R1</b> input overpowers the <b>SET</b> input	<b>bool</b>
<b>ns0</b>	This state is reached when rising edge occurs at the the <b>SET</b> input	<b>long</b>
<b>SET</b>	The rising edge of this signal forces the transition to the <b>ns0</b> state	<b>bool</b>
<b>HLD</b>	The <b>HLD = on</b> freezes the automat, no transitions occur regardless of the input signals, <b>tstep</b> is not increasing	<b>bool</b>
<b>c0...c15</b>	Transition conditions, each input signal contains 16 transition conditions, see details above	

## Outputs

q0...q15	Output signals indicating the current state of the automat, see details above	long
ksa	Integer code of the active state	long
tstep	Time elapsed since the current state was reached; the timer is set to 0 whenever a state transition occurs	double
TOUT	Flag indicating that the time limit for the current state was exceeded	bool

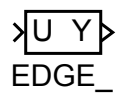
## Parameters

moresteps	Allow multiple transitions in one cycle of the automat off ... Disabled on .... Enabled	bool
ntr	Number of state transition table rows $\downarrow 0 \uparrow 1024 \odot 4$	long
sfcname	Filename of block configurator data file (filename is generated by system if parameter is empty)	string
STT	State transition table (matrix) $\odot [0 \ 0 \ 1; \ 1 \ 1 \ 2; \ 2 \ 2 \ 3; \ 3 \ 3 \ 0]$	short
touts	Vector of timeouts T00, T01, ..., T0255 for the states $S_0, S_1, \dots, S_{255}$ $\odot [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \ 16]$	double

EDGE\_ – Falling/rising edge detection in a binary signal

Block Symbol

Licence: [STANDARD](#)



Function Description

The `EDGE_` block detects rising (`off`→`on`) and/or falling (`on`→`off`) edges in the binary input signal `U`. The type of edges to detect is determined by the `iedge` parameter. As long as the input signal remains constant, the output `Y` is `off`. In the case when an edge corresponding with the `iedge` parameter is detected, the output `Y` is set to `on` for one sampling period.

Input

<code>U</code>	Logical input of the block	<code>bool</code>
----------------	----------------------------	-------------------

Output

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

Parameter

<code>iedge</code>	Type of edges to detect	$\odot$ 1	<code>long</code>
	1 . . . . . Rising edge		
	2 . . . . . Falling edge		
	3 . . . . . Both edges		

## INTSM – Integer number bit shift and mask

Block Symbol

Licence: [STANDARD](#)



### Function Description

The INTSM block performs bit shift of input value **i** by **shift** bits right (if **shift** is positive) or left (if **shift** is negative). Free space resulting from shifting is filled with zeros.

Output value **n** is calculated as bitwise AND of shifted input **i** and bit mask **mask**.

Typical application of this block is extraction of one or more adjacent bits from a given position in integer register which was read from some external system.

### Input

<b>i</b>	Integer value to shift and mask	long
----------	---------------------------------	------

### Parameters

<b>shift</b>	Bit shift (negative=left, positive=right)	↓-31 ↑31	long
<b>mask</b>	Bit mask (applied after bit shift)	↓XXX ↑XXX ⊙XXX	dword

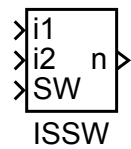
### Output

<b>n</b>	Resulting integer value	long
----------	-------------------------	------

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.

Inputs

<code>i1</code>	First integer input of the block	<code>long</code>
<code>i2</code>	Second integer input of the block	<code>long</code>
<code>SW</code>	Signal selector	<code>bool</code>
	<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	<code>long</code>
----------------	-----------------------------	-------------------



## INTSM – Integer number bit shift and mask

Block Symbol

Licence: [STANDARD](#)



### Function Description

The INTSM block performs bit shift of input value **i** by **shift** bits right (if **shift** is positive) or left (if **shift** is negative). Free space resulting from shifting is filled with zeros.

Output value **n** is calculated as bitwise AND of shifted input **i** and bit mask **mask**.

Typical application of this block is extraction of one or more adjacent bits from a given position in integer register which was read from some external system.

### Input

<b>i</b>	Integer value to shift and mask	long
----------	---------------------------------	------

### Parameters

<b>shift</b>	Bit shift (negative=left, positive=right)	↓-31 ↑31	long
<b>mask</b>	Bit mask (applied after bit shift)	↓XXX ↑XXX ⊙XXX	dword

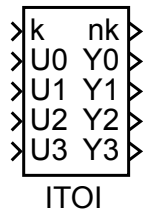
### Output

<b>n</b>	Resulting integer value	long
----------	-------------------------	------

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  $(U3\ U2\ U1\ U0)_2$ .

Inputs

$k$	Integer input of the block	long
$U0$	Binary input digit, weight of 1	bool
$U1$	Binary input digit, weight of 2	bool
$U2$	Binary input digit, weight of 4	bool
$U3$	Binary input digit, weight of 8	bool

Outputs

$nk$	Integer output of the block	long
$Y0$	Binary output digit, weight of 1	bool
$Y1$	Binary output digit, weight of 2	bool
$Y2$	Binary output digit, weight of 4	bool
$Y3$	Binary output digit, weight of 8	bool

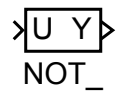
## Parameters

<b>BINF</b>	Enable the binary selectors	⊙on	bool
	off ... Disabled (integer input <i>k</i> )		
	on .... Enabled (binary input signals <i>U3...U0</i> )		
<b>fktab</b>	Vector of mapping table target values		byte
	⊙[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]		

**NOT\_ – Boolean complementation**

Block Symbol

Licence: [STANDARD](#)



**Function Description**

The NOT block negates the input signal.

**Input**

U	Logical input of the block	bool
---	----------------------------	------

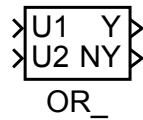
**Output**

Y	Logical output of the block ( $Y = \neg U$ )	bool
---	--	------

## OR\_ – Logical sum of two signals

Block Symbol

Licence: [STANDARD](#)



### Function Description

The **OR** block computes the logical sum of two input signals **U1** and **U2**.

If you need to work with more input signals, use the [OROCT](#) block.

### Inputs

U1	First logical input of the block	bool
U2	Second logical input of the block	bool

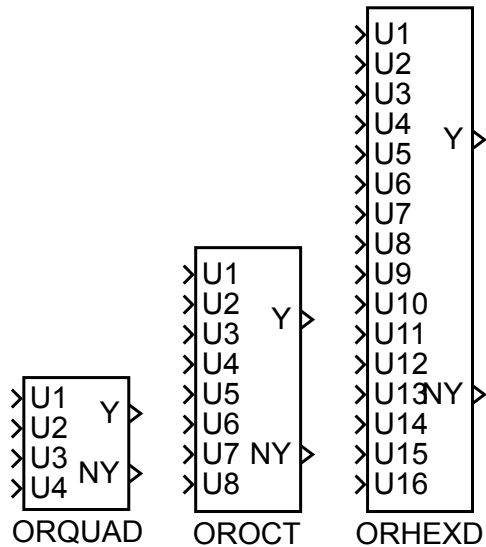
### Outputs

Y	Logical output of the block ( $U1 \vee U2$ )	bool
NY	Boolean complementation of Y ( $NY = \neg Y$ )	bool

ORQUAD, OROCT, ORHEXD – Logical sum of multiple signals

Block Symbols

Licence: [STANDARD](#)



Function Description

The **ORQUAD**, **OROCT** and **ORHEXD** blocks compute the logical sum of up to sixteen input signals **U1**, **U2**, ..., **U16**. The signals listed in the **n1** parameter are negated prior to computing the logical sum.

For an empty **n1** parameter a simple logical sum  $Y = U1 \vee U2 \vee U3 \vee U4 \vee U5 \vee U6 \vee U7 \vee \dots \vee U16$  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.

Inputs

U1..U16	Logical inputs of the block	bool
---------	-----------------------------	------

Outputs

Y	Result of the logical operation	bool
NY	Boolean complementation of Y	bool

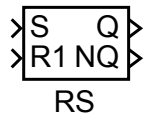
## Parameter

<code>n1</code>	List of signals to negate. The format of the list is e.g. 1,3..5,8. Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.	<code>long</code>
-----------------	---	-------------------

RS – Reset-set flip-flop circuit

Block Symbol

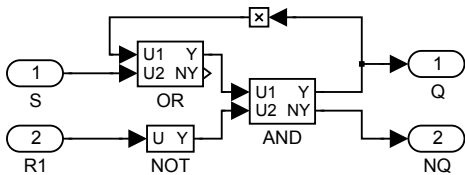
Licence: [STANDARD](#)



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.



Inputs

S	Flip-flop set, sets the Q output to on	bool
R1	Priority flip-flop reset, sets the Q output to off, overpowers the S input	bool

Outputs

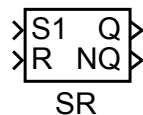
Q	Flip-flop circuit state	bool
NQ	Boolean complementation of Q	bool



## SR – Set-reset flip-flop circuit

Block Symbol

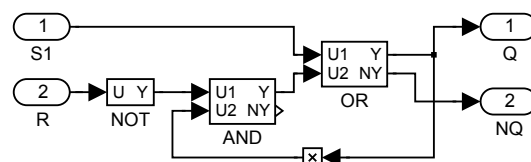
Licence: [STANDARD](#)



### 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.



### Inputs

S1	Priority flip-flop set, sets the Q output to <b>on</b> , overpowers the R input	bool
R	Flip-flop reset, sets the Q output to <b>off</b>	bool

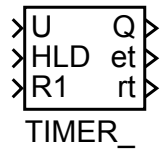
### Outputs

Q	Flip-flop circuit state	bool
NQ	Boolean complementation of Q	bool

TIMER\_ – Multipurpose timer

Block Symbol

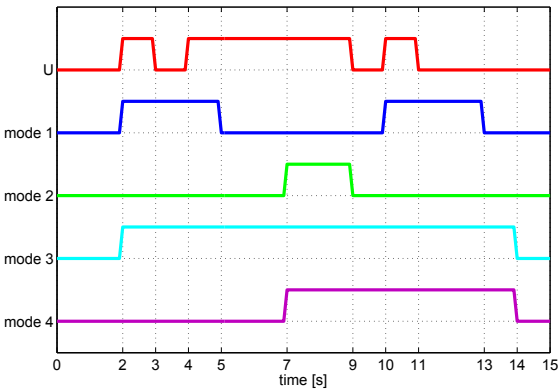
Licence: [STANDARD](#)



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. The timer can be paused by the `HLD` input.

The graph illustrates the behaviour of the block in individual modes for `pt = 3`:



Inputs

U	Trigger of the timer	bool
HLD	Timer hold	bool

Outputs

Q	Timer output	bool
et	Elapsed time [s]	double
rt	Remaining time [s]	double

## Parameters

<code>mode</code>	Timer mode	⊙1	<code>long</code>
1	.....	Pulse – an output pulse of the length <code>pt</code> is generated upon rising edge at the <code>U</code> input. All input pulses during the generation of the output pulse are ignored.	
2	.....	Delayed ON – the input signal <code>U</code> is copied to the <code>Q</code> output, but the start of the pulse is delayed by <code>pt</code> seconds. Any pulse shorter than <code>pt</code> is does not pass through the block.	
3	.....	Delayed OFF – the input signal <code>U</code> is copied to the <code>Q</code> output, but the end of the pulse is delayed by <code>pt</code> seconds. If the break between two pulses is shorter than <code>pt</code> , the output remains <code>on</code> for the whole time.	
4	.....	Delayed change – the <code>Q</code> output is set to the value of the <code>U</code> input no sooner than the input remains unchanged for <code>pt</code> seconds	
<code>pt</code>	Timer interval [s]	⊙1.0	<code>double</code>



## Chapter 9

# TIME – Blocks for handling time

### Contents

---

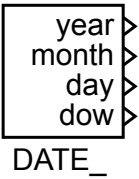
DATE_ – Current date . . . . .	274
DATETIME – Get, set and convert time . . . . .	275
TIME – Current time . . . . .	278
WSCH – Weekly schedule . . . . .	279

---

DATE\_ – Current date

Block Symbol

Licence: [STANDARD](#)



Function Description

The outputs of the `DATE_` function block correspond with the actual date of the operating system. Use the [DATETIME](#) block for advanced operations with date and time.

Outputs

<code>year</code>	Year	<code>long</code>
<code>month</code>	Month	<code>long</code>
<code>day</code>	Day	<code>long</code>
<code>dow</code>	Day of week, first day of week is Sunday (1)	<code>long</code>

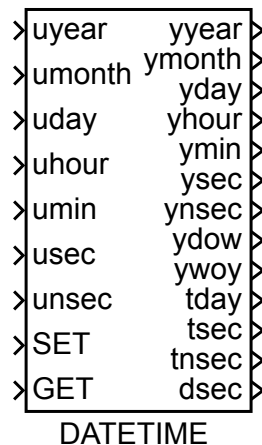
Parameter

<code>tz</code>	Timezone	<code>⊙1</code>	<code>long</code>
	1 . . . . .	Local time	
	2 . . . . .	UTC	

## DATETIME – Get, set and convert time

Block Symbol

Licence: [STANDARD](#)



### Function Description

The **DATETIME** block is intended for advanced date/time operations in the REXYGEN system.

It allows synchronization of the operating system clock and the clock of the REXYGEN system. When the executive of the REXYGEN system is initialized, both clocks are the same. But during long-term operation the clocks may loose synchronization (e.g. due to daylight saving time). If re-synchronization is required, the rising edge (**off**→**on**) at the **SET** input adjusts the clock of the REXYGEN system according to the block inputs and parameters.

It is highly recommended not to adjust the REXYGEN system time when the controlled machine/process is in operation. Unexpected behavior might occur.

If date/time reading or conversion is required, the rising edge (**off**→**on**) at the **GET** input triggers the action and the block outputs are updated. The outputs starting with 't' denote the total number of respective units since January 1st, 2000 UTC.

Both reading and adjusting clock can be repeated periodically if set by **getper** and **setper** parameters.

If the difference of the two clocks is below the tolerance limit **settol**, the clock of the REXYGEN system is not adjusted at once, a gradual synchronization is used instead. In such a case, the timing of the REXYGEN system executive is negligibly altered and the clocks synchronization is achieved after some time. Afterwards the timing of the REXYGEN executive is reverted back to normal.

For simple date/time reading use the [DATE\\_](#) and [TIME](#) function blocks.

## Inputs

uyear	Input for setting year		long
umonth	Input for setting month		long
uday	Input for setting day		long
uhour	Input for setting hours		long
umin	Input for setting minutes		long
usec	Input for setting seconds		long
unsec	Input for setting nanoseconds	↓-9,22E+18 ↑9,22E+18	large
SET	Trigger for setting time		bool
GET	Trigger for getting time		bool

## Outputs

yyear	Year		long
ymonth	Month		long
yday	Day		long
yhour	Hours		long
ymin	Minutes		long
ysec	Seconds		long
ynsec	Nanoseconds		long
ydw	Day of week		long
ywoy	Week of year		long
tday	Total number of days		long
tsec	Total number of seconds		long
tnsec	Total number of nanoseconds		large
dsec	Number of seconds since midnight		long

## Parameters

isetmode	Source for setting time	⊙1	long
	1 ..... OS clock		
	2 ..... Block inputs		
	3 ..... The unsec input		
	4 ..... The usec input		
	5 ..... The unsec input, relative		
igetmode	Source for getting or converting time	⊙6	long
	1 ..... OS clock		
	2 ..... Block inputs		
	3 ..... The unsec input		
	4 ..... The usec input		
	5 ..... The uday input		
	6 ..... REXYGEN clock		
settol	Tolerance for setting the REXYGEN clock [s]	⊙1.0	double
setper	Period for setting time [s] (0=not periodic)		double
getper	Period for getting time [s] (0=not periodic)	⊙0.001	double

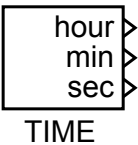


<b>FDOW</b>	First day of week is Sunday	<b>bool</b>
	<b>off</b> ... Week starts on Monday	
	<b>on</b> .... Week starts on Sunday	
<b>tz</b>	Timezone	$\odot$ 1 <b>long</b>
	1 ..... Local time	
	2 ..... UTC	

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.

Outputs

hour	Hours	long
min	Minutes	long
sec	Seconds	long

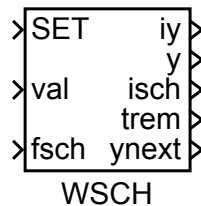
Parameter

tz	Timezone	⊕1	long
	1 . . . . .		Local time
	2 . . . . .		UTC

## WSCH – Weekly schedule

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.

### Inputs

<code>SET</code>	Trigger for setting the <code>y</code> and <code>iy</code> outputs	<code>bool</code>
<code>val</code>	Temporary value to set the <code>y</code> and <code>iy</code> outputs to	<code>double</code>
<code>fsch</code>	Forced schedule	<code>long</code>
	0 ..... standard weekly schedule	
	1 ..... Monday	
	2 ..... Tuesday	
	..... ..	
	7 ..... Sunday	
	8 and above additional daily programs as defined by the <code>wst</code> table	

### Outputs

<code>iy</code>	Integer output value	<code>long</code>
<code>y</code>	Output value	<code>double</code>
<code>isch</code>	Daily schedule identifier	<code>long</code>
<code>trem</code>	Time remaining in the current section (in seconds)	<code>double</code>
<code>ynext</code>	Output in the next section	<code>double</code>

### Parameters

<code>tz</code>	Timezone	$\odot 1$	<code>long</code>
	1 ..... Local time		
	2 ..... UTC		
<code>nmax</code>	Allocated size of arrays	$\downarrow 10 \uparrow 1000000 \odot 100$	<code>long</code>
<code>wst</code>	Weekly schedule table (list of triplets <i>day-hour-value</i> )		<code>double</code>
	$\odot [1 \ 0.01 \ 18.0; \ 2 \ 6.0 \ 22.0; \ 2 \ 18.0 \ 18.0; \ 3 \ 6.0 \ 22.0; \ 3 \ 18.0 \ 18.0; \ 4 \ 6.0 \ 22.0; \ 4$		
<code>specdays</code>	List of special days (list of pairs <i>date-daily program</i> )		<code>long</code>
	$\odot [20150406 \ 1; \ 20151224 \ 1; \ 20151225 \ 1; \ 20151226 \ 1; \ 20160101 \ 1; \ 20160328 \ 1; \ 20170$		

## Chapter 10

# ARC – Data archiving

### Contents

<b>10.1</b>	<b>Functionality of the archiving subsystem</b>	<b>282</b>
<b>10.2</b>	<b>Generating alarms and events</b>	<b>283</b>
	ALB, ALBI – Alarms for Boolean value	283
	ALN, ALNI – Alarms for numerical value	285
<b>10.3</b>	<b>Trends recording</b>	<b>287</b>
	ACD – Archive compression using Delta criterion	287
	TRND – Real-time trend recording	289
	TRNDV – Real-time trend recording with vector input	292
	TRNDLF – * Real-time trend recording (lock-free)	294
	TRNDVLF – * Real-time trend recording (for vector signals, lock-free)	296
<b>10.4</b>	<b>Archive management</b>	<b>297</b>
	AFLUSH – Forced archive flushing	297

The RexCore executive of the REXYGEN system consists of various interconnected subsystems (real-time subsystem, diagnostic subsystem, drivers subsystem, etc.). One of these subsystems is the *archiving subsystem*.

The archiving subsystem takes care of recording the history of the control algorithm. The first chapter describes the functionality of the archiving subsystem while the subsequent chapters describe the function blocks of the REXYGEN system.

The function blocks can be divided into groups by their use:

- Blocks for generating alarms and events
- Blocks for recording trends
- Blocks for handling archives

## 10.1 Functionality of the archiving subsystem

The archive in the REXYGEN system stores the history of events, alarms and trends of selected signals. There can be up to 15 archives in each target device. The types or archives are listed below:

**RAM memory archive** – Suitable for short-term data storage. The data access rate is very high but the data is lost on reboot.

**Archive in a backed-up memory** – Similar to the RAM archive but the data is not lost on restart. Data can be accessed fast but the capacity is usually quite limited (depends on the target platform).

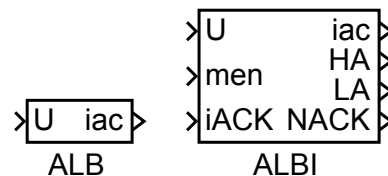
**Disk archive** The disk archives are files in a proprietary binary format. The files are easily transferrable among individual platforms and the main advantage is the size, which is limited only by the capacity of the storage medium. On the other hand, the drawback is the relatively slow data access.

Not all hardware platforms support all types of archives. The individual types which are supported by the platform can be displayed in the REXYGEN Studio program after clicking on the name (IP address) of the target device in the tree view panel. The supported types are listed in the lower left part of the **Target** tab.

## 10.2 Generating alarms and events

### ALB, ALBI – Alarms for Boolean value

Block Symbols

Licence: [STANDARD](#)

### Function Description

The ALB and ALBI blocks generate alarms or events when a Boolean input signal **U** changes. The **men** parameter selects whether the rising or falling or both edges in the input signal should be indicated. The **iac** output shows the current alarm (event) code.

The ALBI block is an extension of the ALB block as the alarms (events) are indicated also by Boolean output signals **HA**, **LA** and **NACK**. The type of edges to watch is selected by the **men** input signal and the alarms are acknowledged by the **iACK** input signal instead of parameters with the same name and meaning.

The events and alarms are differentiated by the **lv1** parameter in the REXYGEN system. The range  $1 \leq lv1 \leq 127$  is reserved for alarms. All starts, ends and acknowledgements of the alarms are stored in the archive. On the contrary, the range  $128 \leq lv1 \leq 255$  indicates events. Only the start (the time instant) of the event is stored in the archive.

### Inputs

<b>U</b>	Logical input of the block whose changes are watched	bool
<b>men</b>	Enable alarms	long
	0 ..... All alarms disabled	
	1 ..... Low-alarm enabled ( <b>LA</b> ) (falling edge in the input signal <b>U</b> )	
	2 ..... High-alarm enabled ( <b>HA</b> )(rising edge in the input signal <b>U</b> )	
	3 ..... All alarms enabled	
<b>iACK</b>	Acknowledge alarm	byte
	1 ..... Low-alarm acknowledge	
	2 ..... High-alarm acknowledge	
	3 ..... Both alarms acknowledge	
	Alarm is acknowledged on rising edge	

## Outputs

<b>iac</b>	Current alarm code	<b>long</b>
	0 ..... All alarms inactive	
	1 ..... Low-alarm active (LA)	
	2 ..... High-alarm active (HA)	
	256 ... Low-alarm not acknowledged (NACK)	
	512 ... High-alarm not acknowledged (NACK)	
<b>HA</b>	High-alarm indicator	<b>bool</b>
<b>LA</b>	Low-alarm indicator	<b>bool</b>
<b>NACK</b>	Alarm-not-acknowledged indicator	<b>bool</b>

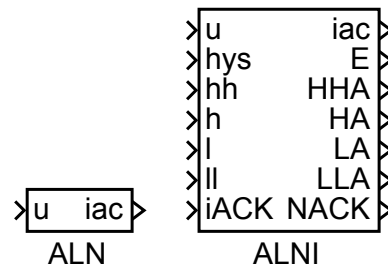
## Parameters

<b>arc</b>	List of archives to store the events. The format of the list is e.g. 1,3..5,8. The event will be stored in all listed archives (see the <a href="#">ARC</a> block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.	<b>word</b>
<b>id</b>	Identification code of the alarm in the archive. This identifier must be unique in the whole target device with the REXYGEN control system (i.e. in all archiving blocks). $\odot 1$	<b>word</b>
<b>lvl</b>	The level of the alarms (HA and LA) which differentiates alarms from events and defines the severity of the alarm/event $\downarrow 1 \odot 1$	<b>byte</b>
<b>Desc</b>	Extended description of the alarm which is displayed by the diagnostic tools of the REXYGEN system $\odot$ Alarm Description	<b>string</b>



## ALN, ALNI – Alarms for numerical value

### Block Symbols

Licence: [STANDARD](#)

### Function Description

The **ALN** and **ALNI** blocks generate two-level alarms or events when a limit value is exceeded (or not reached). There are four limit values the input signal **u** is compared to, namely high-limits **h** and **hh** and low-limits **l** and **ll**. The **iac** output shows the current alarm (event) code.

The **ALNI** block is an extension of the **ALN** block as the alarms (events) are indicated also by Boolean output signals **HHA**, **HA**, **LA** and **LLA** and the variables of the alarm algorithm are given by the input signals **hys**, **hh**, **h**, **l** and **ll** instead of parameters with the same name and meaning.

The events and alarms are differentiated by the **lv1** parameter in the **REXYGEN** system. The range  $1 \leq lv1 \leq 127$  is reserved for alarms. All starts, ends and acknowledgements of the alarms are stored in the archive. On the contrary, the range  $128 \leq lv1 \leq 255$  indicates events. Only the start (the time instant) of the event is stored in the archive.

### Inputs

<b>u</b>	Analog input of the block which is checked to remain within the given limits	double
<b>hys</b>	Alarm hysteresis for switching the alarm off	$\downarrow 0.0 \uparrow 10000000000.0$ double
<b>hh</b>	The second high-alarm limit, must be greater than <b>h</b>	double
<b>h</b>	High-alarm limit, must be greater than <b>l</b>	double
<b>l</b>	Low-alarm limit, must be greater than <b>ll</b>	double
<b>ll</b>	The second low-alarm limit	double
<b>iACK</b>	Alarm is acknowledged on rising edge of the individual bits of this input/parameter. E.g. value 15 acknowledges all alarms.	
<b>byte</b>		
1	Second low-alarm acknowledged	
2	Low-alarm acknowledge	
4	High-alarm acknowledge	
8	Second high-alarm acknowledge	

In case a one-level alarm is required, it is sufficient to set `lv12=0` or set the `hh` and `ll` limits to extreme values which can never be reached by the input signal.

## Outputs

<code>iac</code>	Current alarm code. Additional bitwise combinations of the codes may appear. E.g. 12 means both high alarms.	<code>long</code>
	0 .... Signal within limits	
	1 .... Low-alarm active	
	2 .... High-alarm active	
	4 .... Second low-alarm active	
	8 .... Second high-alarm active	
	256 ... Low-alarm not acknowledged	
	512 ... High-alarm not acknowledged	
	1024 .. Second low-alarm not acknowledged	
	2048 .. Second high-alarm not acknowledged	
	-1 .... Invalid alarm limits	
<code>E</code>	Error flag	<code>bool</code>
	off ... No error	
	on ... An error occurred, alarm limits disordered	
<code>HHA</code>	The second high-alarm indicator	<code>bool</code>
<code>HA</code>	High-alarm indicator	<code>bool</code>
<code>LA</code>	Low-alarm indicator	<code>bool</code>
<code>LLA</code>	The second low-alarm indicator	<code>bool</code>
<code>NACK</code>	Alarm-not-acknowledged indicator	<code>bool</code>

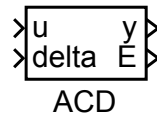
## Parameters

<code>acls</code>	Alarm class (data type to store)	⊙8	<code>byte</code>
	1 .... Bool    4 .... Long    7 .... Float		
	2 .... Byte    5 .... Word    8 .... Double		
	3 .... Short    6 .... DWord    9 .... Time		
<code>arc</code>	List of archives to store the events. The format of the list is e.g. 1,3..5,8. The event will be stored in all listed archives (see the <a href="#">ARC</a> block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.		<code>word</code>
<code>id</code>	Identification code of the alarm in the archive. This identifier must be unique in the whole target device with the REXYGEN control system (i.e. in all archiving blocks).	⊙1	<code>word</code>
<code>lv11</code>	The level of first high- and low-alarms ( <code>HA</code> and <code>LA</code> ) which differentiates alarms from events and defines the severity of the alarm/event	↓1 ⊙1	<code>byte</code>
<code>lv12</code>	The level of second high- and low-alarms ( <code>HHA</code> and <code>LLA</code> ) which differentiates alarms from events and defines the severity of the alarm/event	↓1 ⊙10	<code>byte</code>
<code>Desc</code>	Extended description of the alarm which is displayed by the diagnostic tools of the REXYGEN system	⊙Alarm Description	<code>string</code>

## 10.3 Trends recording

### ACD – Archive compression using Delta criterion

Block Symbol

Licence: [STANDARD](#)

#### Function Description

The **ACD** block is meant for storing compressed analog signals to archives using archive events.

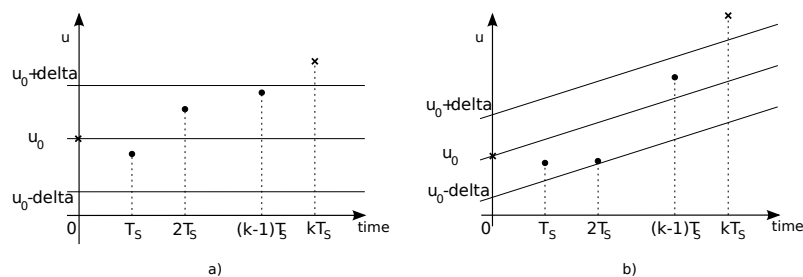
The main idea is to store the input signal **u** only when it changes significantly. The interval between two samples is in the range  $\langle t_{\min}, t_{\max} \rangle$  seconds (rounded to the nearest multiple of the sampling period). A constant input signal is stored every **tmax** seconds while 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.

For **TR = off** the condition  $|u - u_0| > \text{delta}$  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  $\times$ .



#### Inputs

<b>u</b>	Signal to compress and store	double
<b>delta</b>	Threshold for storing the signal	$\downarrow 0.0 \uparrow 10000000000.0$ double

## Outputs

y	The last value stored in the archive	double
E	Error flag – indicates that a significant change in the input signal occurred sooner than the <code>tmin</code> interval passes	bool
	off ... No error	
	on .... An error occurred	

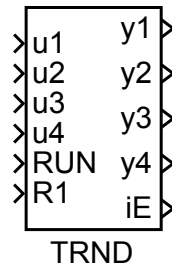
## Parameters

acls	Archive class determining the variable type to store	⊙8	byte
	1 ..... Bool	4 ..... Long	7 ..... Float
	2 ..... Byte	5 ..... Word	8 ..... Double
	3 ..... Short	6 ..... DWord	9 ..... Time
arc	List of archives to store the events. The format of the list is e.g. 1,3..5,8. The event will be stored in all listed archives (see the <a href="#">ARC</a> block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.		word
id	Identification code of the event in the archive. This identifier must be unique in the whole target device with the REXYGEN control system (i.e. in all archiving blocks).	⊙1	word
tmin	The shortest interval between two samples of the <code>u</code> input signal stored in the archive [s]	↓0.001 ↑1000000.0 ⊙1.0	double
tmax	The longest interval between two samples of the <code>u</code> input signal stored in the archive [s]	↓1.0 ↑1000000.0 ⊙1000.0	double
TR	Trend evaluation flag	⊙on	bool
	off ... The deviation of the input signal from the last stored value is evaluated		
	on .... The deviation of the input signal from the last value's trend is evaluated		
Desc	Extended description of the event which is displayed by the diagnostic tools of the REXYGEN system	⊙Value Description	string

## 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.

The number of stored signals is determined by the parameter **n**. In case the trend buffer of length 1 samples gets full, the oldest samples are overwritten. Data can be stored once in **pfac** executions of the block (decimation) and the data can be further processed according to the **p1type** to **p4type** parameters. The other decimation factor **afac** can be used for storing data in archives.

The type of trend buffers can be specified in order to conserve memory of the target device. The memory requirements of the trend buffers are defined by the formula  $s \cdot n \cdot l$ , 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 16 of this reference guide.

It can happen that the processed input value exceeds the representable limits when using different type of buffer than **Double**. In such a case the highest (lowest) representable number of the corresponding type is stored in the buffer and an error is binary encoded to the **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 Diagnostics diagnostic program.

### Inputs

<b>u1..u4</b>	Analog inputs to be processed and stored in the trend	<b>double</b>
<b>RUN</b>	Enable execution. The data are processed and stored if and only if <b>RUN = on</b> .	<b>bool</b>
<b>R1</b>	Input for clearing the trend contents. The buffers are cleared when <b>R1 = on</b> . This flag overpowers the <b>RUN</b> input.	<b>bool</b>

### Outputs

<b>y1..y4</b>	Analog outputs of the block set once in <b>pfac</b> executions of the block to the last values stored in the trend buffers	<b>double</b>
<b>iE</b>	Error code, see the table above	<b>long</b>

### Parameters

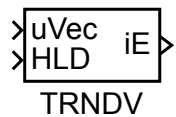
<b>n</b>	Number of signals to process and store in the trend buffers	<b>long</b>
<b>l</b>	Number of samples reserved in memory for each trend buffer	<b>long</b>
<b>btype</b>	Type of all <b>n</b> trend buffers	<b>long</b>
	<div> <div>1 . . . . . Bool</div> <div>2 . . . . . Byte</div> <div>3 . . . . . Short</div> <div>4 . . . . . Long</div> <div>5 . . . . . Word</div> <div>6 . . . . . DWord</div> <div>7 . . . . . Float</div> <div>8 . . . . . Double</div> <div>10 . . . . . Large</div> </div>	
<b>p<sub>type</sub><sub>i</sub></b>	The way the signal <b>u<sub>i</sub></b> , $i = 1 \dots 4$ , is processed. The last <b>pfac</b> samples are processed as selected and the result is stored in the $i$ -th trend buffer.	<b>long</b>
	<div>1 . . . . . No processing, just storing data</div> <div>2 . . . . . Minimum from the last <b>pfac</b> samples</div> <div>3 . . . . . Maximum from the last <b>pfac</b> samples</div> <div>4 . . . . . Sum of the last <b>pfac</b> samples</div> <div>5 . . . . . Simple average of the last <b>pfac</b> samples</div> <div>6 . . . . . Root mean square of the last <b>pfac</b> samples</div> <div>7 . . . . . Variance of the last <b>pfac</b> samples</div>	
<b>pfac</b>	Multiple of the block execution period defining the period for storing the data in the trend buffers. Data are stored with the period of <b>pfac</b> · $T_S$ unless <b>RUN = off</b> , where $T_S$ is the block execution period in seconds.	<b>long</b>

<b>afac</b>	Every <b>afac</b> -th sample stored in the trend buffer is also stored in the archives specified by the <b>arc</b> parameter. There are no data stored in the archives for <b>afac</b> = 0. Data are stored with the period of $\text{afac} \cdot \text{pfac} \cdot T_S$ , where $T_S$ is the block execution period in seconds. ↓0 ↑1000000	<b>long</b>
<b>arc</b>	List of archives to store the trend data. The format of the list is e.g. 1,3..5,8. The data will be stored in all listed archives (see the <a href="#">ARC</a> block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.	<b>word</b>
<b>id</b>	Identification code of the trend block. This identifier must be unique in the whole target device with the REXYGEN system (i.e. in all archiving blocks). ⊙1	<b>word</b>
<b>Title</b>	Title of the trend to be displayed in the diagnostic tools of the REXYGEN system, e.g. in the REXYGEN Diagnostics program ⊙Trend Title	<b>string</b>
<b>timesrc</b>	Source of timestamps. Each data sample in trend buffer is stored with a timestamp. For fast or short term trends where you are interested in sample-by-sample timing more than in absolute time, choose <b>CORETIMER</b> – REXYGEN internal technological time which is incremented by nominal period each base tick. For long running trends where you are interested mostly in absolute time shared with operating system (and possibly synchronized by NTP), choose <b>RTC</b> . Other values are intended for debug or special purposes. ⊙1 1 ..... <b>CORETIMER</b> – technological time – at current tick 2 ..... <b>CORETIMER-PRECISE</b> – technological time – at block execution 3 ..... <b>RTC</b> – real time clock (wallclock) from operating system – at current tick 4 ..... <b>RTC-PRECISE</b> – real time clock (wallclock) from operating system – at block execution 4 ..... <b>PFC</b> – raw high precision time (PerFormanceCounter)	<b>long</b>

TRNDV – Real-time trend recording with vector input

Block Symbol

Licence: [STANDARD](#)



Function Description

The **TRND** block is designed for storing input signals which arrive at the **uVec** input in vector form. On the contrary to the [TRND](#) block it allows storing more than 4 signals. The signals are stored in cyclic buffers in the memory of the target device. The main advantage of the **TRNDV** block is the synchronization with the real-time executive, which allows trending of even very fast signals (i.e. with very high sampling frequency). In contrary to asynchronous data storing in the higher level operator machine (host), there are no samples lost or multiply stored.

The number of stored signals is determined by the parameter **n**. In case the trend buffer of length 1 samples gets full, the oldest samples are overwritten. Data can be stored once in **pfac** executions of the block (decimation). The other decimation factor **afac** can be used for storing data in archives.

The type of trend buffers can be specified in order to conserve memory of the target device. The memory requirements of the trend buffers are defined by the formula  $s \cdot n \cdot 1$ , where *s* is the size of the corresponding variable in bytes. The default type **Double** consumes 8 bytes per sample, thus for storing e.g. **n** = 4 trends of this type and length 1 = 1000,  $8 \cdot 4 \cdot 1000 = 32000$  bytes are required. In case the input signals come from 16-bit A/D converter the **Word** type can be used and memory requirements drop to one quarter. Memory requirements and allowed ranges of individual types are summarized in table 1.1 on page 16 of this reference guide.

It is possible to read, display and export the stored data by the REXYGEN Diagnostics diagnostic program.

Inputs

<b>uVec</b>	Vector signal to record	<b>reference</b>
<b>HLD</b>	Input for freezing the cyclic buffers, no data is appended when HLD = on	<b>bool</b>

Output

<b>iE</b>	Error code	<b>error</b>
	i ..... REXYGEN general error	



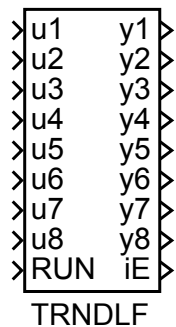
## Parameters

<b>n</b>	Number of signals (trend buffers)	$\downarrow 1 \uparrow 64 \odot 8$	long
<b>l</b>	Number of samples per trend buffer	$\downarrow 2 \uparrow 268435000 \odot 1000$	long
<b>btype</b>	Type of all trend buffers	$\odot 8$	long
	1 ..... Bool      4 ..... Long      7 ..... Float		
	2 ..... Byte      5 ..... Word      8 ..... Double		
	3 ..... Short      6 ..... DWord      10 ..... Large		
<b>pfac</b>	Multiple of the block execution period defining the period for storing the data in the trend buffers. Data are stored with the period of $\text{pfac} \cdot T_S$ unless <code>RUN = off</code> , where $T_S$ is the block execution period in seconds.	$\downarrow 1 \uparrow 1000000 \odot 1$	long
<b>afac</b>	Every <b>afac</b> -th sample stored in the trend buffer is also stored in the archives specified by the <b>arc</b> parameter. There are no data stored in the archives for <b>afac</b> = 0. Data are stored with the period of $\text{afac} \cdot \text{pfac} \cdot T_S$ , where $T_S$ is the block execution period in seconds.	$\downarrow 0 \uparrow 1000000$	long
<b>arc</b>	List of archives to store the trend data. The format of the list is e.g. 1,3,.5,8. The data will be stored in all listed archives (see the <a href="#">ARC</a> block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.		word
<b>id</b>	Identification code of the trend block. This identifier must be unique in the whole target device with the REXYGEN system (i.e. in all archiving blocks).	$\odot 1$	word
<b>Title</b>	Title of the trend to be displayed in the diagnostic tools of the REXYGEN system, e.g. in the REXYGEN Diagnostics program	$\odot \text{Trend Title}$	string

TRNDLF – \* **Real-time trend recording (lock-free)**

Block Symbol

Licence: [ADVANCED](#)



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

u1	First analog input of the block	double
u2	Second analog input of the block	double
u3	Third analog input of the block	double
u4	Fourth analog input of the block	double
u5	Fifth analog input of the block	double
u6	Sixth analog input of the block	double
u7	Seventh analog input of the block	double
u8	Eighth analog input of the block	double
RUN	Enable execution	bool

Parameters

n	Number of signals (trend buffers)	↓1 ↑8 ⊙8	long
1	Number of samples per trend buffer	↓0 ↑268435000 ⊙1024	long

<b>btype</b>	Type of all trend buffers	⊙8	long
	1 ..... Bool		
	2 ..... Byte		
	3 ..... Short		
	4 ..... Long		
	5 ..... Word		
	6 ..... DWord		
	7 ..... Float		
	8 ..... Double		
	-- .....		
	10 .... Large		
<b>Title</b>	Trend title string	⊙Trend Title	string
<b>timesrc</b>	Source of timestamps	⊙1	long

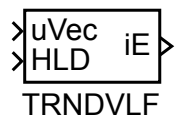
## Outputs

<b>y1</b>	First analog output of the block	double
<b>y2</b>	Second analog output of the block	double
<b>y3</b>	Third analog output of the block	double
<b>y4</b>	Fourth analog output of the block	double
<b>y5</b>	Fifth analog output of the block	double
<b>y6</b>	Sixth analog output of the block	double
<b>y7</b>	Seventh analog output of the block	double
<b>y8</b>	Eighth analog output of the block	double
<b>iE</b>	Error code (bitwise multiplexed)	long

TRNDVLF – \* Real-time trend recording (for vector signals, lock-free)

Block Symbol

Licence: [ADVANCED](#)



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

uVec	Vector signal to record	reference
HLD	Hold	bool

Parameters

n	Number of signals (trend buffers)	↓1 ↑64 ⊙8	long
l	Number of samples per trend buffer	↓2 ↑268435000 ⊙1024	long
btype	Type of all trend buffers	⊙8	long
	1 ..... Bool		
	2 ..... Byte		
	3 ..... Short		
	4 ..... Long		
	5 ..... Word		
	6 ..... DWord		
	7 ..... Float		
	8 ..... Double		
	-- .....		
	10 .... Large		
Title	Trend title string	⊙Trend Title	string
timesrc	Source of timestamps	⊙1	long

Output

iE	Error code	error
	i ..... REXYGEN general error	

## 10.4 Archive management

### 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.

#### Input

<b>FLUSH</b>	Force archive flushing	<b>bool</b>
--------------	------------------------	-------------

#### Parameter

<b>arc</b>	List of archives to store the events. The format of the list is e.g. 1,3..5,8. The event will be stored in all listed archives (see the <a href="#">ARC</a> block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.	<b>word</b>
------------	--	-------------



## Chapter 11

# STRING – Blocks for string operations

### Contents

---

CNS – String constant . . . . .	300
CONCAT – Concat string by pattern . . . . .	301
FIND – Find a Substring . . . . .	302
ITOS – Integer number to string conversion . . . . .	303
LEN – String length . . . . .	304
MID – Substring Extraction . . . . .	305
PJROCT – Parse JSON string (real output) . . . . .	306
PJSOCT – Parse JSON string (string output) . . . . .	307
REGEXP – Regular expresion parser . . . . .	308
REPLACE – Replace substring . . . . .	311
RTOS – Real Number to String Conversion . . . . .	312
SELSOCT – String selector . . . . .	313
STOR – String to real number conversion . . . . .	315

---

CNS – String constant

Block Symbol

Licence: [STANDARD](#)



Function Description

The `CNS` block is a simple string constant with maximal available size. A value of `scv` is always truncated to `nmax`.

Parameters

<code>scv</code>	String (constant) value	<code>string</code>
<code>nmax</code>	Allocated size of string [bytes]	$\downarrow 0 \uparrow 65520$ <code>long</code>

Output

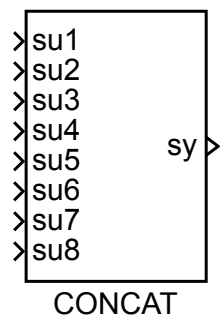
<code>sy</code>	String output value	<code>string</code>
-----------------	---------------------	---------------------



CONCAT – Concat string by pattern

Block Symbol

Licence: [STANDARD](#)



Function Description

Concatenates up to 8 input strings **su1** to **su8** by pattern specified in **ptrn** parameter.

Inputs

<b>su1</b> ..8	String input value	string
----------------	--------------------	--------

Parameters

<b>ptrn</b>	Concatenation pattern	⊙%1%2%3%4	string
<b>nmax</b>	Allocated size of string [bytes]	↓0 ↑65520	long

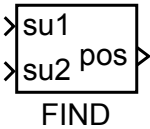
Output

<b>sy</b>	String output value	string
-----------	---------------------	--------

FIND – Find a Substring

Block Symbol

Licence: [STANDARD](#)



Function Description

The **FIND** block searches for the string **su2** in the string **su1** and returns a one-based index into **su1** if a **su2** is found or zero otherwise. Both **su1** and **su2** are truncated to **nmax**.

Inputs

<b>su1</b>	String input value	<b>string</b>
<b>su2</b>	String input value	<b>string</b>

Parameter

<b>nmax</b>	Allocated size of string [bytes]	↓0 ↑65520	<b>long</b>
-------------	----------------------------------	-----------	-------------

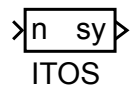
Output

<b>pos</b>	Position of substring	<b>long</b>
------------	-----------------------	-------------

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).

Input

<code>n</code>	Integer input of the block	<code>long</code>
----------------	----------------------------	-------------------

Output

<code>sy</code>	String output value	<code>string</code>
-----------------	---------------------	---------------------

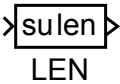
Parameters

<code>len</code>	Minimum length of output string	↓0 ↑30	<code>long</code>
<code>mode</code>	Output string format	⊙1	<code>long</code>
	1 . . . . . align right, fill with spaces		
	2 . . . . . align right, fill with zeroes		
	3 . . . . . align left, fill with spaces		
<code>radix</code>	Radix	⊙10	<code>long</code>
	2 . . . . . Binary		
	8 . . . . . Octal		
	10 . . . . . Decimal		
	16 . . . . . Hexadecimal		

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.

Input

<b>su</b>	String input value	<b>string</b>
-----------	--------------------	---------------

Parameter

<b>nmax</b>	Allocated size of string [bytes]	↓0 ↑65520	<b>long</b>
-------------	----------------------------------	-----------	-------------

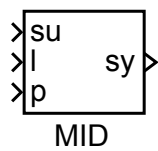
Output

<b>len</b>	Length of input string	<b>long</b>
------------	------------------------	-------------

## MID – Substring Extraction

Block Symbol

Licence: [STANDARD](#)



### Function Description

The **MID** block extracts a substring **sy** from **su**. The parameters **l** and **p** specify position and length of the string being extracted in UTF-8 characters. The parameter **p** is one-based.

### Inputs

<b>su</b>	String input value	<b>string</b>
<b>l</b>	Length of output string	<b>long</b>
<b>p</b>	Position of output string (one-based)	<b>long</b>

### Parameter

<b>nmax</b>	Allocated size of string [bytes]	↓0 ↑65520	<b>long</b>
-------------	----------------------------------	-----------	-------------

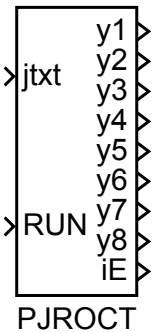
### Output

<b>sy</b>	String output value	<b>string</b>
-----------	---------------------	---------------

PJROCT – Parse JSON string (real output)

Block Symbol

Licence: [STANDARD](#)



Function Description

Parses input JSON string `jtxt` according to specified `name*` parameters when the input `RUN` is `on`. Output signals are `real` type.

Inputs

<code>jtxt</code>	JSON formatted string	<code>string</code>
<code>RUN</code>	Enable execution	<code>bool</code>

Parameters

<code>name1..8</code>	Name of JSON object	<code>string</code>
<code>nmax</code>	Allocated size of string [bytes]	<code>long</code>
<code>yerr</code>	Substitute value for an error case	<code>double</code>

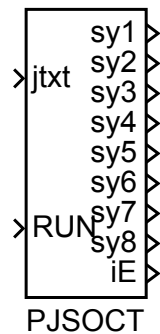
Outputs

<code>y1..8</code>	Block output signal	<code>double</code>
<code>iE</code>	Error code	<code>error</code>

PJSOCT – Parse JSON string (string output)

Block Symbol

Licence: [STANDARD](#)



Function Description

Parses input JSON string `jtxt` according to specified `name*` parameters when the input `RUN` is on. Output signals are `string` type.

Inputs

<code>jtxt</code>	JSON formated string	<code>string</code>
<code>RUN</code>	Enable execution	<code>bool</code>

Parameters

<code>name1..8</code>	Name of JSON object	<code>string</code>
<code>nmax</code>	Allocated size of string [bytes]	<code>long</code>

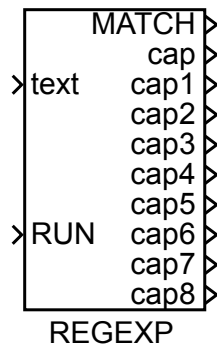
Outputs

<code>sy1..8</code>	String output value	<code>string</code>
<code>iE</code>	Error code	<code>error</code>

## REGEXP – Regular expression parser

Block Symbol

Licence: [ADVANCED](#)



### Function Description

This block implements a subset of Perl or C# or Unix command **grep** regular expression syntax.

Supported syntax is :

- `(?i)` ... Must be at the beginning of the regex. Makes match case-insensitive
- `^` ... Match beginning of a buffer
- `$` ... Match end of a buffer
- `()` ... Grouping and substring capturing
- `\s` ... Match whitespace
- `\S` ... Match non-whitespace
- `\d` ... Match decimal digit
- `\n` ... Match new line character
- `\r` ... Match line feed character
- `\f` ... Match form feed character
- `\v` ... Match vertical tab character
- `\t` ... Match horizontal tab character
- `\b` ... Match backspace character



- `+` ... Match one or more times (greedy)
- `+`? ... Match one or more times (non-greedy)
- `*` ... Match zero or more times (greedy)
- `*`? ... Match zero or more times (non-greedy)
- `?` ... Match zero or once (non-greedy)
- `x|y` ... Match x or y (alternation operator)
- `\meta` ... Match one of the meta characters: `^$().[*+?|\`
- `\xHH` ... Match byte with hex value 0xHH, e.g. `\x4a`
- `[...]` ... Match any character from set. Ranges like `[a-z]` are supported.
- `[^...]` ... Match any character except the ones in set. Ranges like `[a-z]` are supported.

## Examples

- `[0-9]+` ... Find first integer in input string (and put it into **cap** output)
- `[-+]?[0-9]*\.[0-9]+([eE] [-+]?[0-9]+)?` ... Find first real number in input string (and put it into **cap** output)
- `^\s*(.*?)\s*$` ... Put trimmed input string into **cap1** output
- `num\s*:\s*([0-9]*\.[0-9]*)` ... Expect input string in JSON format; find integer parameter **num**, and put its value into **cap1**

## Inputs

<b>text</b>	String to parse	<b>string</b>
<b>RUN</b>	Enable execution	<b>bool</b>

## Parameters

<b>expr</b>	Regular expresion pattern	<b>string</b>
<b>nmax</b>	Allocated size of string	↓0 ↑65534 <b>long</b>
<b>bufmax</b>	Parser internal buffer size (0 = autodetect)	↓0 ↑10000000 <b>long</b>

## Outputs

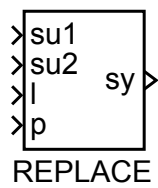
<b>MATCH</b>	Pattern match flag	<b>bool</b>
--------------	--------------------	-------------

<code>cap</code>	Whole matching string	<code>string</code>
<code>cap1</code>	Captured string (string matched to 1st bracket)	<code>string</code>
<code>cap2</code>	Captured string (string matched to 2nd bracket)	<code>string</code>
<code>cap3</code>	Captured string (string matched to 3rd bracket)	<code>string</code>
<code>cap4</code>	Captured string (string matched to 4th bracket)	<code>string</code>
<code>cap5</code>	Captured string (string matched to 5th bracket)	<code>string</code>
<code>cap6</code>	Captured string (string matched to 6th bracket)	<code>string</code>
<code>cap7</code>	Captured string (string matched to 7th bracket)	<code>string</code>
<code>cap8</code>	Captured string (string matched to 8th bracket)	<code>string</code>

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 one-based.

Inputs

su1	String input value	string
su2	String input value	string
l	Length of origin text	long
p	Position of origin text (one-based)	long

Parameter

nmax	Allocated size of string [bytes]	↓0 ↑65520	long
------	----------------------------------	-----------	------

Output

sy	String output value	string
----	---------------------	--------

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 `su`. Precision and format are specified by the `prec` and `mode` parameters.

Input

<code>u</code>	Analog input of the block	<code>double</code>
----------------	---------------------------	---------------------

Parameters

<code>prec</code>	Precision (number of digits)	$\downarrow 0 \uparrow 20$	<code>long</code>
<code>mode</code>	Output string format	$\odot 1$	<code>long</code>
	1 . . . . . best fit		
	2 . . . . . normal		
	3 . . . . . exponencial		

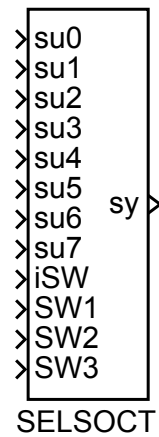
Output

<code>sy</code>	String output value	<code>string</code>
-----------------	---------------------	---------------------

## SELSOCT – String selector

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

### Inputs

su0...7	String input value	string
iSW	Active signal selector	long
SW1...3	Binary signal selector	bool

Parameters

<code>BINF</code>	Enable the binary selectors	<code>bool</code>
<code>nmax</code>	Allocated size of string [bytes]	<code>↓0 ↑65520 long</code>

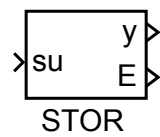
Output

<code>sy</code>	The selected input signal	<code>string</code>
-----------------	---------------------------	---------------------

STOR – String to real number conversion

Block Symbol

Licence: [STANDARD](#)



Function Description

The **STOR** converts a string in **su** into a real number in **y**. An error is signaled in **E** if unsuccessful.

Input

<code>su</code>	String input value	<code>string</code>
-----------------	--------------------	---------------------

Parameter

<code>yerr</code>	Substitute value for an error case	<code>double</code>
-------------------	------------------------------------	---------------------

Outputs

<code>y</code>	Analog output of the block	<code>double</code>
<code>E</code>	Error indicator	<code>bool</code>





## Chapter 12

# PARAM – Blocks for parameter handling

### Contents

---

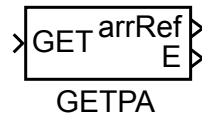
GETPA – Block for remote array parameter acquirement . . . . .	318
GETPR, GETPI, GETPB – Blocks for remote parameter acquirement . . . . .	320
GETPS – * Block for remote string parameter acquirement . . . . .	322
PARA – Block with input-defined array parameter . . . . .	323
PARR, PARI, PARB – Blocks with input-defined parameter . . . . .	324
PARS – * Block with input-defined string parameter . . . . .	326
SETPA – Block for remote array parameter setting . . . . .	327
SETPR, SETPI, SETPB – Blocks for remote parameter setting . . . . .	329
SETPS – * Block for remote string parameter setting . . . . .	331
SGSLP – Set, get, save and load parameters . . . . .	332
SILO – Save input value, load output value . . . . .	336
SILOS – Save input string, load output string . . . . .	338

---

## 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 at the level where the **GETPA** block is located. The string has to be prefixed with **'.'** in this case. Examples of relative paths: **".CNDR:yp"**, **".Lights.ATMT:touts"**.
- Absolute – complete sequence of hierarchic levels down to the block. For referring to blocks located in the driver task (see the [IOTASK](#) block for details on configuration) the **'&'** followed by the driver's name is used at the beginning of the absolute path. Examples of absolute paths: **"task1.inputs.ATMT:touts"**, **"&EfaDrv.measurements.CNDR:yp"**.

The order and names of individual hierarchic levels are displayed in a tree structure in the REXYGEN Diagnostics program.

### Input

<b>GET</b>	Input for initiating one-shot parameter read	<b>bool</b>
------------	--	-------------

### Outputs

<b>arrRef</b>	Array reference	<b>reference</b>
<b>E</b>	Error flag	<b>bool</b>

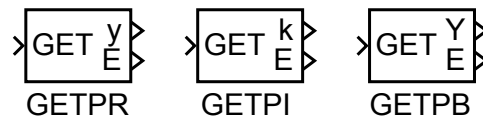
## Parameters

<b>sc</b>	String connection to the parameter	<b>string</b>
<b>GETF</b>	Get parameter only when forced to	<b>bool</b>
	<b>off</b> ... Remote parameter is continuously read	
	<b>on</b> .... One-shot mode, the remote parameter is read only when forced to by the <b>GET</b> input (rising edge)	
<b>nmax</b>	Maximum size of array	⊙256 <b>long</b>

## GETPR, GETPI, GETPB – Blocks for remote parameter acquirement

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. The only difference among the three blocks is the type of parameter which they are acquiring. The **GETPR** block is used for obtaining real parameters, the **GETPI** block for integer parameters and the **GETPB** block for Boolean parameters.

The blocks operate in two modes, which are switched by the **GETF** parameter. For **GETF = off** the output *y* (or *k*, *Y*) is set to the value of the remote parameter at the start and every time when the remote parameter changes. If the **GETF** parameter is set to **on**, then the blocks work in single-shot read mode. In that case the remote parameter is read only when rising edge (**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 at the level where the **GETPR** block (or **GETPI**, **GETPB**) is located. The string has to be prefixed with `'.'` in this case. Examples of relative paths: `".GAIN:k"`, `".Motor1.Position:ycn"`.
- Absolute – complete sequence of hierarchic levels down to the block. For referring to blocks located in the driver task (see the [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.lin1:u2"`, `"&EfaDrv.measurements.DER1:n"`.

The order and names of individual hierarchic levels are displayed in a tree structure in the REXYGEN Diagnostics program.

### Input

<b>GET</b>	Input for initiating one-shot parameter read ( <b>off</b> → <b>on</b> )	<b>bool</b>
------------	---	-------------

## Outputs

y	Parameter value, output of the GETPR block	double
k	Parameter value, output of the GETPI block	long
Y	Parameter value, output of the GETPB block	bool
E	Error flag	bool
	off ... No error	
	on .... An error occurred	

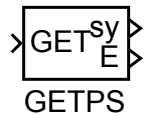
## Parameters

sc	String connection to the remote parameter respecting the above mentioned notation	string
GETF	Continuous or one-shot mode	bool
	off ... Remote parameter is continuously read	
	on .... One-shot mode, the remote parameter is read only when forced to by the GET input (rising edge)	

GETPS – \* **Block for remote string parameter acquirement**

Block Symbol

Licence: [STANDARD](#)



Function Description

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

Input

GET	Input for initiating one-shot parameter read	bool
-----	--	------

Parameters

sc	String connection to the parameter	string
GETF	Get parameter only when forced to	bool
	off ... Remote parameter is continuously read	
	on .... One-shot mode	
nmax	Allocated size of string	long

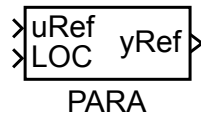
Outputs

sy	Parameter value	string
E	Error indicator	bool
	off ... No error	
	on .... An error occurred	

## PARA – Block with input-defined array parameter

Block Symbol

Licence: [STANDARD](#)



### Function Description

The **PARA** block allows, additionally to the standard way of parameter setting, changing one of its parameters by the input signal. The input-parameter pair is **uRef** and **apar**.

The Boolean input **LOC** (LOCAL) determines whether the value of the **apar** parameter is read from the input **uRef** or is input-independent (**LOC = on**). In the local mode **LOC = on** the parameter **apar** contains the last value of input **uRef** entering the block right before **LOC** was set to **on**.

The output value is equivalent the value of the parameter (**yRef = apar**).

### Inputs

<b>uRef</b>	Array reference	<b>reference</b>
<b>LOC</b>	Activation of local mode	<b>bool</b>
	<b>off</b> ... The parameter follows the input	
	<b>on</b> .... Local mode active	

### Output

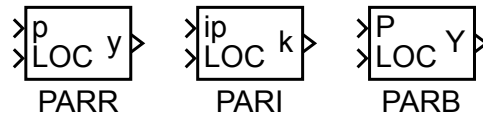
<b>yRef</b>	Array reference	<b>reference</b>
-------------	-----------------	------------------

### Parameters

<b>SETS</b>	Set array size flag. Use this flag to adjust the size of array when setting the parameter.	<b>bool</b>
<b>apar</b>	Initial value of the parameter	$\odot$ [0.0 1.0 2.0 3.0 4.0 5.0] <b>double</b>

## PARR, PARI, PARB – 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 and finally **P** and **PAR** for the PARB block.

The Boolean input **LOC** (LOCal) determines whether the value of the **par** (or **ipar**, **PAR**) parameter is read from the input **p** (or **ip**, **P**) or is input-independent (**LOC** = **on**). In the local mode **LOC** = **on** the parameter **par** (or **ipar**, **PAR**) contains the last value of input **p** (or **ip**, **P**) entering the block right before **LOC** was set to **on**.

The output value is equivalent 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**.

### Inputs

<b>p</b>	Parameter value (the PARR block)	double
<b>ip</b>	Parameter value (the PARI block)	long
<b>P</b>	Parameter value (the PARB block)	bool
<b>LOC</b>	Activation of local mode	bool
	<b>off</b> ... The parameter follows the input	
	<b>on</b> .... Local mode active	

### Output

<b>y</b>	Logical output of the PARR block	double
<b>k</b>	Logical output of the PARI block	long
<b>Y</b>	Logical output of the PARB block	bool

### Parameter

<b>par</b>	Initial value of the parameter (the PARR block)	⊙1.0	double
<b>ipar</b>	Initial value of the parameter (the PARI block)	⊙1	long
<b>PAR</b>	Initial value of the parameter (the PARB block)	⊙on	bool

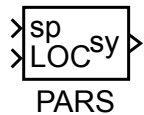


<b>SATF</b>	Activation of the saturation limits for the <b>PARR</b> and <b>PARI</b> blocks		<b>bool</b>
	<b>off</b> ... Signal not limited		
	<b>on</b> .... Saturation limits active		
<b>hilim</b>	Upper limit of the output signal (the <b>PARR</b> and <b>PARI</b> blocks)	⊙1.0	<b>double</b>
<b>lolim</b>	Lower limit of the output signal (the <b>PARR</b> and <b>PARI</b> blocks)	⊙-1.0	<b>double</b>

PARS – \* **Block with input-defined string parameter**

Block Symbol

Licence: [STANDARD](#)



Function Description

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

Inputs

sp	Parameter value	string
LOC	Activation of local mode	bool

Parameters

spar	Initial value of the parameter	string
nmax	Allocated size of string	long

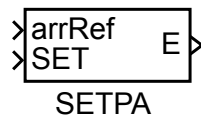
Output

sy	String output of the block	string
----	----------------------------	--------

## SETPA – Block for remote array parameter setting

Block Symbol

Licence: [STANDARD](#)



### Function Description

The **SETPA** block is used for setting the array parameters of other blocks in the model remotely. The block operates in two modes, which are switched by the **SETF** parameter. For **SETF** = **off** the remote parameter **cs** is set to the value of the input vector signal **arrRef** at the start and every time when the input signal changes. If the **SETF** parameter is set to **on**, then the block works in one-shot write mode. In that case the remote parameter is set only when rising edge (**off**→**on**) occurs at the **SET** input.

The name of the remote parameter is determined by the string parameter **sc** (string connection), which has the form **<block\_path:parameter\_name>**. The path to the block whose parameter should be read can contain hierarchic levels separated by dots followed by the block name. The path can be either relative or absolute:

- Relative – starts at the level where the **GETPA** block is located. The string has to be prefixed with **'.'** in this case. Examples of relative paths: **".CNDR:yp"**, **".Lights.ATMT:touts"**.
- Absolute – complete sequence of hierarchic levels down to the block. For referring to blocks located in the driver task (see the **IOTASK** block for details on configuration) the **'&'** followed by the driver's name is used at the beginning of the absolute path. Examples of absolute paths: **"task1.inputs.ATMT:touts"**, **"&EfaDrv.measurements.CNDR:yp"**.

The order and names of individual hierarchic levels are displayed in a tree structure in the REXYGEN Diagnostics program.

### Inputs

<b>arrRef</b>	Array reference	<b>reference</b>
<b>SET</b>	Input for initiating one-shot parameter write	<b>bool</b>

### Output

<b>E</b>	Error flag	<b>bool</b>
----------	------------	-------------

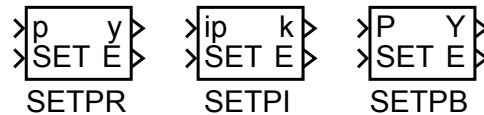
## Parameters

<code>sc</code>	String connection to the parameter	<code>string</code>
<code>SETF</code>	Continuous or one-shot mode	<code>bool</code>
	<code>off</code> ... Remote parameter is continuously updated	
	<code>on</code> .... One-shot mode, the remote parameter is updated only when forced to by the <code>SET</code> input (rising edge)	
<code>SETS</code>	Set array size flag. Use this flag to adjust the size of array when setting the parameter.	<code>bool</code>

## SETPR, SETPI, SETPB – Blocks for remote parameter setting

Block Symbols

Licence: [STANDARD](#)



### Function Description

The **SETPR**, **SETPI** and **SETPB** blocks are used for setting the parameters of other blocks in the model remotely. The only difference among the three blocks is the type of parameter which they are setting. The **SETPR** block is used for setting real parameters, the **SETPI** block for integer parameters and the **SETPB** block for Boolean parameters.

The blocks operate in two modes, which are switched by the **SETF** parameter. For **SETF = off** the remote parameter **sc** is set to the value of the input signal **p** (or **ip**, **P**) at the start and every time when the input changes. If the **SETF** parameter is set to **on**, then the blocks work in one-shot write mode. In that case the remote parameter is set only when rising edge (**off**→**on**) occurs at the **SET** input. Successful modification of the remote parameter is indicated by zero error output **E = off** and the output **y** (or **k**, **Y**) is set to the value of the modified parameter. The error output is set to **E = on** in case of write error.

The name of the remote parameter is determined by the string parameter **sc** (string connection), which has the form `<block_path:parameter_name>`. It is also possible to access individual items of array-type parameters (e.g. the **tout** parameter of the [ATMT](#) block). This can be achieved using the square brackets and item number, e.g. `.ATMT:touts[2]`. The items are numbered from zero, thus the string connection stated above refers to the third element of the array.

The path to the block whose parameter should be set can contain hierarchic levels separated by dots followed by the block name. The path can be either relative or absolute:

- Relative – starts at the level where the **SETPR** block (or **SETPI**, **SETPB**) is located. The string has to be prefixed with `'.'` in this case. Examples of relative paths: `".GAIN:k"`, `".Motor1.Position:ycn"`.
- Absolute – complete sequence of hierarchic levels down to the block. For referring to blocks located in the driver task (see the [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.lin1:u2"`, `"&EfaDrv.measurements.DER1:n"`.

The order and names of individual hierarchic levels are displayed in a tree structure in the REXYGEN Diagnostics program.

## Inputs

<code>p</code>	Desired parameter value at the SETPR block input	double
<code>ip</code>	Desired parameter value at the SETPI block input	long
<code>P</code>	Desired parameter value at the SETPB block input	bool
<code>SET</code>	Input for initiating one-shot parameter write ( <code>off</code> → <code>on</code> )	bool

## Outputs

<code>y</code>	Parameter value (the SETPR block)	double
<code>k</code>	Parameter value (the SETPI block)	long
<code>Y</code>	Parameter value (the SETPB block)	bool
<code>E</code>	Error flag	bool
	<code>off</code> ... No error	
	<code>on</code> .... An error occurred	

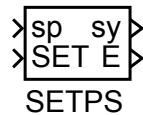
## Parameters

<code>sc</code>	String connection to the remote parameter respecting the above mentioned notation	string
<code>SETF</code>	Continuous or one-shot mode	bool
	<code>off</code> ... Remote parameter is continuously updated	
	<code>on</code> .... One-shot mode, the remote parameter is updated only when forced to by the SET input (rising edge)	

## SETPS — \* Block for remote string parameter setting

Block Symbol

Licence: [STANDARD](#)



### Function Description

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

### Inputs

<code>sp</code>	Desired parameter value	<code>string</code>
<code>SET</code>	Input for initiating one-shot parameter write	<code>bool</code>

### Parameters

<code>sc</code>	String connection to the parameter	<code>string</code>
<code>SETF</code>	Set parameter only when forced to	<code>bool</code>
<code>nmax</code>	Allocated size of string	<code>long</code>

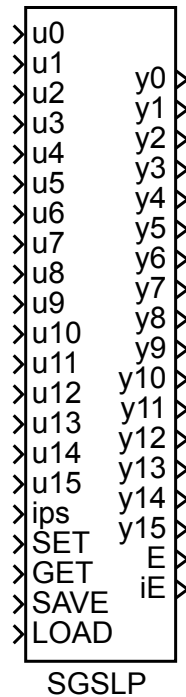
### Outputs

<code>sy</code>	Parameter value	<code>string</code>
<code>E</code>	Error indicator	<code>bool</code>

## 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 **<blok>:<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 **<blok>** 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 **<blok>** 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.

### Inputs

<b>ui</b>	$i$ -th analog input signal, $i = 0, \dots, 15$	<b>double</b>
<b>ips</b>	Parameter set index (numbered from zero)	<b>long</b>
<b>SET</b>	Set the parameters of the <b>ips</b> parameter set according to the values of the <b>ui</b> inputs. The values can be found at the <b>yi</b> outputs after a successful operation.	<b>bool</b>
<b>GET</b>	Get the parameters of the <b>ips</b> parameter set. The values can be found at the <b>yi</b> outputs after a successful operation.	<b>bool</b>
<b>SAVE</b>	Save the <b>ips</b> parameter set to a file on the target device	<b>bool</b>
<b>LOAD</b>	Load the <b>ips</b> parameter set from a file on the target device	<b>bool</b>

### Outputs

<b>yi</b>	$i$ -th analog output signal, $i = 0, \dots, 15$	<b>double</b>
<b>E</b>	Error flag	<b>bool</b>
	<b>off</b> ... No error	
	<b>on</b> .... An error occurred (see <b>iE</b> )	

<b>iE</b>	Error or warning code of the last operation	<b>long</b>
0	..... Operation successful	
1	..... Fatal error of the Matlab system (only in Simulink), the block is no longer executed	
2	..... Error opening the file for reading (LOAD operation)	
3	..... Error opening the file for writing (SAVE operation)	
4	..... Incorrect file format	
5	..... The <b>ips</b> parameter set not found in the file	
6	..... Parameter not found in the configuration, name mismatch (LOAD operation)	
7	..... Unexpected end of file	
8	..... Error writing to file (disk full?)	
9	..... Parameter syntax error (the ':' character not found)	
10	..... Only whitespace in the parameter name	
11	..... Error creating the backup file	
12	..... Error obtaining the parameter value by the GET operation (non-existing parameter?)	
13	..... Error setting the parameter value by the SET operation (non-existing parameter?)	
14	..... Timeout during obtaining/setting the parameter	
15	..... The specified parameter is read-only	
16	..... The <b>ips</b> parameter is out of range	

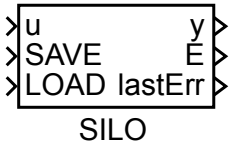
## Parameters

<b>nps</b>	Number of parameter sets	$\downarrow 1 \uparrow 16 \odot 1$	<b>long</b>
<b>ips0</b>	Index of parameter set to load and set during the block initialization. No set is read for $\text{ips0} < 0$ or $\text{ips0} \geq \text{nps}$	$\downarrow -1 \uparrow 15$	<b>long</b>
<b>iprec</b>	Precision (number of digits) for storing the values of <b>double</b> type in a file	$\downarrow 2 \uparrow 15 \odot 12$	<b>long</b>
<b>icolw</b>	Requested column width in the status file. Spaces are appended to the parameter value when necessary.	$\downarrow 0 \uparrow 22$	<b>long</b>
<b>fname</b>	Name of the file the <b>SAVE</b> and <b>LOAD</b> operations work with	$\odot \text{status}$	<b>string</b>
<b>broot</b>	Root block in hierarchy, inserted at the beginning of all <b>sci</b> parameters, see the description above	$\odot .$	<b>string</b>
<b>sci</b>	Strings defining the connection of <b>ui</b> inputs and <b>yi</b> outputs to the parameters, $i = 0, \dots, 15$ , see details above		<b>string</b>

## 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.

### Inputs

<b>u</b>	Input signal	<b>double</b>
<b>SAVE</b>	Save value to file	<b>bool</b>
<b>LOAD</b>	Load value from file	<b>bool</b>

### Parameters

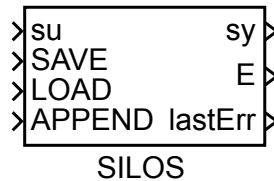
<b>fname</b>	Name of persistent storage file	<b>string</b>
<b>CSF</b>	Flag for continuous saving	<b>bool</b>
<b>CLF</b>	Flag for continuous loading	<b>bool</b>
<b>yerr</b>	Substitute value for an error case	<b>double</b>

## Outputs

<code>y</code>	Output signal	<code>double</code>
<code>E</code>	Error flag	<code>bool</code>
<code>lastErr</code>	Result of last operation	<code>long</code>

## SILOS – Save input string, load output string

Block Symbol

Licence: [STANDARD](#)

### Function Description

The **SILOS** block can be used to export or import a string to/from a file. The string is saved when a rising edge (**off**→**on**) occurs at the **SAVE** input and the string is also set to the **sy** output. The string is loaded at startup and when a rising edge (**off**→**on**) occurs at the **LOAD** input.

If a logical true (**on**) is brought to the **APPEND** input, the input string is added to the end of the file when it is saved. This mode is useful for logging events into text files. This input signal has no effect on loading from the file.

The **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.

### Inputs

<b>su</b>	String input of the block	⊙0	<b>string</b>
<b>SAVE</b>	Save string to file		<b>bool</b>
<b>LOAD</b>	Load string from file		<b>bool</b>
<b>APPEND</b>	Append saved string to file		<b>bool</b>

### Outputs

<b>sy</b>	String output of the block	<b>string</b>
-----------	----------------------------	---------------

<b>E</b>	Error indicator	<b>bool</b>
	<b>off</b> ... No error	
	<b>on</b> .... An error occurred	
<b>lastErr</b>	Result of last operation	<b>long</b>

## Parameters

<b>fname</b>	Name of persistent storage file	<b>string</b>
<b>CSF</b>	Continuous saving	<b>bool</b>
<b>CLF</b>	Continuous loading	<b>bool</b>
<b>LL0</b>	Last line only loading	<b>bool</b>
<b>nmax</b>	Allocated size of string	↓0 ↑65520 <b>long</b>





## Chapter 13

# MODEL – Dynamic systems simulation

### Contents

---

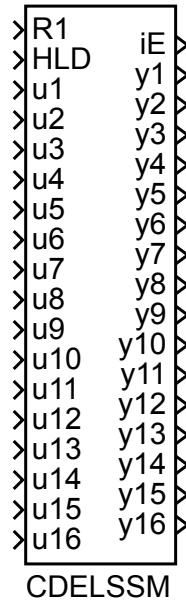
CDELSSM – Continuous state space model of a linear system with time delay . . . . .	342
CSSM – Continuous state space model of a linear system . . . . .	345
DDELSSM – Discrete state space model of a linear system with time delay . . . . .	348
DSSM – Discrete state space model of a linear system . . . . .	350
FMUCS – * Import modelu FMU CS (pro Co-Simulation) . . . . .	352
FMUINFO – * Informace o importovaném modelu FMU . . . . .	355
FOPDT – First order plus dead-time model . . . . .	356
MDL – Process model . . . . .	357
MDLI – Process model with input-defined parameters . . . . .	358
MVD – Motorized valve drive . . . . .	359
SOPDT – Second order plus dead-time model . . . . .	360

---

## CDELSSM – Continuous state space model of a linear system 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.

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.

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 [\[5\]](#), which uses Padé approximations of matrix exponential and its integral and scaling technique.

During the real-time simulation, single simulation step of the above discrete state space model is computed in each execution time instant.

## Inputs

<b>R1</b>	Reset signal. When <b>R1</b> = <b>on</b> , the state vector <b>x</b> is set to its initial value <b>x0</b> . The simulation continues on the falling edge of <b>R1</b> ( <b>on</b> → <b>off</b> ).	<b>bool</b>
<b>HLD</b>	Simulation output holds its value if <b>HLD</b> = <b>on</b> .	<b>bool</b>
<b>u1..u16</b>	Simulated system inputs. First $m$ simulation inputs are used where $m$ is the number of columns of the matrix <b>Bc</b> .	<b>double</b>

## Outputs

<b>iE</b>	Block error code 0 ..... O.K., the simulation runs correctly -213 .. incompatibility of the state space model matrices dimensions -510 .. the model is badly conditioned (some working matrix is singular or nearly singular) xxx ... error code <b>xxx</b> of <b>REXYGEN</b> , see <a href="#">appendix C</a> for details	<b>error</b>
<b>y1..y16</b>	Simulated system outputs. First $p$ simulation outputs are used where $p$ is the number of rows of the matrix <b>Cc</b> .	<b>double</b>

## Parameters

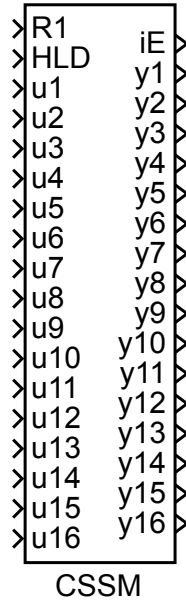
<b>UD</b>	Matrix <b>Dc</b> usage flag. If <b>UD</b> = <b>off</b> then the <b>Dc</b> matrix is not used for simulation (simulation behaves as if the <b>Dc</b> matrix is zero).	<b>bool</b>
<b>del</b>	Model time delay [s].	$\downarrow 0.0$ <b>double</b>

<b>is</b>	Order of the Padé approximation of the matrix exponential for the computation of the discretized system matrices. $\downarrow 0 \uparrow 4 \odot 2$	<b>long</b>
<b>eps</b>	Required accuracy of the Padé approximation. $\downarrow 0.0 \uparrow 1.0 \odot 0.0$	<b>double</b>
<b>Ac</b>	Matrix ( $n \times n$ ) of the continuous linear system dynamics.	<b>double</b>
<b>Bc</b>	Input matrix ( $n \times m$ ) of the continuous linear system.	<b>double</b>
<b>Cc</b>	Output matrix ( $p \times n$ ) of the continuous linear system.	<b>double</b>
<b>Dc</b>	Direct transmission (feedthrough) matrix ( $p \times m$ ) of the continuous linear system. The matrix is used only if the parameter UD=on. If UD=off, the dimensions of the Dc matrix are not checked.	<b>double</b>
<b>x0</b>	Initial value of the state vector (of dimension $n$ ) of the continuous linear system.	<b>double</b>

## CSSM – Continuous state space model of a linear system

Block Symbol

Licence: [ADVANCED](#)



### Function Description

The **CSSM** block (Continuous State Space Model) simulates behavior of a linear system

$$\begin{aligned}\frac{dx(t)}{dt} &= A_c x(t) + B_c u(t), \quad x(0) = x_0 \\ y(t) &= C_c x(t) + D_c u(t),\end{aligned}$$

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.

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 [5], which uses Padé approximations of matrix exponential and its integral and scaling technique.

During the real-time simulation, single simulation step of the above discrete state space model is computed in each execution time instant.

### Inputs

<b>R1</b>	Reset signal. When <b>R1</b> = on, the state vector <b>x</b> is set to its initial value <b>x0</b> . The simulation continues on the falling edge of <b>R1</b> (on→off).	<b>bool</b>
<b>HLD</b>	Simulation output holds its value if <b>HLD</b> =on.	<b>bool</b>
<b>u1..u16</b>	Simulated system inputs. First $m$ simulation inputs are used where $m$ is the number of columns of the matrix <b>Bc</b> .	<b>double</b>

### Outputs

<b>iE</b>	Block error code 0 ..... O.K., the simulation runs correctly -213 .. incompatibility of the state space model matrices dimensions -510 .. the model is badly conditioned (some working matrix is singular or nearly singular) xxx ... error code xxx of REXYGEN, see appendix C for details	<b>error</b>
<b>y1..y16</b>	Simulated system outputs. First $p$ simulation outputs are used where $p$ is the number of rows of the matrix <b>Cc</b> .	<b>double</b>

### Parameters

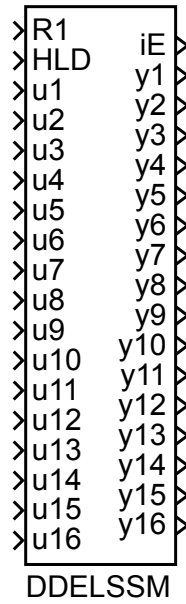
<b>UD</b>	Matrix <b>Dc</b> usage flag. If <b>UD</b> =off then the <b>Dc</b> matrix is not used for simulation (simulation behaves as if the <b>Dc</b> matrix is zero).	<b>bool</b>
<b>is</b>	Order of the Padé approximation of the matrix exponential for the computation of the discretized system matrices.      ↓0 ↑4 ⊙2	<b>long</b>
<b>eps</b>	Required accuracy of the Padé approximation.      ↓0.0 ↑1.0 ⊙0.0	<b>double</b>
<b>Ac</b>	Matrix ( $n \times n$ ) of the continuous linear system dynamics.	<b>double</b>
<b>Bc</b>	Input matrix ( $n \times m$ ) of the continuous linear system.	<b>double</b>
<b>Cc</b>	Output matrix ( $p \times n$ ) of the continuous linear system.	<b>double</b>

<b>Dc</b>	Direct transmission (feedthrough) matrix ( $p \times m$ ) of the continuous linear system. The matrix is used only if the parameter <b>UD=on</b> . If <b>UD=off</b> , the dimensions of the <b>Dc</b> matrix are not checked.	<b>double</b>
<b>x0</b>	Initial value of the state vector (of dimension $n$ ) of the continuous linear system.	<b>double</b>

## DDELSSM – Discrete state space model of a linear system 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. 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.



## Inputs

R1	Reset signal. When R1 = on, the state vector $\mathbf{x}$ is set to its initial value $\mathbf{x0}$ . The simulation continues on the falling edge of R1 (on→off).	bool
HLD	Simulation output holds its value if HLD=on.	bool
u1..u16	Simulated system inputs. First $m$ simulation inputs are used where $m$ is the number of columns of the matrix Bd.	double

## Outputs

iE	Block error code 0 ..... O.K., the simulation runs correctly -213 .. incompatibility of the state space model matrices dimensions xxx ... error code xxx of REXYGEN, see appendix C for details	error
y1..y16	Simulated system outputs. First $p$ simulation outputs are used where $p$ is the number of rows of the matrix Cd.	double

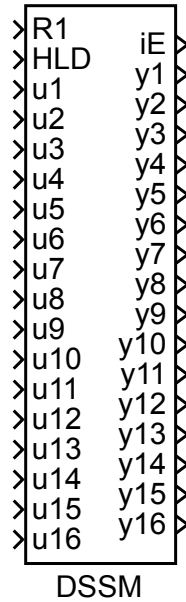
## Parameters

UD	Matrix Dd usage flag. If UD=off then the Dd matrix is not used for simulation (simulation behaves as if the Dd matrix is zero).	bool
del	Model time delay [s].	↓0.0 double
Ad	Matrix ( $n \times n$ ) of the discrete linear system dynamics.	double
Bd	Input matrix ( $n \times m$ ) of the discrete linear system.	double
Cd	Output matrix ( $p \times n$ ) of the discrete linear system.	double
Dd	Direct transmission (feedthrough) matrix ( $p \times m$ ) of the discrete linear system. The matrix is used only if the parameter UD=on. If UD=off, the dimensions of the Dd matrix are not checked.	double
x0	Initial value of the state vector (of dimension $n$ ) of the discrete linear system.	double

## DSSM – Discrete state space model of a linear system

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.

All matrices are specified in the same format as in Matlab, i.e. the whole matrix is placed in brackets, elements are entered by rows, elements of a row are separated by spaces (blanks), rows are separated by semicolons. The  $x_0$  vector is a column, therefore the elements are separated by semicolons (each element is in a separate row).

During the real-time simulation, single simulation step of the above discrete state space model is computed in each execution time instant.

## Inputs

R1	Reset signal. When R1 = on, the state vector $\mathbf{x}$ is set to its initial value $\mathbf{x0}$ . The simulation continues on the falling edge of R1 (on→off).	bool
HLD	Simulation output holds its value if HLD=on.	bool
u1..u16	Simulated system inputs. First $m$ simulation inputs are used where $m$ is the number of columns of the matrix Bd.	double

## Outputs

iE	Block error code 0 ..... O.K., the simulation runs correctly -213 .. incompatibility of the state space model matrices dimensions xxx ... error code xxx of REXYGEN, see appendix C for details	error
y1..y16	Simulated system outputs. First $p$ simulation outputs are used where $p$ is the number of rows of the matrix Cd.	double

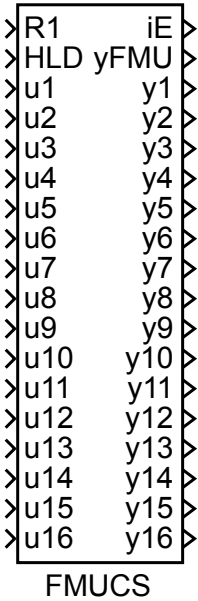
## Parameters

UD	Matrix Dd usage flag. If UD=off then the Dd matrix is not used for simulation (simulation behaves as if the Dd matrix is zero).	bool
Ad	Matrix ( $n \times n$ ) of the discrete linear system dynamics.	double
Bd	Input matrix ( $n \times m$ ) of the discrete linear system.	double
Cd	Output matrix ( $p \times n$ ) of the discrete linear system.	double
Dd	Direct transmission (feedthrough) matrix ( $p \times m$ ) of the discrete linear system. The matrix is used only if the parameter UD=on. If UD=off, the dimensions of the Dd matrix are not checked.	double
x0	Initial value of the state vector (of dimension $n$ ) of the discrete linear system.	double

FMUCS – \* Import modelu FMU CS (pro Co-Simulation)

Block Symbol

Licence: [ADVANCED](#)



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

R1	Reset bloku	bool
HLD	Podržení aktuálního stavu modelu	bool
u1	Analogový vstupní signál	double
u2	Analogový vstupní signál	double
u3	Analogový vstupní signál	double
u4	Analogový vstupní signál	double
u5	Analogový vstupní signál	double
u6	Analogový vstupní signál	double
u7	Analogový vstupní signál	double
u8	Analogový vstupní signál	double
u9	Analogový vstupní signál	double
u10	Analogový vstupní signál	double
u11	Analogový vstupní signál	double

u12	Analogový vstupní signál	double
u13	Analogový vstupní signál	double
u14	Analogový vstupní signál	double
u15	Analogový vstupní signál	double
u16	Analogový vstupní signál	double

## Parameters

tstop	Koncový čas simulace	$\downarrow 0.000001 \odot 1.0$	double
eps	Přesnost aproximace	$\downarrow 0.0 \uparrow 1.0 \odot 0.000001$	double
loglevel	Úroveň protokolování knihovny FMI do systémového logu	$\downarrow 0 \uparrow 7 \odot 2$	long
	0 ..... Nic		
	1 ..... Fatální		
	2 ..... Chyba		
	3 ..... Varování		
	4 ..... Info		
	5 ..... Podrobný		
	6 ..... Ladění		
	7 ..... Všechno		
SelPars	Seznam vybraných parametrů FMU		string
TUNEALLP	Považuj všechny vybrané parametry za laditelné parametry		bool
p1	Analogový parametr bloku		double
p2	Analogový parametr bloku		double
p3	Analogový parametr bloku		double
p4	Analogový parametr bloku		double
p5	Analogový parametr bloku		double
p6	Analogový parametr bloku		double
p7	Analogový parametr bloku		double
p8	Analogový parametr bloku		double
p9	Analogový parametr bloku		double
p10	Analogový parametr bloku		double
p11	Analogový parametr bloku		double
p12	Analogový parametr bloku		double
p13	Analogový parametr bloku		double
p14	Analogový parametr bloku		double
p15	Analogový parametr bloku		double
p16	Analogový parametr bloku		double

## Outputs

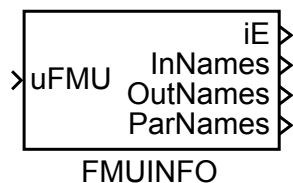
iE	Kód chyby	error
yFMU	Výstupní odkaz na instanci FMU	reference
y1	Analogový výstupní signál	double
y2	Analogový výstupní signál	double
y3	Analogový výstupní signál	double

y4	Analogový výstupní signál	double
y5	Analogový výstupní signál	double
y6	Analogový výstupní signál	double
y7	Analogový výstupní signál	double
y8	Analogový výstupní signál	double
y9	Analogový výstupní signál	double
y10	Analogový výstupní signál	double
y11	Analogový výstupní signál	double
y12	Analogový výstupní signál	double
y13	Analogový výstupní signál	double
y14	Analogový výstupní signál	double
y15	Analogový výstupní signál	double
y16	Analogový výstupní signál	double

## FMUINFO — \* Informace o importovaném modelu FMU

Block Symbol

Licence: [ADVANCED](#)



### 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.

### Vstup

<b>uFMU</b>	Vstupní odkaz na instanci FMU	<b>reference</b>
-------------	-------------------------------	------------------

### Parameters

<b>SelPars</b>	Seznam vybraných parametrů FMU	<b>string</b>
<b>Separ</b>	Oddělovač jmen v řetězcových výstupech	⊙, <b>string</b>

### Outputs

<b>iE</b>	Kód chyby	<b>error</b>
<b>InNames</b>	Seznam jmen vstupů FMU	<b>string</b>
<b>OutNames</b>	Seznam jmen výstupů FMU	<b>string</b>
<b>ParNames</b>	Seznam jmen vybraných parametrů FMU	<b>string</b>

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:

$$P(s) = \frac{k0}{(\tau \cdot s + 1)} \cdot e^{-del \cdot s}$$

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.

Input

u	Analog input of the block	double
---	---------------------------	--------

Output

y	Analog output of the block	double
---	----------------------------	--------

Parameters

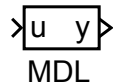
k0	Static gain	⊙1.0	double
del	Dead time [s]		double
tau	Time constant	⊙1.0	double
nmax	Size of delay buffer (number of samples) for the time delay del. Used for internal memory allocation.	↓10 ↑100000000 ⊙1000	long



## 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**.

### Input

<b>u</b>	Analog input of the block	double
----------	---------------------------	--------

### Output

<b>y</b>	Analog output of the block	double
----------	----------------------------	--------

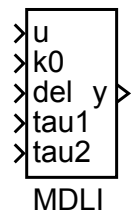
### Parameters

<b>k0</b>	Static gain	⊙1.0 double
<b>del</b>	Dead time [s]	double
<b>tau1</b>	The first time constant	⊙1.0 double
<b>tau2</b>	The second time constant	⊙2.0 double
<b>nmax</b>	Size of delay buffer (number of samples) for the time delay <b>del</b> . Used for internal memory allocation.	long ↓10 ↑10000000 ⊙1000

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.

Inputs

<code>u</code>	Analog input of the block	<code>double</code>
<code>k0</code>	Static gain	<code>double</code>
<code>del</code>	Dead time [s]	<code>double</code>
<code>tau1</code>	The first time constant	<code>double</code>
<code>tau2</code>	The second time constant	<code>double</code>

Output

<code>y</code>	Analog output of the block	<code>double</code>
----------------	----------------------------	---------------------

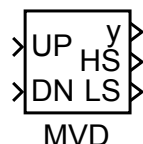
Parameters

<code>nmax</code>	Size of delay buffer (number of samples) for the time delay <code>del</code> . Used for internal memory allocation.	<code>long</code> ↓10 ↑100000000 ◎1000
-------------------	---	---

## 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).

### Inputs

UP	Open	bool
DN	Close	bool

### Outputs

y	Valve position	double
HS	Upper end switch	bool
LS	Lower end switch	bool

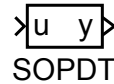
### Parameters

y0	Initial valve position	double
tv	Time required for transition between $y = 0$ and $y = 1$ [s]	$\odot 10.0$ double
hilim	Upper limit position (open)	$\odot 1.0$ double
lolim	Lower limit position (closed)	double

## 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.

$$\begin{aligned} \text{itf} = 1: \quad P(s) &= \frac{\text{pb1} \cdot s + \text{pb0}}{s^2 + \text{pa1} \cdot s + \text{pa0}} \cdot e^{-\text{del} \cdot s} \\ \text{itf} = 2: \quad P(s) &= \frac{k0 (\text{tau} \cdot s + 1)}{(\text{tau1} \cdot s + 1) (\text{tau2} \cdot s + 1)} \cdot e^{-\text{del} \cdot s} \\ \text{itf} = 3: \quad P(s) &= \frac{k0 \cdot \text{om}^2 \cdot (\text{tau}/\text{om} \cdot s + 1)}{(s^2 + 2 \cdot \text{xi} \cdot \text{om} \cdot s + \text{om}^2)} \cdot e^{-\text{del} \cdot s} \\ \text{itf} = 4: \quad P(s) &= \frac{k0 (\text{tau} \cdot s + 1)}{(\text{tau1} \cdot s + 1) s} \cdot e^{-\text{del} \cdot s} \end{aligned}$$

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.

### Input

<b>u</b>	Analog input of the block	double
----------	---------------------------	--------

### Output

<b>y</b>	Analog output of the block	double
----------	----------------------------	--------

## Parameters

<code>itf</code>	Transfer function form	⊙1	long
	1 ..... A general form		
	2 ..... A form with real poles		
	3 ..... A form with complex poles		
	4 ..... A form with integrator		
<code>k0</code>	Static gain	⊙1.0	double
<code>tau</code>	Numerator time constant		double
<code>tau1</code>	The first time constant	⊙1.0	double
<code>tau2</code>	The second time constant	⊙1.0	double
<code>om</code>	Natural frequency	⊙1.0	double
<code>xi</code>	Relative damping coefficient	⊙1.0	double
<code>pb0</code>	Numerator coefficient: $s^0$	⊙1.0	double
<code>pb1</code>	Numerator coefficient: $s^1$	⊙1.0	double
<code>pa0</code>	Denominator coefficient: $s^0$	⊙1.0	double
<code>pa1</code>	Denominator coefficient: $s^1$	⊙1.0	double
<code>del</code>	Dead time [s]		double
<code>nmax</code>	Size of delay buffer (number of samples) for the time delay <code>del</code> . Used for internal memory allocation.		long
		↓10 ↑100000000 ⊙1000	



## Chapter 14

# MATRIX – Blocks for matrix and vector operations

### Contents

---

CNA – Array (vector/matrix) constant . . . . .	365
MB_DASUM – Sum of the absolute values . . . . .	366
MB_DAXPY – Performs $y := a*x + y$ for vectors $x, y$ . . . . .	368
MB_DCOPY – Copies vector $x$ to vector $y$ . . . . .	370
MB_DDOT – Dot product of two vectors . . . . .	372
MB_DGEMM – Performs $C := \alpha*op(A)*op(B) + \beta*C$ , where $op(X) = X$ or $op(X) = X^T$ . . . . .	374
MB_DGEMV – Performs $y := \alpha*A*x + \beta*y$ or $y := \alpha*A^T*x + \beta*y$ . . . . .	376
MB_DGER – Performs $A := \alpha*x*y^T + A$ . . . . .	378
MB_DNRM2 – Euclidean norm of a vector . . . . .	380
MB_DROT – Plain rotation of a vector . . . . .	382
MB_DSCAL – Scales a vector by a constant . . . . .	384
MB_DSWAP – Interchanges two vectors . . . . .	386
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 $A$ . . . . .	388
MB_DTRMV – Performs $x := A*x$ or $x := A^T*x$ for triangular matrix $A$ . . . . .	390
MB_DTRSV – Solves one of the system of equations $A*x = b$ or $A^T*x = b$ for triangular matrix $A$ . . . . .	392
ML_DGEBAK – Backward transformation to ML_DGEBAL of left or right eigenvectors . . . . .	394
ML_DGEBAL – Balancing of a general real matrix . . . . .	396
ML_DGEBRD – Reduces a general real matrix to bidiagonal form by an orthogonal transformation . . . . .	399

ML_DGECON – Estimates the reciprocal of the condition number of a general real matrix . . . . .	401
ML_DGEES – Computes the eigenvalues, the Schur form, and, optionally, the matrix of Schur vectors . . . . .	403
ML_DGEEV – Computes the eigenvalues and, optionally, the left and/or right eigenvectors . . . . .	405
ML_DGEHRD – Reduces a real general matrix A to upper Hessenberg form . . . . .	407
ML_DGELQF – Computes an LQ factorization of a real M-by-N matrix A . . . . .	409
ML_DGELS – Computes the minimum-norm solution to a real linear least squares problem . . . . .	411
ML_DGEQRF – Computes a QR factorization of a real M-by-N matrix A . . . . .	413
ML_DGESDD – Computes the singular value decomposition (SVD) of a real M-by-N matrix A . . . . .	415
ML_DTRSYL – Solves the real Sylvester matrix equation for quasi-triangular matrices A and B . . . . .	417
MX_CTODPA – Discretizes continuous model given by (A,B) to (Ad,Bd) using Pade approximations . . . . .	419
MX_DIM – Matrix/Vector dimensions . . . . .	421
MX_DSAGET – Set subarray of A into B . . . . .	422
MX_DSAREF – Set reference to subarray of A into B . . . . .	424
MX_DSASET – Set A into subarray of B . . . . .	426
MX_DTRNSP – General matrix transposition: $B := \alpha * A^T$ . . . . .	428
MX_DTRNSQ – Square matrix in-place transposition: $A := \alpha * A^T$ . . . . .	430
MX_FILL – Fill real matrix or vector . . . . .	432
MX_MAT – Matrix data storage block . . . . .	433
MX_RAND – Randomly generated matrix or vector . . . . .	434
MX_REFCOPY – Copies input references of matrices A and B to their output references . . . . .	436
MX_VEC – Vector data storage block . . . . .	437
MX_WRITE – Write a Matrix/Vector to the console/system log . . . . .	438
RTOV – Vector multiplexer . . . . .	440
SWVMR – Vector/matrix/reference signal switch . . . . .	442
VTOR – Vector demultiplexer . . . . .	443

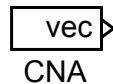
---



## 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**. If the parameter **TRN** = **on** then the output reference **vec** contains transposed data.

### Parameters

<b>filename</b>	CSV data file		<b>string</b>
<b>TRN</b>	Transpose loaded matrix		<b>bool</b>
<b>nmax</b>	Allocated size of array	↓2 ↑100000000 ⊙100	<b>long</b>
<b>etype</b>	Type of elements	⊙8	<b>long</b>
	1 .... Bool		
	2 .... Byte		
	3 .... Short		
	4 .... Long		
	5 .... Word		
	6 .... DWord		
	7 .... Float		
	8 .... Double		
	-- ....		
	10 .... Large		
<b>acn</b>	Initial array value	⊙[0 1 2 3]	<b>double</b>

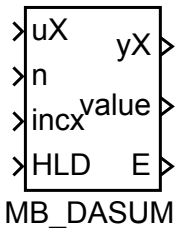
### Output

<b>vec</b>	Reference to vector/matrix data	<b>reference</b>
------------	---------------------------------	------------------

### 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 [\[6\]](#) for more details.

### Inputs

<b>uX</b>	Input reference to vector x	<b>reference</b>
<b>n</b>	Number of processed vector elements	<b>long</b>
<b>incx</b>	Index increment of vector x	<b>long</b>
<b>HLD</b>	Hold	<b>bool</b>

### Outputs

<b>yX</b>	Output reference to vector x	<b>reference</b>
-----------	------------------------------	------------------

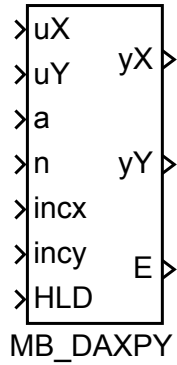
<b>value</b>	Return value of the function
<b>E</b>	Error flag

<b>double</b>
<b>bool</b>

**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 [\[6\]](#) for more details.

## Inputs

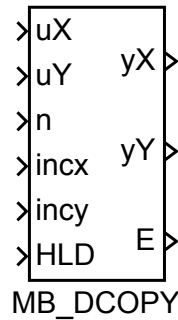
uX	Input reference to vector x	reference
uY	Input reference to vector y	reference
a	Scalar coefficient a	double
n	Number of processed vector elements	long
incx	Index increment of vector x	long
incy	Index increment of vector y	long
HLD	Hold	bool

## Outputs

yX	Output reference to vector x	reference
yY	Output reference to vector y	reference
E	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  $incx \neq 0$  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 [\[6\]](#) for more details.

## Inputs

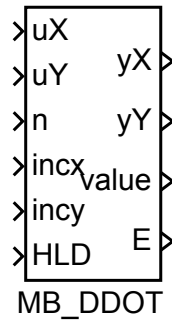
<code>uX</code>	Input reference to vector x	<code>reference</code>
<code>uY</code>	Input reference to vector y	<code>reference</code>
<code>n</code>	Number of processed vector elements	<code>long</code>
<code>incx</code>	Index increment of vector x	<code>long</code>
<code>incy</code>	Index increment of vector y	<code>long</code>
<code>HLD</code>	Hold	<code>bool</code>

## Outputs

<code>yX</code>	Output reference to vector x	<code>reference</code>
<code>yY</code>	Output reference to vector y	<code>reference</code>
<code>E</code>	Error indicator	<code>bool</code>

**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 ≠ 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 [\[6\]](#) for more details.



## Inputs

<code>uX</code>	Input reference to vector x	<code>reference</code>
<code>uY</code>	Input reference to vector y	<code>reference</code>
<code>n</code>	Number of processed vector elements	<code>long</code>
<code>incx</code>	Index increment of vector x	<code>long</code>
<code>incy</code>	Index increment of vector y	<code>long</code>
<code>HLD</code>	Hold	<code>bool</code>

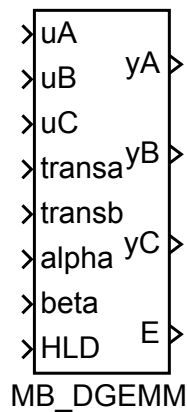
## Outputs

<code>yX</code>	Output reference to vector x	<code>reference</code>
<code>yY</code>	Output reference to vector y	<code>reference</code>
<code>value</code>	Return value of the function	<code>double</code>
<code>E</code>	Error indicator	<code>bool</code>

**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  $op(A)$  and  $op(B)$  have to be multipliable).
- the call of the function `DGEMM` returns error using the function `XERBLA`, see the system log.

See BLAS documentation [\[6\]](#) for more details.

## Inputs

<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>transa</code>	Transposition of matrix A	↓0 ↑3	long
<code>transb</code>	Transposition of matrix B	↓0 ↑3	long
<code>alpha</code>	Scalar coefficient alpha		double
<code>beta</code>	Scalar coefficient beta		double
<code>HLD</code>	Hold		bool

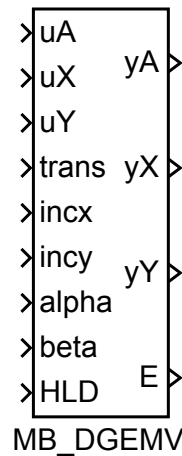
## Outputs

<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>E</code>	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 [\[6\]](#) for more details.

## Inputs

<code>uA</code>	Input reference to matrix A		reference
<code>uX</code>	Input reference to vector x		reference
<code>uY</code>	Input reference to vector y		reference
<code>trans</code>	Transposition of the input matrix	↓0 ↑3	long
<code>incx</code>	Index increment of vector x		long
<code>incy</code>	Index increment of vector y		long
<code>alpha</code>	Scalar coefficient alpha		double
<code>beta</code>	Scalar coefficient beta		double
<code>HLD</code>	Hold		bool

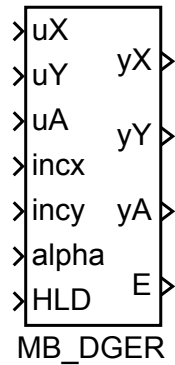
## Outputs

<code>yA</code>	Output reference to matrix A		reference
<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\_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 [6] for more details.

## Inputs

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
incy	Index increment of vector y	long
alpha	Scalar coefficient alpha	double
HLD	Hold	bool

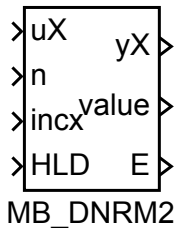
## Outputs

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 [\[6\]](#) for more details.

**Inputs**

<b>uX</b>	Input reference to vector x	<b>reference</b>
<b>n</b>	Number of processed vector elements	<b>long</b>
<b>incx</b>	Index increment of vector x	<b>long</b>
<b>HLD</b>	Hold	<b>bool</b>

**Outputs**

<b>yX</b>	Output reference to vector x	<b>reference</b>
-----------	------------------------------	------------------

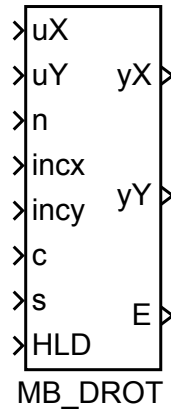


<code>value</code>	Return value of the function
<code>E</code>	Error indicator

<code>double</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 [\[6\]](#) for more details.

## Inputs

<code>uX</code>	Input reference to vector x	<code>reference</code>
<code>uY</code>	Input reference to vector y	<code>reference</code>
<code>n</code>	Number of processed vector elements	<code>long</code>
<code>incx</code>	Index increment of vector x	<code>long</code>
<code>incy</code>	Index increment of vector y	<code>long</code>
<code>c</code>	Scalar coefficient c	<code>double</code>
<code>s</code>	Scalar coefficient s	<code>double</code>
<code>HLD</code>	Hold	<code>bool</code>

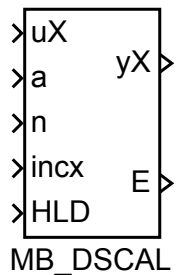
## Outputs

<code>yX</code>	Output reference to vector x	<code>reference</code>
<code>yY</code>	Output reference to vector y	<code>reference</code>
<code>E</code>	Error indicator	<code>bool</code>

**MB\_DSCAL – Scales a vector by a constant**

Block Symbol

Licence: [STANDARD](#)



**Function Description**

The output reference `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 [\[6\]](#) for more details.

**Inputs**

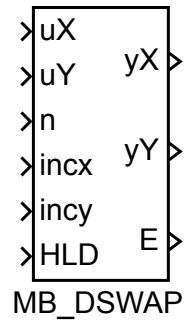
<code>uX</code>	Input reference to vector x	<b>reference</b>
<code>a</code>	Scalar coefficient a	<b>double</b>
<code>n</code>	Number of processed vector elements	<b>long</b>
<code>incx</code>	Index increment of vector x	<b>long</b>
<code>HLD</code>	Hold	<b>bool</b>

## Outputs

<code>yX</code>	Output reference to vector <code>x</code>	<code>reference</code>
<code>E</code>	Error indicator	<code>bool</code>

**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  $incx \neq 0$  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 [6] for more details.

## Inputs

<code>uX</code>	Input reference to vector x	<code>reference</code>
<code>uY</code>	Input reference to vector y	<code>reference</code>
<code>n</code>	Number of processed vector elements	<code>long</code>
<code>incx</code>	Index increment of vector x	<code>long</code>
<code>incy</code>	Index increment of vector y	<code>long</code>
<code>HLD</code>	Hold	<code>bool</code>

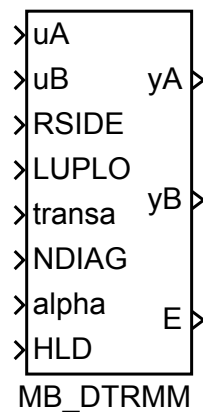
## Outputs

<code>yX</code>	Output reference to vector x	<code>reference</code>
<code>yY</code>	Output reference to vector y	<code>reference</code>
<code>E</code>	Error indicator	<code>bool</code>

**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 [\[6\]](#) for more details.

## Inputs

<code>uA</code>	Input reference to matrix A		<code>reference</code>
<code>uB</code>	Input reference to matrix B		<code>reference</code>
<code>RSIDE</code>	Operation is applied from right side		<code>bool</code>
<code>LUPL0</code>	Matrix A is a lower triangular matrix		<code>bool</code>
<code>transa</code>	Transposition of matrix A	↓0 ↑3	<code>long</code>
<code>NDIAG</code>	Matrix A is not assumed to be unit triangular		<code>bool</code>
<code>alpha</code>	Scalar coefficient alpha		<code>double</code>
<code>HLD</code>	Hold		<code>bool</code>

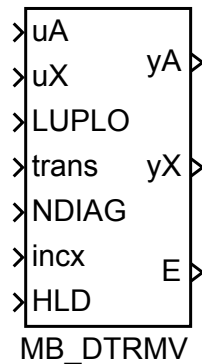
## Outputs

<code>yA</code>	Output reference to matrix A	<code>reference</code>
<code>yB</code>	Output reference to matrix B	<code>reference</code>
<code>E</code>	Error indicator	<code>bool</code>

**MB\_DTRMV** – Performs  $\mathbf{x} := \mathbf{A} * \mathbf{x}$  or  $\mathbf{x} := \mathbf{A}^T * \mathbf{x}$  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 [\[6\]](#) for more details.

## Inputs

<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
<code>NDIAG</code>	Matrix A is not assumed to be unit triangular		bool
<code>incx</code>	Index increment of vector x		long
<code>HLD</code>	Hold		bool

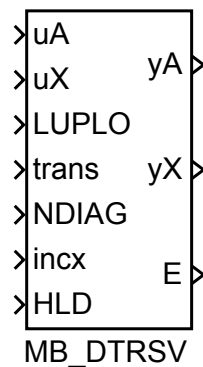
## Outputs

<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(sLUPLO, sTRANS, sNDIAG, N, uA, LDA, uX, INCX);
```

where parameters of **DTRSV** 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 [\[6\]](#) for more details.

## Inputs

<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
<code>NDIAG</code>	Matrix A is not assumed to be unit triangular		bool
<code>incx</code>	Index increment of vector x		long
<code>HLD</code>	Hold		bool

## Outputs

<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 [7] for more details.

## Inputs

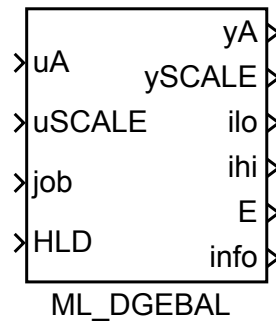
<code>uSCALE</code>	Input reference to vector <code>SCALE</code>	<code>reference</code>
<code>uV</code>	Reference to matrix of right or left eigenvectors to be transformed	<code>reference</code>
<code>job</code>	Type of backward transformation required	<code>long</code>
<code>RSIDE</code>	Operation is applied from right side	<code>bool</code>
<code>ilo</code>	Zero based low row and column index of working submatrix	<code>long</code>
<code>ihi</code>	Zero based high row and column index of working submatrix	<code>long</code>
<code>HLD</code>	Hold	<code>bool</code>

## Outputs

<code>ySCALE</code>	Output reference to vector <code>SCALE</code>	<code>reference</code>
<code>yV</code>	Reference to matrix of transformed right or left eigenvectors	<code>reference</code>
<code>E</code>	Error indicator	<code>bool</code>
<code>info</code>	LAPACK function result info. If <code>info = -i</code> , the <code>i</code> -th argument had an illegal value	<code>long</code>

**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 [\[7\]](#) for more details.

## Inputs

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
HLD	Hold		bool

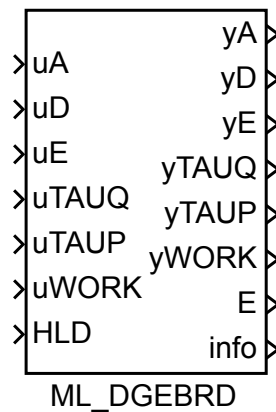
## Outputs

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
ihi	Zero based high row and column index of working submatrix		long
E	Error indicator		bool
info	LAPACK function result info. If info = -i, the i=th argument had an illegal value		long

## 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 [7] for more details.

### Inputs

<code>uA</code>	Input reference to matrix A	reference
<code>uD</code>	Diagonal elements of the bidiagonal matrix B	reference
<code>uE</code>	Off-diagonal elements of the bidiagonal matrix B	reference
<code>uTAUQ</code>	Reference to a vector of scalar factors of the elementary reflectors which represent the orthogonal matrix Q	reference
<code>uTAUP</code>	Reference to a vector of scalar factors of the elementary reflectors which represent the orthogonal matrix P	reference
<code>uWORK</code>	Input reference to working vector WORK	reference
<code>HLD</code>	Hold	bool

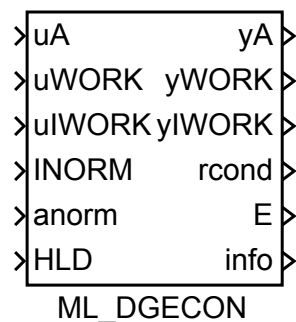
### Outputs

<code>yA</code>	Output reference to matrix A	reference
<code>yD</code>	Output reference to D	reference
<code>yE</code>	Output reference to E	reference
<code>yTAUQ</code>	Output reference to TAUQ	reference
<code>yTAUP</code>	Output reference to TAUP	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

## ML\_DGECN – 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 `DGECN` is called internally:

```
DGECN(sINORM, N, uA, LDA, anorm, rcond, uWORK, uIWORK, info);
```

where parameters of `DGECN` are set in the following way:

- If `INORM = on` then the string `sINORM` is set to "I" else it is set to "1".
- `N` is number of columns of the matrix referenced by `uA`.
- `LDA` is the leading dimension of the matrix referenced by `uA`.
- `rcond` is returned reciprocal value of the condition number of the matrix referenced by `uA`.
- `info` is return code from the function `DGECN`.

The error flag `E` is set to `on` if:

- the reference `uA` or `uWORK` or `uIWORK` is not defined (i.e. input `uA` or `uWORK` or `uIWORK` is not connected),
- the matrix referenced by `uA` is not square,
- number of elements of the vector referenced by `uWORK` is less than  $4 * N$ ,
- number of elements of the integer vector referenced by `uIWORK` is less than  $N$ ,

- the call of the function `DGECOM` returns error using the function `XERBLA`, see the return code `info` and system log.

See LAPACK documentation [7] for more details.

### Inputs

<code>uA</code>	Input reference to matrix A	<code>reference</code>
<code>uWORK</code>	Input reference to working vector WORK	<code>reference</code>
<code>uIWORK</code>	Input reference to integer working vector WORK	<code>reference</code>
<code>INORM</code>	Use Infinity-norm	<code>bool</code>
<code>anorm</code>	Norm of the original matrix A	<code>double</code>
<code>HLD</code>	Hold	<code>bool</code>

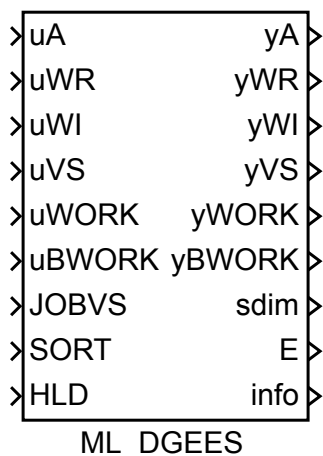
### Outputs

<code>yA</code>	Output reference to matrix A	<code>reference</code>
<code>yWORK</code>	Output reference to working vector WORK	<code>reference</code>
<code>yIWORK</code>	Output reference to integer working vector WORK	<code>reference</code>
<code>rcond</code>	The reciprocal of the condition number of the matrix A	<code>double</code>
<code>E</code>	Error indicator	<code>bool</code>
<code>info</code>	LAPACK function result info. If <code>info = -i</code> , the <code>i</code> -th argument had an illegal value	<code>long</code>

**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 [7] for more details.

## Inputs

<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

## Outputs

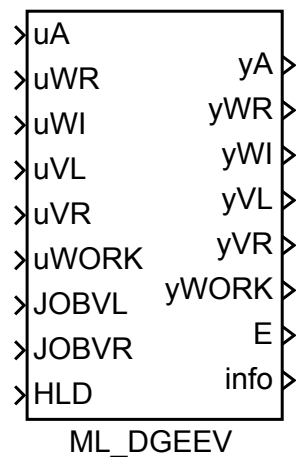
<code>yA</code>	Output reference to matrix <code>A</code>	reference
<code>yWR</code>	Output reference to vector of real parts of eigenvalues	reference
<code>yWI</code>	Output reference to vector of imaginary parts of eigenvalues	reference
<code>yVS</code>	Output reference to <code>VS</code>	reference
<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
<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



## 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 [7] for more details.

## Inputs

<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

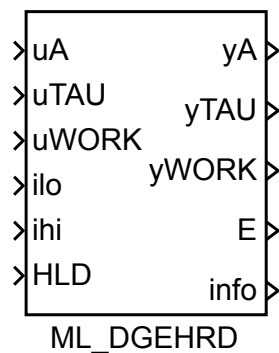
## Outputs

<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

## 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 [7] for more details.

### Inputs

<code>uA</code>	Input reference to matrix A	<b>reference</b>
<code>uTAU</code>	Input reference to vector of scalar factors of the elementary reflectors	<b>reference</b>
<code>uWORK</code>	Input reference to working vector <code>WORK</code>	<b>reference</b>
<code>ilo</code>	Zero based low row and column index of working submatrix	<b>long</b>
<code>ihi</code>	Zero based high row and column index of working submatrix	<b>long</b>
<code>HLD</code>	Hold	<b>bool</b>

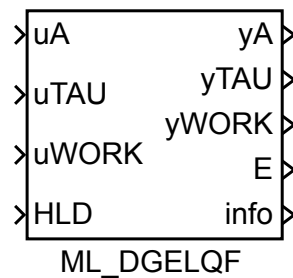
### Outputs

<code>yA</code>	Output reference to matrix A	<b>reference</b>
<code>yTAU</code>	Output reference to vector of scalar factors of the elementary reflectors	<b>reference</b>
<code>yWORK</code>	Output reference to working vector <code>WORK</code>	<b>reference</b>
<code>E</code>	Error indicator	<b>bool</b>
<code>info</code>	LAPACK function result info. If <code>info = -i</code> , the <code>i</code> -th argument had an illegal value	<b>long</b>

## 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 [\[7\]](#) for more details.

## Inputs

<code>uA</code>	Input reference to matrix A	<code>reference</code>
<code>uTAU</code>	Input reference to vector of scalar factors of the elementary reflectors	<code>reference</code>
<code>uWORK</code>	Input reference to working vector WORK	<code>reference</code>
<code>HLD</code>	Hold	<code>bool</code>

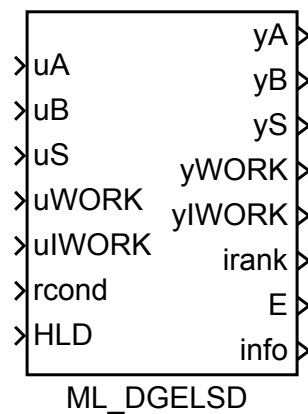
## Outputs

<code>yA</code>	Output reference to matrix A	<code>reference</code>
<code>yTAU</code>	Output reference to vector of scalar factors of the elementary reflectors	<code>reference</code>
<code>yWORK</code>	Output reference to working vector WORK	<code>reference</code>
<code>E</code>	Error indicator	<code>bool</code>
<code>info</code>	LAPACK function result info. If <code>info = -i</code> , the <code>i</code> -th argument had an illegal value	<code>long</code>

## 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 [7] for more details.

### Inputs

<code>uA</code>	Input reference to matrix <code>A</code>	reference
<code>uB</code>	Input reference to matrix <code>B</code>	reference
<code>uS</code>	Input reference to vector of singular values	reference
<code>uWORK</code>	Input reference to working vector <code>WORK</code>	reference
<code>uIWORK</code>	Input reference to integer working vector <code>WORK</code>	reference
<code>rcond</code>	Used to determine the effective rank of <code>A</code>	double
<code>HLD</code>	Hold	bool

### Outputs

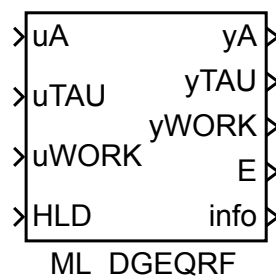
<code>yA</code>	Output reference to matrix <code>A</code>	reference
<code>yB</code>	Output reference to matrix <code>B</code>	reference
<code>yS</code>	Output reference to vector of singular values	reference
<code>yWORK</code>	Output reference to working vector <code>WORK</code>	reference
<code>yIWORK</code>	Output reference to integer working vector <code>WORK</code>	reference
<code>irank</code>	Effective rank of <code>A</code>	long
<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



## 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 [\[7\]](#) for more details.

## Inputs

<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

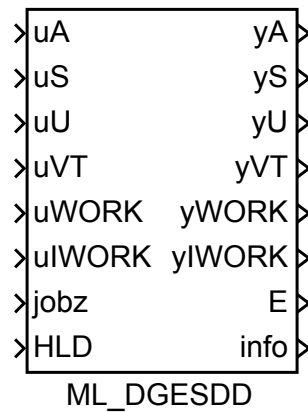
## Outputs

<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

## 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`:  $\{0,1\} \rightarrow \text{"A"}$ ,  $\{2\} \rightarrow \text{"S"}$ ,  $\{3\} \rightarrow \text{"O"}$  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 [7] for more details.

### Inputs

<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
<code>HLD</code>	Hold	bool

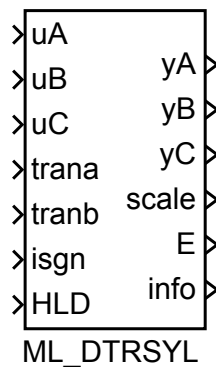
### Outputs

<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

## 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(sSTRANA, sSTRANB, 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 **sSTRANA** and **sSTRANB**:  $\{0, 1\} \rightarrow \text{"N"}, \{2\} \rightarrow \text{"T"}$  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 [7] for more details.

### Inputs

<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
<code>tranb</code>	Transposition of matrix B	↓0 ↑3	long
<code>isgn</code>	Sign in the equation (1 or -1)	↓-1 ↑1	long
<code>HLD</code>	Hold		bool

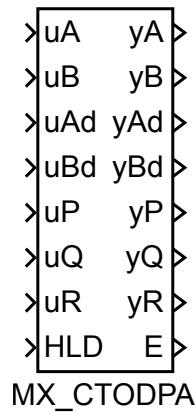
### Outputs

<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
<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

## MX\_CTODPA – Discretizes continuous model given by (A,B) to (Ad,Bd) using Pade approximations

Block Symbol

Licence: [STANDARD](#)



### Function Description

This function block discretizes a continuous state space model using Padé approximations of matrix exponential and its integral and scaling technique ([5]). The used technique is similar to method 3 Scaling and squaring described in [8].

The output references  $y_A$ ,  $y_B$ ,  $y_{Ad}$ ,  $y_{Bd}$ ,  $y_P$ ,  $y_Q$  and  $y_R$  are always set to the corresponding input references  $u_A$ ,  $u_B$ ,  $u_{Ad}$ ,  $u_{Bd}$ ,  $u_P$ ,  $u_Q$  and  $u_R$ . If  $HLD = on$  then nothing is computed otherwise the function `mCtoD` is called internally:

```
mCtoD(nRes, 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 **N**,
- 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.

### Inputs

<b>uA</b>	Input reference to matrix A	reference
<b>uB</b>	Input reference to matrix B	reference
<b>uAd</b>	Input reference to discretized matrix A	reference
<b>uBd</b>	Input reference to discretized matrix B	reference
<b>uP</b>	Input reference to a helper matrix	reference
<b>uQ</b>	Input reference to a helper matrix	reference
<b>uR</b>	Input reference to a helper matrix	reference
<b>HLd</b>	Hold	bool

### Parameters

<b>is</b>	Pade approximation order	$\downarrow 0 \uparrow 4 \odot 2$	long
<b>eps</b>	Approximation accuracy	$\downarrow 0.0 \uparrow 0.001 \odot 0.0$	double

### Outputs

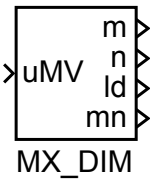
<b>yA</b>	Output reference to matrix A	reference
<b>yB</b>	Output reference to matrix B	reference
<b>yAd</b>	Output reference to discretized matrix A	reference
<b>yBd</b>	Output reference to discretized matrix B	reference
<b>yP</b>	Output reference to a helper matrix	reference
<b>yQ</b>	Output reference to a helper matrix	reference
<b>yR</b>	Output reference to a helper matrix	reference
<b>E</b>	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**.

**Input**

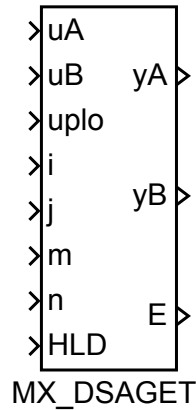
<b>uMV</b>	Input reference to a matrix or vector	<b>reference</b>
------------	---------------------------------------	------------------

**Outputs**

<b>m</b>	Number of matrix rows	<b>long</b>
<b>n</b>	Number of matrix columns	<b>long</b>
<b>ld</b>	Leading dimension ( $\geq$ number of rows)	<b>long</b>
<b>mn</b>	Count of elements ( $m \times n$ )	<b>long</b>

**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  $i \geq MA$  then **I** is set to  $MA - 1$  else **I** is set to **i**, where **MA** is the number of rows of the matrix referenced by **uA**.
- If the input  $j \leq 0$  then **J** is set to 0 else if  $j \geq NA$  then **J** is set to  $NA - 1$  else **J** is set to **j**, where **NA** is the number of columns of the matrix referenced by **uA**.
- Number of copied rows **M** is set in two stages. First, **M** is set to minimum of  $MA - I$  and number of rows of the matrix referenced by **uB**. Second, if  $m > 0$  then **M** is set to the minimum of **m** and **M**.
- Number of copied columns **N** is set in two stages. First, **N** is set to minimum of  $NA - J$  and number of columns of the matrix referenced by **uB**. Second, if  $n > 0$  then **N** is set to the minimum of **n** and **N**.

The error flag **E** is set to on if:

- the reference **uA** or **uB** is not defined (i.e. input **uA** or **uB** is not connected),

- `uplo` is less than 0 or greater than 3,
- the number of elements of the matrix referenced by `uB` is less than  $M * N$ .

## Inputs

<code>uA</code>	Input reference to matrix A	<code>reference</code>
<code>uB</code>	Input reference to matrix B	<code>reference</code>
<code>uplo</code>	Part of the matrix to be copied	<code>long</code>
	0 ..... All                      2 ..... Upper	
	1 ..... All                      3 ..... Lower	
<code>i</code>	Index of the subarray first row	<code>long</code>
<code>j</code>	Index of the subarray first column	<code>long</code>
<code>m</code>	Number of matrix rows	<code>long</code>
<code>n</code>	Number of matrix columns	<code>long</code>
<code>HLD</code>	Hold	<code>bool</code>

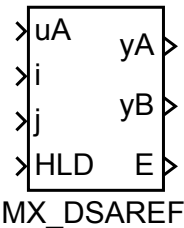
## Outputs

<code>yA</code>	Output reference to matrix A	<code>reference</code>
<code>yB</code>	Output reference to matrix B	<code>reference</code>
<code>E</code>	Error indicator	<code>bool</code>

# 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),
- $0 > i \geq M$ .
- $0 > j \geq N$ .

## Inputs

<b>uA</b>	Input reference to matrix A	<b>reference</b>
<b>i</b>	Index of the subarray first row	<b>long</b>
<b>j</b>	Index of the subarray first column	<b>long</b>
<b>HLD</b>	Hold	<b>bool</b>

## Parameter

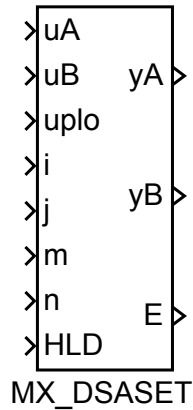
ay	Output reference of the subarray	$\odot[0\ 0]$	double
----	----------------------------------	---------------	--------

## Outputs

yA	Output reference to matrix A	reference
yB	Output reference to matrix B	reference
E	Error indicator	bool

**MX\_DSASET – Set A into subarray of B**

Block Symbol

Licence: [STANDARD](#)**Function Description**

Generally, the function block **MX\_DSASET** copies the matrix referenced by **uA** into the subarray (submatrix) of the matrix referenced by **uB**.

The output references **yA** and **yB** are always set to the corresponding input references **uA** and **uB**. If **HLD = on** then nothing is copied otherwise the matrix referenced by **uA** is copied (with respect to the value of the input **uplo**) to the submatrix of the matrix referenced by **uB** to the row with zero based index **I** and the column with zero based index **J** containing **M** rows and **N** columns. The mentioned variables have the following meanings:

- If the input  $i \leq 0$  then **I** is set to 0 else if  $i \geq \text{MB}$  then **I** is set to  $\text{MB} - 1$  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  $\text{NB} - 1$  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  $\text{MB} - \text{I}$  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  $\text{NB} - \text{J}$  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  $M * N$ .

## Inputs

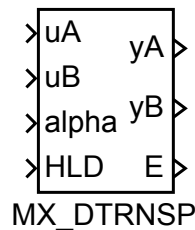
<code>uA</code>	Input reference to matrix A	<code>reference</code>
<code>uB</code>	Input reference to matrix B	<code>reference</code>
<code>uplo</code>	Part of the matrix to be copied	<code>long</code>
	0 ..... All                      2 ..... Upper	
	1 ..... All                      3 ..... Lower	
<code>i</code>	Index of the subarray first row	<code>long</code>
<code>j</code>	Index of the subarray first column	<code>long</code>
<code>m</code>	Number of matrix rows	<code>long</code>
<code>n</code>	Number of matrix columns	<code>long</code>
<code>HLD</code>	Hold	<code>bool</code>

## Outputs

<code>yA</code>	Output reference to matrix A	<code>reference</code>
<code>yB</code>	Output reference to matrix B	<code>reference</code>
<code>E</code>	Error indicator	<code>bool</code>

**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.

**Inputs**

<b>uA</b>	Input reference to matrix A	<b>reference</b>
<b>uB</b>	Input reference to matrix B	<b>reference</b>
<b>alpha</b>	Scalar coefficient alpha	<b>double</b>
<b>HLD</b>	Hold	<b>bool</b>



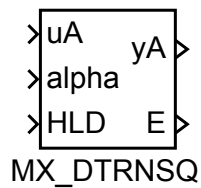
## Outputs

yA	Output reference to matrix A	reference
yB	Output reference to matrix B	reference
E	Error indicator	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.

### Inputs

<b>uA</b>	Input reference to matrix <b>A</b>	<b>reference</b>
<b>alpha</b>	Scalar coefficient <b>alpha</b>	<b>double</b>
<b>HLD</b>	Hold	<b>bool</b>

## Outputs

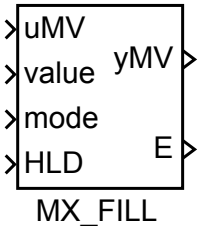
yA      Output reference to matrix A  
E      Error indicator

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),
- $0 > \text{mode} > 4$ .

### Inputs

<b>uMV</b>	Input reference to a matrix or vector	<b>reference</b>
<b>value</b>	Fill value of matrix/vector	<b>double</b>
<b>mode</b>	Fill mode	<b>long</b>
	0,1 ... Value – All elements are set to <b>value</b>	
	2 ..... Ones – All elements are set to 1	
	3 ..... Diagonal value – Diagonal is set to <b>value</b> , the other elements to 0	
	4 ..... Diagonal ones – Initializes identity matrix ( <b>eye</b> )	
<b>HLD</b>	Hold	<b>bool</b>

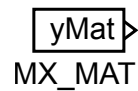
### Outputs

<b>yMV</b>	Output reference to a matrix or vector	<b>reference</b>
<b>E</b>	Error indicator	<b>bool</b>

## 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**.

Note that the present version of the **MATRIX** function block set supports only matrices with the **etype** equal to 8: **Double**.

### Parameters

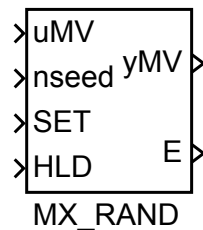
<b>m</b>	Number of matrix rows	↓1 ↑1000000000 ⊕10	long
<b>n</b>	Number of matrix columns	↓1 ↑1000000000 ⊕10	long
<b>ld</b>	Leading dimension ( $\geq$ number of rows)	↓0 ↑1000000000	long
<b>etype</b>	Type of elements	⊕8	long
	1 ..... Bool	6 ..... DWord	
	2 ..... Byte	7 ..... Float	
	3 ..... Short	8 ..... Double	
	4 ..... Long	-- ....	
	5 ..... Word	10 .... Large	

### Output

<b>yMat</b>	Output reference to a matrix	<b>reference</b>
-------------	------------------------------	------------------

**MX\_RAND – Randomly generated matrix or vector**

Block Symbol

Licence: [STANDARD](#)**Function Description**

The function block **MX\_RAND** generates random elements of the matrix or vector referenced by **uMV**.

The output reference **yMV** is always set to the corresponding input references **uMV**. If **HLD = on** then nothing is generated otherwise pseudo-random values of the matrix or vector elements referenced by **uMV** are generated using these rules:

- If the parameter **BIP** is **on** then the generated elements are inside the interval  $[-\text{scale}; \text{scale}]$  else they are inside the interval  $[0; \text{scale}]$ .
- Elements are internally generated using the standard C language function **rand()** which generates pseudo-random numbers in the range from 0 to **RAND\_MAX**. Note, that the value of **RAND\_MAX** can be platform dependent (and it should be at least 32767).
- The rising edge on the input **SET** causes that the standard C language function **srand(nseed)** (initializes the pseudo-random generator with the value of **nseed**) is called before the generation of random elements. The same sequences of pseudo-random numbers are generated after calls of **srand(nseed)** for the same values of **nseed**.

The error flag **E** is set to **on** if the reference **uMV** is not defined (i.e. input **uMV** is not connected).

**Inputs**

<b>uMV</b>	Input reference to a matrix or vector	<b>reference</b>
<b>nseed</b>	Random number seed	<b>long</b>
<b>SET</b>	Set initial value of random number generator to <b>nseed</b> on rising edge	<b>bool</b>
<b>HLD</b>	Hold	<b>bool</b>

## Parameters

<code>BIP</code>	Bipolar random values flag	<code>bool</code>
<code>scale</code>	Random values multiplication factor	$\odot 1.0$ <code>double</code>

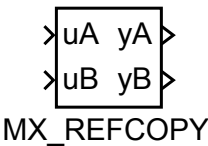
## Outputs

<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**.

**Inputs**

<b>uA</b>	Input reference to matrix A	<b>reference</b>
<b>uB</b>	Input reference to matrix B	<b>reference</b>

**Outputs**

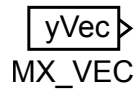
<b>yA</b>	Output reference to matrix A	<b>reference</b>
<b>yB</b>	Output reference to matrix B	<b>reference</b>



## 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**.

Note that the present version of the **MATRIX** function block set supports only vectors with the **etype** equal to 8: **Double**.

### Parameters

<b>n</b>	Number of vector elements	↓1 ↑10000000000 ⊖10	long
<b>etype</b>	Type of elements	⊖8	long
1	..... Bool	6	..... DWord
2	..... Byte	7	..... Float
3	..... Short	8	..... Double
4	..... Long	--	.....
5	..... Word	10	..... Large

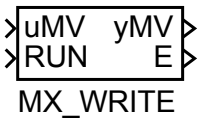
### Output

<b>yVec</b>	Output reference to a vector	<b>reference</b>
-------------	------------------------------	------------------

**MX\_WRITE – Write a Matrix/Vector to the console/system log**

Block Symbol

Licence: [STANDARD](#)



**Function Description**

This function block can write a vector or matrix to the console or the system log. The severity of the console/system log output is set by the parameter `mode` in combination with settings of system log from REXYGEN Studio, menu **Target/Configure System Log**. Written data can be viewed in REXYGEN Studio, after opening the system log window by the command **Target/Show System Log**. The function block is very useful for debugging purposes of matrix/vector algorithms.

The output references `yMV` is always set to the input reference `uMV`. If `RUN = off` then nothing else is done otherwise matrix or vector is written to the system log if the configured target logging level for function blocks contains the configured `mode`. Format of each matrix/vector element is determined by parameters `mchars` and `mdec`.

The error flag `E` is set to `on` if:

- the reference `uMV` is not defined (i.e. input `uMV` is not connected),
- $3 > \text{mchars} > 25$ ,
- $0 > \text{mdec} > \text{mchars} - 2$ .

**Inputs**

<code>uMV</code>	Input reference to a matrix or vector	reference
<code>RUN</code>	Enable execution	bool

**Parameters**

<code>Symbol</code>	Matrix/vector symbolic name for console or log output	⊙A	string
<code>mchars</code>	Number of characters per single element	↓3 ↑25 ⊙8	long
<code>mdec</code>	Number of decimal digits per single element	↓0 ↑23 ⊙4	long

<b>mode</b>	Severity mode of writing	⊙3	long
1	..... None		
2	..... Verbose		
3	..... Information		
4	..... Warning		
5	..... Error		

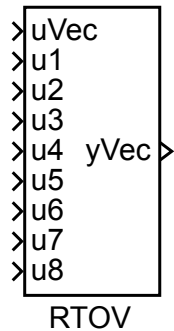
## Outputs

<b>yMV</b>	Output reference to a matrix or vector	<b>reference</b>
<b>E</b>	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.

Inputs

<b>uVec</b>	Vector signal	<b>reference</b>
<b>u1</b>	Analog input of the block	<b>double</b>
<b>u2</b>	Analog input of the block	<b>double</b>
<b>u3</b>	Analog input of the block	<b>double</b>
<b>u4</b>	Analog input of the block	<b>double</b>
<b>u5</b>	Analog input of the block	<b>double</b>
<b>u6</b>	Analog input of the block	<b>double</b>
<b>u7</b>	Analog input of the block	<b>double</b>
<b>u8</b>	Analog input of the block	<b>double</b>

Parameters

<b>nmax</b>	Allocated size of vector	↓0 ↻8	<b>long</b>
<b>offset</b>	Index of the first input in vector	↓0	<b>long</b>
<b>n</b>	Number of valid inputs	↓1 ↑8 ↻8	<b>long</b>

## Output

yVec

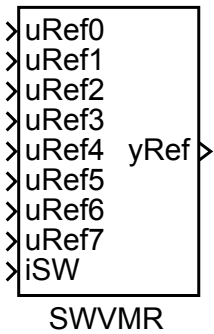
Vector signal

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.

Inputs

uRef0	Vector signal	reference
uRef1	Vector signal	reference
uRef2	Vector signal	reference
uRef3	Vector signal	reference
uRef4	Vector signal	reference
uRef5	Vector signal	reference
uRef6	Vector signal	reference
uRef7	Vector signal	reference
iSW	Active signal selector	long

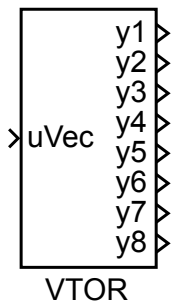
Output

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.

### Input

uVec	Vector signal	reference
------	---------------	-----------

### Parameters

n	Number of valid outputs	↓1 ↑8 ⊖8 long
offset	Index of the first output	↓0 long

### Outputs

y1	Analog output of the block	double
y2	Analog output of the block	double
y3	Analog output of the block	double
y4	Analog output of the block	double
y5	Analog output of the block	double
y6	Analog output of the block	double
y7	Analog output of the block	double
y8	Analog output of the block	double





## Chapter 15

# SPEC – Special blocks

### Contents

---

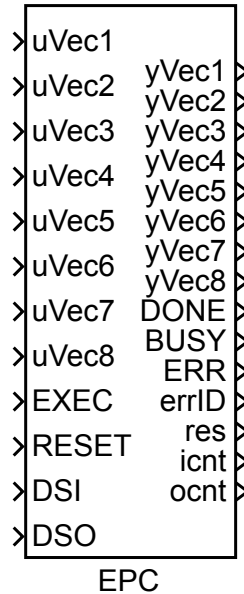
EPC – External program call . . . . .	446
HTTP – HTTP GET or POST request (obsolete) . . . . .	449
HTTP2 – Block for generating HTTP GET or POST requests . .	451
SMTP – Send email message via SMTP . . . . .	453
RDC – Remote data connection . . . . .	455
REXLANG – User programmable block . . . . .	460

---

## EPC – External program call

Block Symbol

Licence: [ADVANCED](#)



### Function Description

The **EPC** block executes an external program upon a rising edge (**off**→**on**) occurring at the **EXEC** input. The name and options of the program are defined by the **cmd** parameter. The format is the same as if the program was executed from the command line of the operating system.

It is possible to pass data from the **REXYGEN** system to the external program via files. The formatting of the files is defined by the **format** parameter. All the currently supported formats are textual and simple, which allows straightforward processing of the data in arbitrary program. Use e.g.

```
values=load('-ASCII', 'epc_inputVec1');
```

for loading the data in MATLAB or

```
values=read('epc_inputVec1',-1,32);
```

in SCILAB. The filename and number of columns must be adjusted for the given project. Data exchange in the opposite direction is naturally also supported, the **REXYGEN** system can read the files in the same format.

The block works in two modes. In *basic mode*, the rising edge on the **EXEC** input triggers reading the data on inputs and storing them in the **ifns** file. The values of the *i*-th input vector **uVec**<*i*> are stored in the *i*-th file from the **ifns** list. In *sampling mode*, the data from the input vectors are stored in each period of the control algorithm. In

both cases the values from one time instant form one line in the file.

Analogically, the data from output files are copied to the outputs of the block (one line from the *i*-th file in the **ofns** list to the *i*-th output vector **yVec<i>**).

The inputs working in the *sampling mode* are defined by the **s1** list (comma-separated numbers). The outputs work always in the *sampling mode* – the last values are kept when the end of file is reached. The copying of data to input files can be blocked by the **DSI** input, the same holds for output data and the **DSO** input.

Use the **RTOV** block to combine individual signals into a vector one for the **uVec** input. The **RTOV** blocks can be chained, therefore it is possible to create a vector of arbitrary dimension. Similarly, use the **VTOR** block to demultiplex a vector signal to individual signals.

## Inputs

<b>uVec1..uVec8</b>	Input vector signal	reference
<b>EXEC</b>	External program is called on rising edge	bool
<b>RESET</b>	Block reset (deletes the input and output files and terminates the external program)	bool
<b>DSI</b>	Disable inputs sampling	bool
<b>DSO</b>	Disable outputs sampling	bool

## Outputs

<b>yVec1..yVec8</b>	Output vector signal	reference
<b>DONE</b>	External program finished	bool
<b>BUSY</b>	External program running	bool
<b>ERR</b>	Error flag	bool
<b>errID</b>	Error code	error
	<b>i</b> ..... REXYGEN general error	
<b>res</b>	External program return code	long
<b>icnt</b>	Current input sample	long
<b>ocnt</b>	Current output sample	long

## Parameters

<b>cmd</b>	Operating system command to execute	string
<b>ifns</b>	Input filenames (separated by semicolon)	⊙epc_uVec1;epc_uVec2 string
<b>ofns</b>	Output filenames (separated by semicolon)	⊙epc_yVec1;epc_yVec2 string
<b>s1</b>	List of inputs working in the <i>sampling mode</i> . The format of the list is e.g. 1,3..5,8. Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.	long ↓0 ↑255 ⊙85
<b>ifm</b>	Maximum number of input samples	⊙10000 long

<b>format</b>	Format of input and output files	⊙1	long
	1 ..... Space-delimited values		
	2 ..... CSV (decimal point and commas)		
	3 ..... CSV (decimal comma and semicolons)		
<b>nmax</b>	Maximum output vectors length	↓2 ↑1000000 ⊙100	long

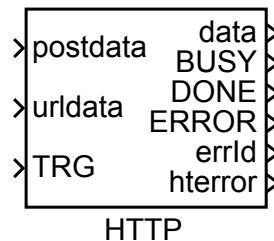
## Notes

- 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 used for execution of some operating system functions.

## HTTP – HTTP GET or POST request (obsolete)

Block Symbol

Licence: [ADVANCED](#)



### Function Description

The **HTTP** block performs a single HTTP GET or POST request. Target address (URL) is defined by **url** parameter and **urldata** input. A final URL is formed in the way so that **urldata** input is simply added to **url** parameter.

HTTP request is started by the **TRG** parameter. Then the **BUSY** output is set until a request is finished, which is signaled by the **DONE** output. In case of an error, the **ERROR** output is set. The **errId** output carries last error identified by REXYGEN system error code. The **hterror** carries a HTTP status code. All data sent back by server to client is stored in the **data** output.

The block may be run in blocking or non-blocking mode which is specified by the **BLOCKING** parameter. In blocking mode, execution of a task is suspended until a request is finished. In non-blocking mode, the block performs only single operation depending on available data and execution of a task is not blocked. It is advised to always run **HTTP** block in non-blocking mode. It is however necessary to mention that on various operating systems some operations can not be performed in the non-blocking mode, so be careful and do not use this block in quick tasks or in tasks with short execution period. The non-blocking operation is best supported on GNU/Linux operating system. The maximal duration of a request performed by the **HTTP** block is specified by the **timeout** parameter.

The block supports user authentication using basic HTTP authentication method. User name and password may be specified by **user** and **password** parameters. The block also supports secure HTTP (HTTPS). It is also possible to let the block verify server's certificate by setting the **VERIFY** parameter. SSL certificate of a server or server's trusted certificate authority must be stored in the **certificate** parameter in a PEM format. The block does not support any certificate storage.

Parameters **postmime** and **acceptmime** specify MIME encoding of data being sent to server or expected encoding of a HTTP response.

Parameters **nmax**, **postmax**, and **datamax** specify maximum sizes of buffers allocated by the block. The **nmax** parameter is maximal size of any string parameter. The **postmax** parameter specifies a maximal size of **postdata**. The **datamax** parameter specifies a

maximal size of `data`.

## Inputs

<code>postdata</code>	Data to put in HTTP POST request	<code>string</code>
<code>urldata</code>	Data to append to URL address	<code>string</code>
<code>TRG</code>	Trigger of the selected action	<code>bool</code>

## Parameters

<code>url</code>	URL address to send the HTTP request to		<code>string</code>
<code>method</code>	HTTP request type	⊙1	<code>long</code>
	1 ..... GET		
	2 ..... POST		
<code>user</code>	User name		<code>string</code>
<code>password</code>	Password		<code>string</code>
<code>certificate</code>	Authentication certificate		<code>string</code>
<code>VERIFY</code>	Enable server verification (valid certificate)		<code>bool</code>
<code>postmime</code>	MIME encoding for POST request	⊙application/json	<code>string</code>
<code>acceptmime</code>	MIME encoding of HTTP response	⊙application/json	<code>string</code>
<code>timeout</code>	Timeout interval	⊙5.0	<code>double</code>
<code>BLOCKING</code>	Wait for the operation to finish		<code>bool</code>
<code>nmax</code>	Allocated size of string	↓0 ↑65520	<code>long</code>
<code>postmax</code>	Allocated memory for POST request data	↓128 ↑65520 ⊙256	<code>long</code>
<code>datamax</code>	Allocated memory for HTTP response	↓128 ↑10000000 ⊙1024	<code>long</code>

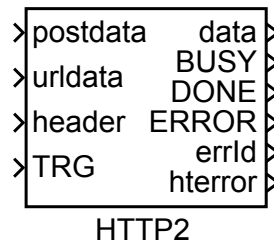
## Outputs

<code>data</code>	Response data	<code>string</code>
<code>BUSY</code>	Sending HTTP request	<code>bool</code>
<code>DONE</code>	HTTP request processed	<code>bool</code>
<code>ERROR</code>	Error indicator	<code>bool</code>
<code>errId</code>	Error code	<code>error</code>
<code>hterror</code>	HTTP response	<code>long</code>

## HTTP2 – Block for generating HTTP GET or POST requests

Block Symbol

Licence: [ADVANCED](#)



### Function Description

The **HTTP** block performs a single HTTP GET or POST request. Target address (URL) is defined by **url** parameter and **urldata** input. A final URL is formed in the way so that **urldata** input is simply added to **url** parameter. The **header** input can be used for declaration of additional header fields.

HTTP request is started by the **TRG** parameter. Then the **BUSY** output is set until a request is finished, which is signaled by the **DONE** output. In case of an error, the **ERROR** output is set. The **errId** output carries last error identified by REXYGEN system error code. The **hterror** carries a HTTP status code. All data sent back by server to client is stored in the **data** output.

The block may be run in blocking or non-blocking mode which is specified by the **BLOCKING** parameter. In blocking mode, execution of a task is suspended until a request is finished. In non-blocking mode, the block performs only single operation depending on available data and execution of a task is not blocked. It is advised to always run **HTTP** block in non-blocking mode. It is however necessary to mention that on various operating systems some operations can not be performed in the non-blocking mode, so be careful and do not use this block in quick tasks or in tasks with short execution period. The non-blocking operation is best supported on GNU/Linux operating system. The maximal duration of a request performed by the **HTTP** block is specified by the **timeout** parameter.

The block supports user authentication using basic HTTP authentication method. User name and password may be specified by **user** and **password** parameters. The block also supports secure HTTP (HTTPS). It is also possible to let the block verify server's certificate by setting the **VERIFY** parameter. SSL certificate of a server or server's trusted certificate authority must be stored in the **certificate** parameter in a PEM format. The block does not support any certificate storage.

Parameters **postmime** and **acceptmime** specify MIME encoding of data being sent to server or expected encoding of a HTTP response.

Parameters **nmax**, **postmax**, and **datamax** specify maximum sizes of buffers allocated by the block. The **nmax** parameter is maximal size of any string parameter. The **postmax**

parameter specifies a maximal size of `postdata`. The `datamax` parameter specifies a maximal size of `data`.

### Inputs

<code>postdata</code>	Data to put in HTTP POST request	<code>string</code>
<code>urldata</code>	Data to append to URL address	<code>string</code>
<code>header</code>	Additional header fields	<code>string</code>
<code>TRG</code>	Trigger of the selected action	<code>bool</code>

### Parameters

<code>url</code>	URL address to send the HTTP request to	<code>string</code>
<code>method</code>	HTTP request type	⊙1 <code>long</code>
	1 ..... GET	
	2 ..... POST	
<code>user</code>	User name	<code>string</code>
<code>password</code>	Password	<code>string</code>
<code>certificate</code>	Authentication certificate	<code>string</code>
<code>VERIFY</code>	Enable server verification (valid certificate)	<code>bool</code>
<code>postmime</code>	MIME encoding for POST request	⊙application/json <code>string</code>
<code>acceptmime</code>	MIME encoding for GET request	⊙application/json <code>string</code>
<code>timeout</code>	Timeout interval	⊙5.0 <code>double</code>
<code>BLOCKING</code>	Wait for the operation to finish	<code>bool</code>
<code>nmax</code>	Allocated size of string	↓0 ↑65520 <code>long</code>
<code>postmax</code>	Allocated memory for POST request data	↓128 ↑65520 ⊙256 <code>long</code>
<code>datamax</code>	Allocated memory for HTTP response	↓128 ↑10000000 ⊙1024 <code>long</code>

### Outputs

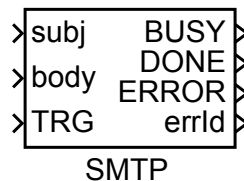
<code>data</code>	Response data	<code>string</code>
<code>BUSY</code>	Sending HTTP request	<code>bool</code>
<code>DONE</code>	HTTP request processed	<code>bool</code>
<code>ERROR</code>	Error indicator	<code>bool</code>
<code>errId</code>	Error code	<code>error</code>
<code>hterror</code>	HTTP response	<code>long</code>



## SMTP – Send email message via SMTP

Block Symbol

Licence: [ADVANCED](#)



### Function Description

The **SMTP** block sends a single email message via standard SMTP protocol. The block acts as a simple email client. It does not implement a mail server.

Contents of a message is defined by the inputs **subj** and **body**. Parameters **from** and **to** specify sender and receiver of a message. A message is sent when the **TRG** parameter is set. Then the **BUSY** output is set until the request is finished, which is signaled by the **DONE** output. In case of an error, the **ERROR** output is set. The **errId** output carries last error identified by REXYGEN system error code. The **domain** parameter must always be set to identify the target device. A default value should work in most cases.

The block may be run in blocking or non-blocking mode which is specified by the **BLOCKING** parameter. In a blocking mode, the execution of a task is suspended until a request is finished. In a non-blocking mode, the block performs only a single operation depending on available data and the execution of a task is not blocked. It is advised to always run the **SMTP** block in a non-blocking mode. It is however necessary to mention that on various operating systems some operations may not be performed in a non-blocking mode, so be careful and do not use this block in quick tasks or in tasks with short execution period. The non-blocking operation is best supported on GNU/Linux operating system. A maximal duration of a request performed by the **SMTP** block is specified by the **timeout** parameter.

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. An 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.

Parameters **nmax** and **datamax** specify maximum sizes of buffers allocated by the block. The **nmax** parameter is maximal size of any string parameter. The **datamax** parameter specifies a maximal size of a **data**.

## Inputs

subj	Subject of the e-mail message	string
body	Body of the e-mail message	string
TRG	Trigger of the selected action	bool

## Parameters

server	SMTP server address	string
to	E-mail of the recipient	string
from	E-mail of the sender	string
tls	Encryption method	⊙1 long
	1 ..... None	
	2 ..... StartTLS	
	3 ..... TLS	
user	User name	string
password	Password	string
domain	Domain name or identification of the target device	string
auth	Authentication method	⊙1 long
	1 ..... Login	
	2 ..... Plain	
certificate	Authentication certificate	string
VERIFY	Enable server verification (valid certificate)	bool
timeout	Timeout interval	double
BLOCKING	Wait for the operation to finish	bool
nmax	Allocated size of string	↓0 ↑65520 ⊙512 long
datamax	Allocated memory for HTTP response	↓128 ↑65520 long

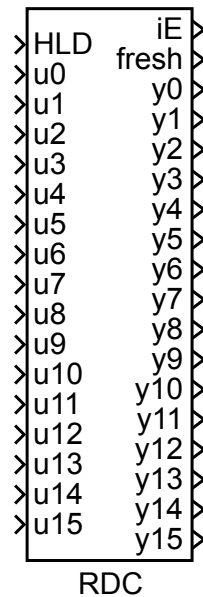
## Outputs

BUSY	Sending e-mail	bool
DONE	E-mail has been sent	bool
ERROR	Error indicator	bool
errId	Error code	error

## RDC – Remote data connection

Block Symbol

Licence: [ADVANCED](#)



### Function Description

The **RDC** block is a special input-output block. The values are transferred between two blocks on different computers, eventually two different Simulinks on the same computer or Simulink and the REXYGEN system on the same computer. In order to communicate, the two **RDC** blocks must have the same **id** number. The communication is based on UDP/IP protocol. This protocol is used as commonly as the more known TCP/IP, i.e. it works over all LAN networks and the Internet. The algorithm performs the following operations in each step:

- If **HLD** = **on**, the block execution is terminated.
- If the **period** parameter is a positive number, the difference between the system timer and the time of the last packet sending is evaluated. The block execution is stopped if the difference does not exceed the **period** parameter. If the **period** parameter is zero or negative, the time difference is not checked.
- A data packet is created. The packet includes block number, the so-called **invoke** number (serial number of the packet) and the values **u0** to **u15**. All values are stored in the commonly used so-called network byte order, therefore the application is computer and/or processor independent.

- The packet is sent to the specified IP address and port.
- The `invoke` number is increased by 1.
- It is checked whether any incoming packets have been received.
- If so, the packet validity is checked (size, `id` number, `invoke` number).
- If the data is valid, all outputs `y0` to `y15` are set to the values contained in the packet received.
- The `fresh` output is updated. In case of error, the error code is displayed by the `err` output.

There are 16 values transmitted in each direction periodically between two blocks with the same `id` number. The `u(i)` input of the first block is transmitted the `y(i)` output of the other block. Unlike the TCP/IP protocol, the UDP/IP protocol does not have any mechanism for dealing with lost or duplicate packets, so it must be handled by the algorithm itself. The `invoke` number is used for this purpose. This state variable is increased by 1 each time a packet is sent. The block stores also the `invoke` number of the last received packet. It is possible to distinguish between various events by comparing these two invoke numbers. The packets with invoke numbers lower than the invoke number of the last received packet are denied unless the difference is greater than 10. This solves the situation when one of the RDC blocks is restarted and its `invoke` number is reset.

All RDC blocks in the same application must have the same `local port` number and the number of RDC blocks is limited to 64 for implementation reasons. If there are two applications using the RDC block running on the same machine, then each of them must use a different `local port` number.

### Inputs

<code>HLD</code>	Input for disabling the execution of the block. No packets are received nor transmitted when <code>HLD = on</code> .	<code>bool</code>
<code>u0..u15</code>	Values which are sent/written to the output values <code>y0</code> to <code>y15</code> of the paired block	<code>double</code>

### Outputs

<code>iE</code>	Displays the code of the last error. The error codes are listed below: 0 ..... No error	<code>long</code>
-----------------	--	-------------------

*Persistent errors originating in the initialization phase ( $< 0$ ). Cannot be fixed automatically.*

- 1 .... Maximum number of blocks exceeded ( $> 64$ )
- 2 .... Local ports mismatch; the `lport` parameter must be the same for all RDC blocks within one application
- 3 .... Error opening socket (the UDP/IP protocol is not available)
- 4 .... Error assigning local port (port already occupied by another service or application)
- 5 .... Error setting the so-called non-blocking socket mode (the RDC block requires this mode)
- 10 ... Error initializing the socket library
- 11 ... Error initializing the socket library
- 12 ... Error initializing the socket library

*Temporary errors originating in any cycle of the code ( $> 0$ ). Can be fixed automatically.*

- 1 ..... Initialization successful, yet no data packet has been received
- 2 ..... Packet consistency error (incorrect length – transmission error or conflicting service/application is running)
- 4 ..... Error receiving packet (socket library error)
- 8 ..... Error sending packet (socket library error)

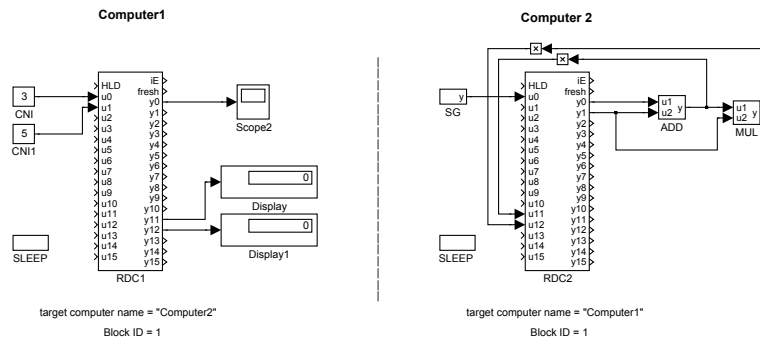
<code>fresh</code>	Elapsed time (in seconds) since the last received packet. Can be used for detection of an error in the paired block.	double
<code>y0..y15</code>	Values transmitted from the input ports <code>u0</code> to <code>u15</code> of the paired RDC block – data from the last packet received	double

## Parameters

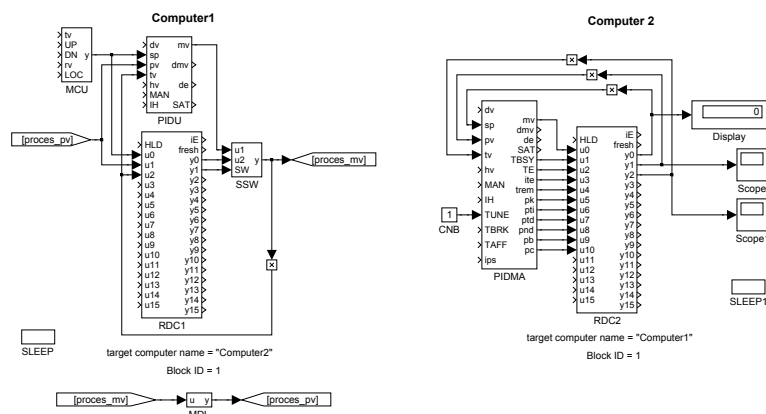
<code>target</code>	Name or IP address running the paired RDC block. Broadcast address is allowed.	string
<code>rport</code>	Remote port – address of the UDP/IP protocol service, it is recommended to keep the default value unless necessary (service/application conflict) $\odot 1288$	word
<code>lport</code>	Local port – similar meaning as the <code>rport</code> parameter; remote port applies to the receiving machine, local port applies to the machine sending the packet $\odot 1288$	word
<code>id</code>	Block ID – this number is contained within the data packet in order to reach the proper target block (all blocks on the target receive the packet but only the one with the corresponding <code>id</code> decomposes it and uses the data contained to update its outputs) $\downarrow 1 \uparrow 32767 \odot 1$	long
<code>period</code>	The shortest time interval between transmitting/receiving packets (in seconds). The packets are transmitted/received during each execution of the block for <code>period</code> $\leq 0$ while the positive values of this parameter are extremely useful when sending data out of the Simulink continuous models based on a <code>Variable step</code> solver.	double

## Example

The following example explains the function of the RDC block. The constants 3 and 5 are sent from **Computer1** to **Computer2**, where they appear at the 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 outputs 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. Note that **Display** and **Scope** are Matlab/Simulink blocks – to view the data in the REXYGEN system, the REXYGEN Diagnostics diagnostic program and the **TRND** block must be used.



The simplicity of the example is intentional. The goal is to demonstrate the functionality of the block, not the complexity of the system. In reality, the RDC block is used in more complex tasks, e.g. for remote tuning of the PID controller as shown below. The PID control algorithm is running on **Computer1** while the tuning algorithm is executed by **Computer2**. See the **PIDU**, **PIDMA** and **SSW** blocks for more details.



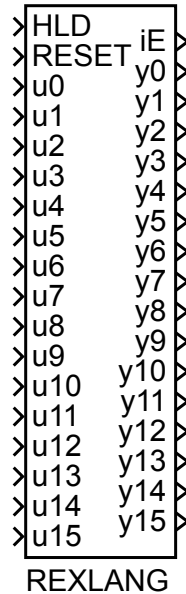
OPC server of the RDC block

There is also an OPC server embedded in the RDC block. Detailed description will be available soon.

## REXLANG – User programmable block

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. It implements an user-defined algorithm written in a scripting language very similar to the C language (or Java).

### Scripting language

As mentioned, the scripting language is similar to the C language. Nevertheless, there are some differences and limitations:

- Only the **double** and **long** data types are supported (it is possible to use **int**, **short**, **bool** as well, but these are internally converted to **long**. The **float** type can be used but it is converted internally to **double**. The **typedef** type is not defined.
- Pointers and structures are not implemented. However, it is possible to define arrays and use the indexes (the **[ ]** operator).
- The **' , '** operator is not implemented.



- The preprocessor supports the following commands: `#include`, `#define`, `#ifdef .. [#else .. ] #endif`, `#ifndef .. [#else .. ] #endif` (i.e. `#pragma` and `#if .. [#else .. ] #endif` are not supported).
- The standard ANSI C libraries are not implemented, however the majority of mathematic functions from `math.h` and some other functions are implemented (see the text below).
- The `input`, `output` and `parameter` keywords are defined for referencing the REXLANG block inputs, outputs and parameters. System functions for controlling the execution and diagnostics are implemented (see the text below).
- The `main()` function is executed periodically during runtime. Alongside the `main()` function the `init()` (executed once at startup), `exit()` (executed once when the control algorithm is stopped) and the `parchange()` (executed on parameters change in the REXYGEN system, executed in each step in Simulink).
- The functions and procedures without parameters must be explicitly declared `void`.
- The identifiers cannot be overloaded, i.e. the keywords and built-in functions cannot share the name with an identifier. The local and global variables cannot share the same name.
- Array initializers are not supported. Neither in local arrays nor the global ones.

## Scripting language syntax

The scripting language syntax is based on the C language, but pointers are not supported and the data types are limited to `long` and `double`. Moreover the `input`, `output` and `parameter` keywords are defined for referencing the REXLANG block inputs, outputs and parameters. The syntax is as follows:

- `<type> input(<input number>) <variable name>;`
- `<type> output(<outpt number>) <variable name>;`
- `<type> parameter(<parameter number>) <variable name>;`

The `input` and `parameter` variables are read-only while the `output` variables are write-only. For example:

```
double input(1) input_signal; /* declaration of a variable of type
                               double, which corresponds with the
                               u1 input of the block */
long output(2) output_signal; /* declaration of a variable of type
                               long, which corresponds with the y2
                               output of the block */
```

```
input_signal = 3;           //not allowed, inputs are read-only
sum = output_signal + 1;    //not allowed, outputs are write-only
if (input_signal>1) output_signal = 3 + input_signal; //correct
```

## Available functions

The following functions are available in the scripting language:

- **Mathematic functions** (see ANSI C, `math.h`):  
`atan`, `sin`, `cos`, `exp`, `log`, `sqrt`, `tan`, `asin`, `acos`, `fabs`, `fmod`, `sinh`, `cosh`, `tanh`,  
`pow`, `atan2`, `ceil`, `floor` and `abs` Please note that the `abs` function works with  
integer numbers. All the other functions work with variables of type `double`.
- **Vector functions** (not part of ANSI C)  

```
double max([n,]val1,...,valn)
    Returns the maximum value. The first parameter defining the number of
    items is optional.
double max(n,vec)
    Returns the value of maximal item in the vec vector.
double min([n,]val1,...,valn)
    Returns the minimum value. The first parameter defining the number of
    items is optional.
double min(n,vec)
    Returns the value of minimal item in the vec vector.
double poly([n,]x,an,...,a1,a0)
    Evaluates the polynomial  $y = a_n x^n + \dots + a_1 x + a_0$ . The first parameter
    defining the number of items is optional.
double poly(n,x,vec)
    Evaluates the polynomial  $y = \text{vec}[n] * x^n + \dots + \text{vec}[1] * x + \text{vec}[0]$ .
double scal(n,vec1,vec2)
    Evaluates the scalar product  $y = \text{vec1}[0] * \text{vec2}[0] + \dots + \text{vec1}[n-1] * \text{vec2}[n-1]$ .
double scal(n,vec1,vec2,skip1,skip2)
    Evaluates the scalar product  $y = \text{vec1}[0] * \text{vec2}[0] + \text{vec1}[\text{skip1}] * \text{vec2}[\text{skip2}] + \dots + \text{vec1}[(n-1)*\text{skip1}] * \text{vec2}[(n-1)*\text{skip2}]$ . 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)
    Evaluates the convolatory product  $y = \text{vec1}[0] * \text{vec2}[n-1] + \text{vec1}[1] * \text{vec2}[n-1] + \dots + \text{vec1}[n-1] * \text{vec2}[0]$ .
double sum(n,vec)
    Sums the items in a vector, i.e.  $y = \text{vec}[0] + \text{vec}[1] + \dots + \text{vec}[n-1]$ .
double sum([n,]val1,...,valn)
    Sums the items, i.e.  $y = \text{val1} + \text{val2} + \dots + \text{valn}$ . The first parameter
    defining the number of items is optional.
```

`[] array([n],[an-1,...,a1,a0])`

Returns an array/vector with the given values. The first parameter defining the number of items is optional. The type of the returned value is chosen automatically to fit the type of parameters (all must be of the same type).

`[] subarray(idx,vec)`

Returns a subarray/subvector of the `vec` array, starting at the `idx` index. The type of the returned value is chosen automatically according to the `vec` array.

`copyarray(count,vecSource,idxSource,vecTarget,idxTarget)`

Copies `count` items of the `vecSource` array, starting at `idxSource` index, to the `vecTarget` array, starting at `idxTarget` index. Both arrays must be of the same type.

`void fillarray(vector, value, count)`

Copies `value` to `count` items of the `vector` array (always starting from index 0).

- **String functions** (ANSI C contains analogous functions in the `string.h` file)

`string strstr(str,idx,len)`

Returns a substring of length `len` starting at index `idx`.

`long strlen(str)`

Returns string length (number of characters).

`long strfind(str,substr)`

Returns the position of first occurrence of `substr` in `str`.

`long strrfind(str,substr)`

Returns the position of last occurrence of `substr` in `str`.

`strreplace(str,pattern,substr)`

Find all occurrences of `pattern` in `str` and replace it with `substr` (in-place replacement, so new string is stored into `str`).

`string strupr(str)`

Converts a string to uppercase.

`long str2long(str)`

Converts string to integer number. The first non-numerical character is considered the end of the input string and the remaining characters are ignored.

`double str2double(str)`

Converts string to a decimal number. The first non-numerical character is considered the end of the input string and the remaining characters are ignored.

`string long2str(num)`

Converts an integer number `num` to text.

`string double2str(num)`

Converts a decimal number `num` to text.

**strcpy(dest,src)**

Function copies the **src** string to the **dest** string. Implemented for compatibility with ANSI C. The construction **dest=src** yields the same result.

**strcat(dest,src)**

Function appends a copy of the **src** string to the **dest** string. Implemented for compatibility with ANSI C. The construction **dest=dest+src** yields the same result.

**strcmp(str1,str2)**

Function compares strings **str1** and **str2**. Implemented for compatibility with ANSI C. The construction **str1==str2** yields the same result.

**long RegExp(str,regexp,capture[])**

Compares the **str** string with regular expression **regexp**. When the string matches the pattern, the **capture** array contains individual sections of the regular expression. **capture[0]** is always the complete regular expression. The function return the number of captured strings or a negative value in case of an error. The regular expression may contain the following:

(?i) ... Must be at the beginning of the regular expression. Makes the matching case-insensitive.

^ ... Match beginning of a string

\$ ... Match end of a string

() ... Grouping and substring capturing

\s ... Match whitespace

\S ... Match non-whitespace

\d ... Match decimal digit

\n ... Match new line character

\r ... Match line feed character

\f ... Match vertical tab character

\v ... Match horizontal tab character

\t ... Match horizontal tab character

\b ... Match backspace character

+ ... Match one or more times (greedy)

+? ... Match one or more times (non-greedy)

\* ... Match zero or more times (greedy)

\*? ... Match zero or more times (non-greedy)

? ... Match zero or once (non-greedy)

x|y ... Match x or y (alternation operator)

\meta ... Match one of the meta characters: ^\$() . [] \*+?|\

\xHH ... Match byte with hex value 0xHH, e.g. \x4a.

[...] ... Match any character from the set. Ranges like [a-z] are supported.

[^...] ... Match any character but the ones from the set.

`long ParseJson(json,cnt,names[],values[])`

The `json` string is supposed to contain text in JSON format. The `names` array contain the requested objects (subitems are accessed via `.`, index of the array via `[]`). The `values` array then contains values of the requested objects. The `cnt` parameter defines the number of requested objects (length of both the `names` and `values` arrays). The function returns the number of values, negative numbers indicate errors.

Note: String variable is declared just like in ANSI C, i.e. `char <variable name>[<maximum number of characters>];`. For passing the strings to functions use `char <variable name>[]` or `string <variable name>`.

- **System functions** (not part of ANSI C)

`Trace(id,val)`

Displays the `id` value and the `val` value. The function is intended for debugging. The `id` is a user-defined constant (from 0 to 9999) for easy identification of the displayed message. The `val` can be of any data type including text string. The output can be found in the system log of REXYGEN. In Simulink the output is displayed directly in the command window of Matlab.

In order to view these debugging messages in the REXYGEN Diagnostics program 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 have to be also enabled for the particular block by checking item "Logging" in the "Runtime" section in 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** tab.

`TraceError(id,val)` `TraceWarning(id,val)` `TraceVerbose(id,val)`

On the contrary to the **Trace** command, the output is routed to the corresponding logging group. To view the messages, enable the corresponding group. See the **Trace** command for details. Messages with the "Error" level are written to the log allways, regardless the "Logging" item is checked for the block.

`Suspend(sec)`

The script is suspended if its execution within the given sampling period takes more seconds than specified by the `sec` parameter. At the next start of the block the script continues from the point where it was suspended. Use `Suspend(0)` to suspend the code immediately.

`double GetPeriod()`

Returns the sampling period of the block in seconds.

`double CurrentTime()`

Returns the current time (in internal format). Intended for use with the `ElapsedTime()` function.

**double ElapsedTime(new\_time, old\_time)**  
 Returns the elapsed time in seconds (decimal number), i.e. the difference between the two time values **new\_time** and **old\_time**. The **CurrentTime()** function is typically used in place of the **new\_time** parameter.

**double Random()**  
 Returns a pseudo-random number from the  $(0,1)$  interval. The pseudo-random number generator is initialized prior to calling the **init()** function so the sequence is always the same.

**long QGet(var)**  
 Returns the quality of the **var** variable (see the **QFC**, **QFD**, **VIN**, **VOUT** blocks). The function is intended for use with the inputs, outputs and parameters. It always returns 0 for internal variables.

**void QSet(var, value)**  
 Sets the quality of the **var** variable (see the **QFC**, **QFD**, **VIN**, **VOUT** blocks). The function is intended for use with the inputs, outputs and parameters. It has no meaning for internal variables.

**long QPropag([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. A binary file with **double** values or a text file with values on individual lines is supposed. The **idx** index (binary file) or line number (text file) starts at 0. The file is identified by **fileid**. At present the following values are supported:

- 0 ...file on a disk identified by the **p0** parameter
- 1 ...file on disk identified by name of the **REXLANG** block and extension **.dat**
- 2 ...file on a disk identified by the **srcname** parameter, but the extension is changed to **.dat**
- 3 ...**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 the **value** to a file. The meaning of parameters is the same as in the **LoadValue** function.

**void GetSystemTime(time)**  
 Returns the system time. The time is usually returned as UTC but this can be altered by the operating system settings. The **time** parameter must be an array of at least 8 items of type **long**. The function fills the array with

the following values in the given order: year, month, day (in the month), day of week, hours, minutes, seconds, milliseconds. On some platforms the milliseconds value has a limited precision or is not available at all (the function returns 0 ms).

**void Sleep(seconds)**

Stop execution of the block's algorithm (and whole task) for defined time. Shortest possible time is about 0.01s, but depend on 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.

**long memrd32(hMem, offset)**

Reading physical memory. Get the handle by `Open(72, "/dev/mem", <physical address>, <area size>)`.

**long memwr32(hMem, offset, value)**

Writing to physical memory. Get the handle by `OpenMemory("/dev/mem", <physical address>, <area size>)`.

- **Communication functions** (not part of ANSI C)

This set of functions is intended for communication over TCP/IP, UDP/IP, serial line (RS-232 or RS-485), SPI bus and I2C bus. Only a brief list of available functions is given below, see the example projects of the REXYGEN system for more details.

```

long OpenFile(string filename)
    Function for opening a file. Identification number (the so-called handle)
    of the file is returned. If a negative value is returned, the opening was not
    successful.
long OpenCom(string comname, long baudrate, long parity)
    Function for opening a serial line. Identification number (the so-called han-
    dle) of serial port is returned. If a negative value is returned, the opening
    was not successful.
long OpenUDP(string localname, long lclPort, string remotename, long remPort)
    Function for opening a UDP socket. Identification number (the so-called
    handle) of the socket is returned. If a negative value is returned, the open-
    ing was not successful.
long OpenTCPSvr(string localname, long lclPort)
    Function for opening a TCP socket (server, listening). Identification num-
    ber (the so-called handle) of the socket is returned. If a negative value is
    returned, the opening was not successful.
long OpenTCPcli(string remotename, long remPort)
    Function for opening a TCP socket (client). Identification number (the
    so-called handle) of the socket is returned. If a negative value is returned,
    the opening was not successful.
long OpenI2C(string devicename)
    Function for opening the I2C bus. Identification number (the so-called
    handle) of the bus is returned. If a negative value is returned, the opening
    was not successful.
long OpenSPI(string devicename)
    Function for opening the SPI bus. Identification number (the so-called
    handle) of the bus is returned. If a negative value is returned, the opening
    was not successful.
long OpenMemory(string devicename, long baseaddr, long size)
    Function for mapping physical memory. Identification number (the so-
    called handle) of the memory address is returned. If a negative value is
    returned, the opening was not successful.
long Close(long handle)
    Closes the socket, serial line, file or any device opened by the Open...
    functions.
void SetOptions(long handle, long params[])
    Sets the parameters of a socket or serial line. The array size must be at
    least:
        – 22 for serial line,
        – 2 for file (1st item is mode: 1=seek begin, 2=seek current, 3=seek end,
          4=set file end, 2nd item is offset for seek),
        – 3 for SPI (1st item is SPI mode, 2nd item is bits per word, 3rd item
          is max speed in Hz),

```



- 5 for I2C (1st item is slave address, 2nd item is 10-bit address flag, 3rd item is Packet Error Checking flag, 4th item is number of retries, 5th item is timeout)

**void GetOptions(long handle, long params[])**

Reads parameters of a socket or serial line to the **params** array. The array size must be big enough, at least 2 for files, 2 for a socket and 22 for serial line (see **SetOptions**).

**long Accept(long hListen)**

Accepts the connection to listening socket **hListen** invoked by the client. A communication socket handle or an error is returned.

**long Read(long handle, long buffer[], long count)**

Receives data from a serial line or socket. The **count** parameter defines the maximum number of bytes to read. The count of bytes read or an error code is returned. Each byte of incoming data is put to the **buffer** array of type **long** in the corresponding order.

It is also possible to use the form

**long Read(long handle, string data[], long count)** (i.e. a string is used instead of a data array; one byte in the input file corresponds to one character; not applicable to binary files).

The error codes are:

- 1 it is necessary to wait for the operation to finish (the function is "non-blocking")
- 309 reading failed; the operating system error code appears in the log (when function block logging is enabled)
- 307 file/socket is not open

**long Write(long handle, long buffer[], long count)**

Sends the data to a serial line or socket. The **count** parameter defines the number of bytes to send. The count of bytes or an error code sent is returned. Each byte of outgoing data is read from the **buffer** array of type **long** in the corresponding order.

It is also possible to use the form

**long Write(long handle, string data)** (i.e. a string is used instead of a data array; one byte in the output file corresponds to one character; not applicable to binary files).

The error codes are:

- 1 it is necessary to wait for the operation to finish (the function is "non-blocking")
- 310 write failed; the operating system error code appears in the log (when function block logging is enabled)
- 307 file/socket is not open

**long ReadLine(long handle, string data)**

Read one line from (text) file, serial line or socket; read characters are in the variable **data** up to allocated size of the string; the function return real size (number of bytes) of line or error code.

```

long DeleteFile(string filename)
    Delete file. Return 0 if success; negative value is error code.
long RenameFile(string filename, string newfilename)
    Rename file. Return 0 if success; negative value is error code.
long I2C(long handle, long addr, long bufW[], long cntW, long bufR[], long cntR)
    Communication over the I2C bus. Works only in Linux operating system
    on devices with the I2C bus (e.g. Raspberry Pi). Sends and receives data
    to/from the slave device with address addr. The parameter handle is
    returned by the OpenI2C function, whose parameter defines the device
    name (according to the operating system). The parameter bufW is a buffer
    (an array) for the data which is sent out, cntW is the number of bytes to
    send out, bufR is a buffer (an array) for the data which comes in and cntR
    is the number of bytes to receive. The function returns 0 or an error code.
long SPI(long handle, 0, long bufW[], long cntW, long bufR[], long cntR)
    Execution of one transaction over the SPI bus. Works only in Linux operat-
    ing system on devices with the SPI bus (e.g. Raspberry Pi). The parameter
    handle is returned by the OpenSPI function, whose parameter defines the
    device name (according to the operating system). The second parameter
    is always 0 (reserved for internal use). The parameter bufW is a buffer (an
    array) for the data which is sent out, cntW is the number of bytes to send
    out, bufR is a buffer (an array) for the data which comes in and cntR is the
    number of bytes to receive. Note that SPI communication is full-duplex,
    therefore the resulting length of the SPI transaction is given by maximum
    of the cntW and cntR parameters, not their sum. The function returns 0
    or an error code.
long Seek(long handle, long mode[], long offset)
    Set position for Read/Write command. Parameter mode means: 1=offset
    from begin of the file, 2= offset from current position, 3=offset from end
    of the file.
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.

```

## Remarks

- The data type of inputs **u0..u15**, outputs **y0..y15** and parameters **p0..p15** is determined during compilation of the source code according to the **input**, **output** and **parameter** definitions.
- All error codes  $< 0$  require restarting of the REXYGEN executive. Of course it is necessary to remove the cause of the error first.

- **WARNING!** – The inputs and outputs of the block cannot be accessed within the `init()` function (the values of inputs are 0, outputs are not set).
- It is possible to include path in the `srcname` parameter. Otherwise the file is expected directly in the project directory or in the directories specified by the `-I` command line option of the **REXYGEN Compiler** compiler.
- All parameters of the vector functions are of type `double` (or array of type `double`). The only exception is the `n` parameter of type `long`. Note that the functions with one vector parameter exist in three variants:

`double function(val1,...,valn)`

Vector is defined as a sequence of values of type `double`.

`double function(n,val1,...,valn)`

Vector is defined as in the first case, only the first parameter defines the number of values – the size of the vector. This variant is compatible with the C compiler. The `n` parameter must be a number, not the so-called `const` variable and it must correspond with the number of the following elements defining the vector.

`double function(n,vec)`

The `n` parameter is an arbitrary expression of type `long` and defines the number of elements the function takes into account.

- The optional parameter `n` of the vector functions must be specified if the compatibility with C/C++ compiler is required. In such a case all the nonstandard functions must be implemented as well and the functions with variable number of parameters need to know the parameter count.
- In all case it is important to keep in mind that the vectors start at index 0 and that the array limits are not checked (just like in the C language). E.g. if `double vec[10], x;` is defined, the elements have indexes 0 to 9. The expression `x=vec[10];` is neither a syntax nor runtime error, the value is not defined. More importantly, it is possible to write `vec[11]=x;`, which poses a threat, because some other variable might be overwritten and the program works unexpectedly or even crashes.
- Only the **parser error** and line number are reported during compilation. This means a syntax error. If everything seems fine, the problem can be caused by identifier/keyword/function name conflict.
- All jumps are translated as relative, i.e. the corresponding code is restricted to 32767 instructions (in portable format for various platforms).

- All valid variables and temporary results are stored in the stack, namely:
  - Global variables and local **static** variables (permanently at the beginning of the stack)
  - Return addresses of functions
  - Parameters of functions
  - Local function variables
  - Return value of function
  - Temporary results of operations (i.e. the expression **a=b+c**; is evaluated in the following manner: **b** is stored in the stack, **c** is stored in the stack (it follows after **b**), the sum is evaluated, both values are removed from the stack and the result is stored in the stack

Each simple variable (**long** or **double**) thus counts as one item in the stack. For arrays, only the size is important, not the type.

- The arrays are passed to the functions as a reference. This means that the parameter counts as one item in the stack and that the function works directly with the referenced array, not its local copy.
- If the stack size is not sufficient (less than space required for global variables plus 10), the stack size is automatically set to twice the size of the space required for the global variables plus 100 (for computations, function parameters and local variables in the case that only a few global variables are present).
- If basic debug level is selected, several checks are performed during the execution of the script, namely initialization of the values which are read and array index limits. Also a couple of uninitialized values are inserted in front of and at the back of each declared array. The **NOP** instructions with line number of the source file are added to the \*.ill file.
- If full debug is selected, additional check is engaged – the attempts to access invalid data range are monitored (e.g. stack overflow).
- The program size and stack size are set to a fixed value of 16384 in Simulink (for implementation reasons). If this size is exceeded, an error is reported.
- The term instruction in the context of this block refers to an instruction of a processor-independent mnemocode. The mnemocode is stored in the \*.ill file.
- The **Open()** function set serial line always 19200Bd, no parity, 8 bit per character, 1 stopbit, binary mode, no timeout. Optional 2nd (bitrate) and 3th (parity) parametrns can be used in the **Open()** function.
- Accessing text file is significantly slower than binary file. A advantage of the text file is possibility view/edit data in file without special editor.

- This block does not call the `parchange()` function. It is necessary to call it in `init()` function (if it is required).
- The block's inputs are available in the `init()` function, but all are equal to zero. It is possible (but not common) to set block's outputs.
- The `Open()` function also allows opening of a regular file. Same codes like in the `LoadValue()` function are used.

## Debugging the code

Use the `Trace` command mentioned above.

### Inputs

HLD	Hold – the block code is not executed if the input is set to on	bool
RESET	Rising edge resets the block. The block gets initialized again (all global variables are cleared and the <code>Init()</code> function is called).	bool
u0..u15	Input signals which are accessible from the script	unknown

### Outputs

iE	Runtime error code. Unless <code>iE = 0</code> or <code>iE = -1</code> the algorithm is stopped until it is reinitialized by the <code>RESET</code> input or by restarting the executive) <ul style="list-style-type: none"> <li>0 ..... No error occurred, the whole <code>main()</code> function was executed (also the <code>init()</code> function).</li> <li>-1 ..... The execution was suspended using the <code>Suspend()</code> command, i.e. the execution will resume as soon as the <code>REXLANG</code> block is executed again</li> <li>xxx ... Error code of the <code>REXYGEN</code> system, see Appendix C</li> </ul>	error
y0..y15	Output signals which can be set from within the script	unknown

### Parameters

srcname	Source file name	⊙srcfile.c	string
---------	------------------	------------	--------

<b>srctype</b>	Coding of source file	⊙1	long
	1: C-like Text file respecting the C-like syntax described above		
	2: STL Text file respecting the IEC61131-3 standard. The standard is implemented with the same limitations as the C-like script (i.e. no structures, only INT, REAL and STRING data types, function blocks are global variables VAR_INPUT, outputs are global variables VAR_OUTPUT, parameters are global variables VAR_PARAMETER, standard functions according to specification, system and communication functions are the same as in C-like).		
	3: RLB REXLANG binary file which results from compilation of C-like or STL scripts. Use this option if you do not wish to share the source code of your block.		
	4: ILL Text file with mnemocodes, which can be compared to assembler. This choice is currently not supported.		
<b>stack</b>	Stack size defined as number of variables. Default and recommended value is 0, which enables automatic estimation of the necessary stack size.		long
<b>debug</b>	Debug level – checking is safer but slows down the execution of the algorithm. Option No check can crash REXYGEN application on target platform if code is incorrect.	⊙3	long
	1 ..... No check		
	2 ..... Basic check		
	3 ..... Full check		
<b>strs</b>	Total size of buffer for strings. Enter the maximum number of characters to allocate memory for. The default value 0 means that the buffer size is determined automatically.		long
<b>p0..p15</b>	Parameters which are accessible from the script		unknown

### Example C-like

The following example shows a simple code to sum two input signals and also sum two user-defined parameters.

```
double input(0) input_u0;
double input(2) input_u2;

double parameter(0) param_p0;
double parameter(1) param_p1;

double output(0) output_y0;
double output(1) output_y1;

double my_value;

long init(void)
```

```

{
    my_value = 3.14;
    return 0;
}

long main(void)
{
    output_y0 = input_u0 + input_u2;
    output_y1 = param_p0 + param_p1 + my_value;
    return 0;
}

long exit(void)
{
    return 0;
}

```

## Example STL

And here is the same example in Structured Text.

```

VAR_INPUT
    input_u0:REAL;
    input_u1:REAL;
    input_u2:REAL;
END_VAR

VAR_OUTPUT
    output_y0:REAL;
    output_y1:REAL;
END_VAR

VAR_PARAMETER
    param_p0:REAL;
    param_p1:REAL;
END_VAR

VAR
    my_value: REAL;
END_VAR

FUNCTION init : INT;
    my_value := 3.14;
    init := 0;
END_FUNCTION

```

```
FUNCTION main : INT;  
    output_y0 := input_u0 + input_u2;  
    output_y1 := param_p0 + param_p1 + my_value;  
    main := 0;  
END_FUNCTION
```

```
FUNCTION exit : INT;  
    exit := 0;  
END_FUNCTION
```



## Chapter 16

# MC\_SINGLE – Motion control - single axis blocks

### Contents

---

RM_Axis – Motion control axis . . . . .	480
MC_AccelerationProfile, MCP_AccelerationProfile – Acceleration profile . . . . .	486
MC_Halt, MCP_Halt – Stopping a movement (interruptible) . . . . .	490
MC_HaltSuperimposed, MCP_HaltSuperimposed – Stopping a move- ment (superimposed and interruptible) . . . . .	492
MC_Home, MCP_Home – Homing . . . . .	494
MC_MoveAbsolute, MCP_MoveAbsolute – Move to position (absolute coordinate) . . . . .	496
MC_MoveAdditive, MCP_MoveAdditive – Move to position (relative to previous motion) . . . . .	500
MC_MoveRelative, MCP_MoveRelative – Move to position (relative to execution point) . . . . .	503
MC_MoveSuperimposed, MCP_MoveSuperimposed – Superimposed move	506
MC_MoveContinuousAbsolute, MCP_MoveContinuousAbsolute – Move to position (absolute coordinate) . . . . .	509
MC_MoveContinuousRelative, MCP_MoveContinuousRelative – Move to position (relative to previous motion) . . . . .	513
MC_MoveVelocity, MCP_MoveVelocity – Move with constant velocity	517
MC_PositionProfile, MCP_PositionProfile – Position profile . . . . .	521
MC_Power – Axis activation (power on/off) . . . . .	525
MC_ReadActualPosition – Read actual position . . . . .	527
MC_ReadAxisError – Read axis error . . . . .	528
MC_ReadBoolParameter – Read axis parameter (bool) . . . . .	529
MC_ReadParameter – Read axis parameter . . . . .	530

MC_ReadStatus – Read axis status . . . . .	532
MC_Reset – Reset axis errors . . . . .	534
MC_SetOverride, MCP_SetOverride – Set override factors . . . . .	535
MC_Stop, MCP_Stop – Stopping a movement . . . . .	537
MC_TorqueControl, MCP_TorqueControl – Torque/force control . . . . .	539
MC_VelocityProfile, MCP_VelocityProfile – Velocity profile . . . . .	542
MC_WriteBoolParameter – Write axis parameter (bool) . . . . .	546
MC_WriteParameter – Write axis parameter . . . . .	547
RM_AxisOut – Axis output . . . . .	549
RM_AxisSpline – Commanded values interpolation . . . . .	551
RM_Track – Tracking and inching . . . . .	553

---

This library includes functional blocks for single axis motion control as it is defined in the PLCopen specification. It is recommended to study the PLCopen specification prior to using the blocks from this library. The knowledge of PLCopen is necessary for advanced use of the blocks included in this library.

PLCopen defines all blocks with the MC\_ prefix. This notation is kept within this library. Nevertheless, there are also function blocks, which are not described by PLCopen or are described as *vendor specific*. These blocks can be recognized by the RM\_ prefix. Note that PLCopen (and also IEC 61131-3 which is the base for PLCopen) does not use block parameters, all the parameters are specified by input signals. In the REXYGEN, block parameters are used to simplify usage of the blocks. To keep compatibility with PLCopen and improve usability of the blocks, almost all of them are implemented twice: with prefix MC\_ without parameters (parameters are inputs) and with prefix MCP\_ with parameters. Some blocks require additional *vendor specific* parameters. In such a case even the MC\_-prefixed blocks contain parameters.

PLCopen specifies that all inputs/parameters are sampled at rising-edge of the Execute input. In REXYGEN block parameters are usually changed very rare. Therefore the parameters of the activated block have not be changed until block is finished (e.g. while output Busy is on).

The REXYGEN system does not allow input-output signals and all signals must have different name. For these reasons the Axis input-output signal, which is used in all blocks, is divided into input uAxis and output yAxis. The block algorithm copies the input uAxis to the output yAxis. The yAxis output is not necessary for the function of motion control blocks, but "chaining" the axis references makes it possible to order the blocks and define priorities. Other reference signals are either defined as input-only or use this mechanism as well.

PLCopen defines the outputs Busy, Active, CommandAborted as optional in almost all blocks. In REXYGEN, some of them are never set, but the outputs are defined to simplify future extensions and/or changes in the implementation.

Units used for position and distance of axis are implementation specific. It can be meters, millimeters, encoder ticks, angular degrees (for rotational axis) or any others,

but all blocks connected to one axis must use the same position units. Time is always defined in seconds. Velocity unit is thus "position units per second" and acceleration unit is "position units per square second".

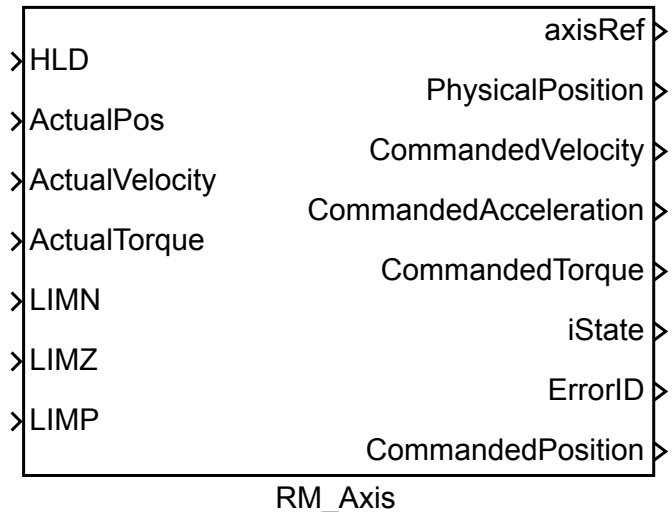
The REXYGEN system uses more threads for execution of the function blocks. In standard function blocks the synchronization is provided by the system and the user does not need to care about it. But using the **reference** references could violate the synchronization mechanisms. However, there is no problem if all referenced blocks are located in the same task and therefore e.g. the [RM\\_Axis](#) block must be in the same task as all other blocks connected to this axis.

Some inputs/parameters are of enumeration type (for example **BufferMode** or **Direction**). It is possible to choose any of the defined values for this type in the **MCP\_** version of the blocks, although not all of them are valid for all blocks (for example the block [MC\\_MoveVelocity](#) does not support **Direction = shortest\_way**). Valid values for each block are listed in this manual.

### RM\_Axis – Motion control axis

Block Symbol

Licence: [MOTION CONTROL](#)



### Function Description

The **RM\_AXIS** block is a cornerstone of the motion control solution within the REXYGEN system. This base block keeps all status values and implements basic algorithm for one motion control axis (one motor), which includes limits checking, emergency stop, etc. The block is used for both real and virtual axes. The real axis must have a position feedback controller, which is out of this block's scope. The key status values are commanded position, velocity, acceleration and torque, as well as state of the axis, axis error code and a reference to the block, which controls the axis.

This block (like all blocks in the motion control library) does not implement a feedback controller which would keep the actual position as near to the commanded position as possible. Such a controller must be provided by using other blocks (e.g. [PIDU](#)) or external (hardware) controller. The feedback signals are used for lag checking, homing and could be used in special motion control blocks.

The parameters of this block correspond with the requirements of the **PLCopen** standard for an axis. If improper parameters are set, the **errorID** output is set to -700 (invalid parameter) and all motion blocks fail with the -720 error code (general failure).

Note that the default values for position, velocity and acceleration limits are intentionally set to 0, which makes them invalid. Limits must always be set by the user according to the real axis and the axis actuator.

## Inputs

HLD	Hold	bool
	off ... Motion is allowed	
	on .... Axis is halted and no motion is possible	
ActualPos	Current position of the axis (feedback)	double
ActualVelocity	Current velocity of the axis (feedback)	double
ActualTorque	Current torque in the axis (feedback)	double
LIMN	Limit switch in negative direction	bool
LIMZ	Absolute switch or reference pulse for homing	bool
LIMP	Limit switch in positive direction	bool

## Outputs

axisRef	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	reference
CommandedPosition	Requested (commanded) position of the axis	double
CommandedVelocity	Requested (commanded) velocity of the axis	double
CommandedAcceleration	Requested (commanded) acceleration of the axis	double
CommandedTorque	Requested (commanded) torque in the axis	double
iState	State of the axis	long
	0 ..... Disabled	
	1 ..... Stand still	
	2 ..... Homing	
	3 ..... Discrete motion	
	4 ..... Continuous motion	
	5 ..... Synchronized motion	
	6 ..... Coordinated motion	
	7 ..... Stopping	
	8 ..... Error stop	
ErrorID	Result of the last operation	error
	i ..... REXYGEN general error	

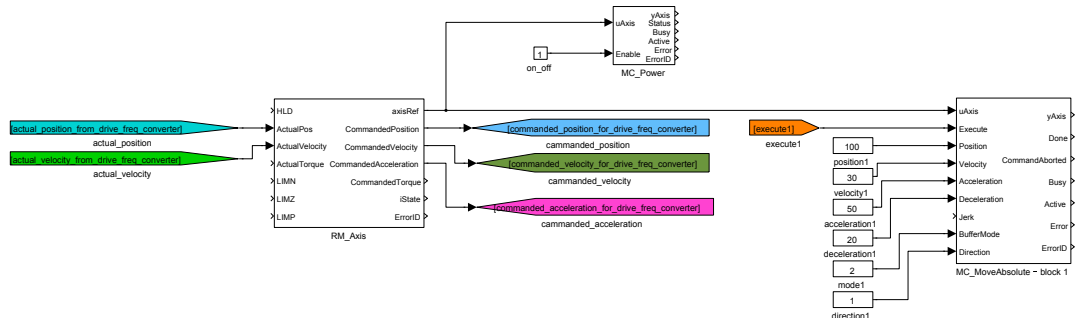
## Parameters

AxisType	Type of the axis	⊙1	long
	1 ..... Linear axis		
	2 ..... Cyclic axis with cyclic position sensor		
	3 ..... Cyclic axis with linear position sensor		
EnableLimitPos	Enable positive position limit checking		bool
SWLimitPos	Positive position limit for application (MC blocks)		double
MaxPosSystem	Positive position limit for system		double
EnableLimitNeg	Enable negative position limit checking		bool
SWLimitNeg	Negative position limit for application (MC blocks)		double
MinPosSystem	Negative position limit for system		double
EnablePosLagMonitor	Enable monitoring of position lag		bool

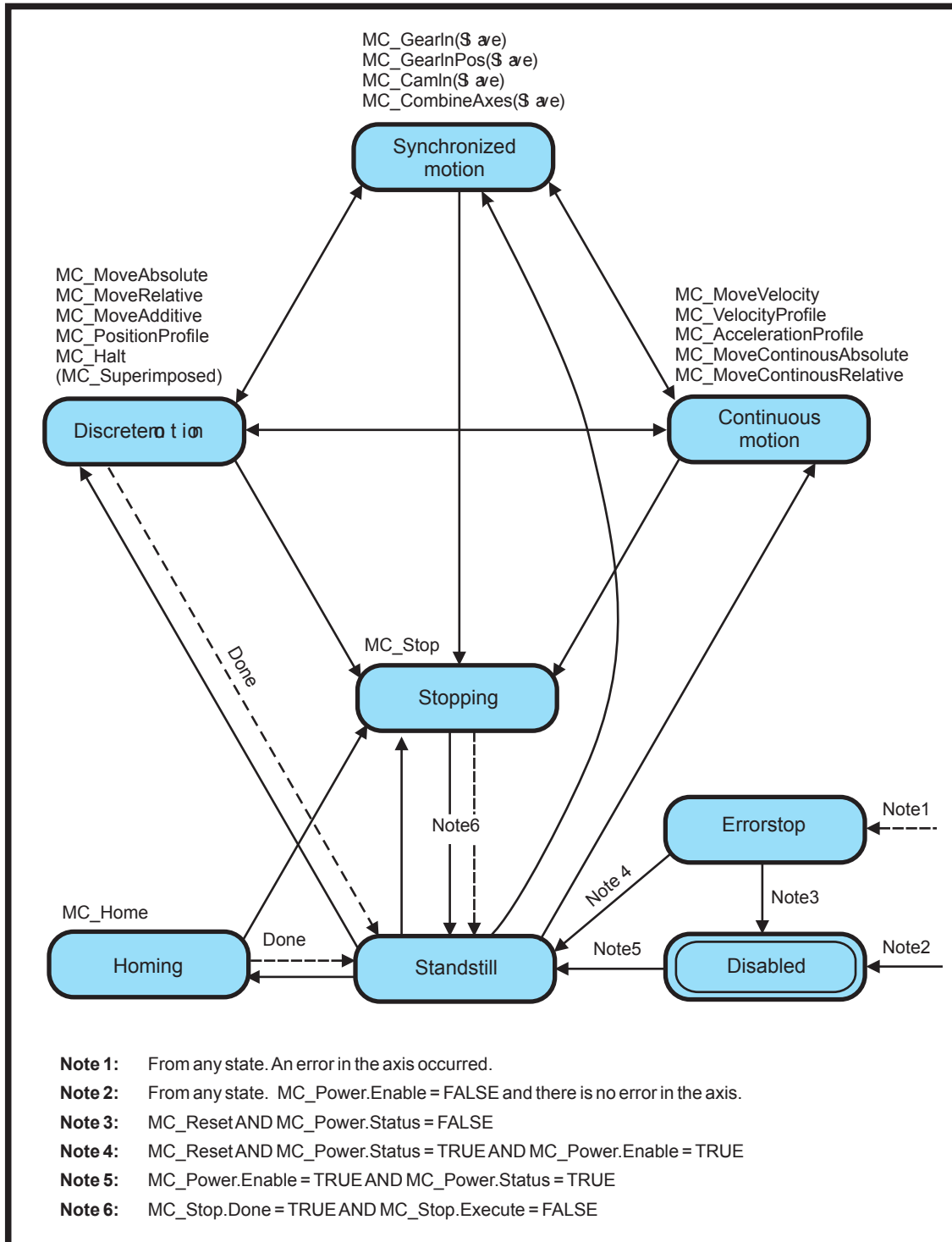
<b>MaxPositionLag</b>	Maximal position lag	double
<b>MaxVelocitySystem</b>	Maximal allowed velocity for system	double
<b>MaxVelocityApp1</b>	Maximal allowed velocity for application (MC blocks)	double
<b>MaxAccelerationSystem</b>	Maximal allowed acceleration for system	double
<b>MaxAccelerationApp1</b>	Maximal allowed acceleration for application (MC blocks)	double
<b>MaxDecelerationSystem</b>	Maximal allowed deceleration for system	double
<b>MaxDecelerationApp1</b>	Maximal allowed deceleration for application (MC blocks)	double
<b>MaxJerk</b>	Maximal allowed jerk [unit/ $s^3$ ]	double
<b>MaxTorque</b>	Maximal motor torque/force (0=not used)	double
<b>kta</b>	Torque-Acceleration ratio	double
<b>ReverseLimit</b>	Invert meaning of LIMN, LIMZ and LIMP inputs	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



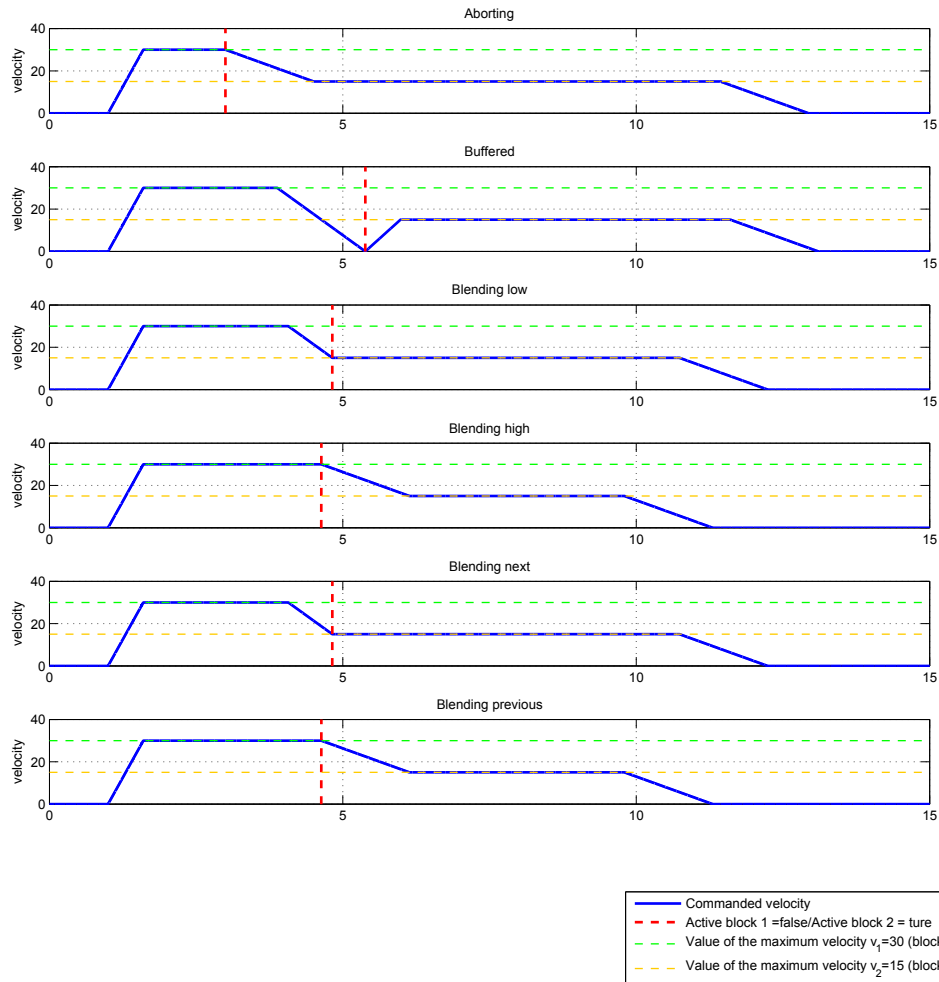
## Motion blending

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.

<b>Aborting</b>	The new motion is executed immediately
<b>Buffered</b>	the new motion is executed immediately after finishing the previous one, there is no blending
<b>Blending low</b>	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
<b>Blending high</b>	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
<b>Blending previous</b>	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
<b>Blending next</b>	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

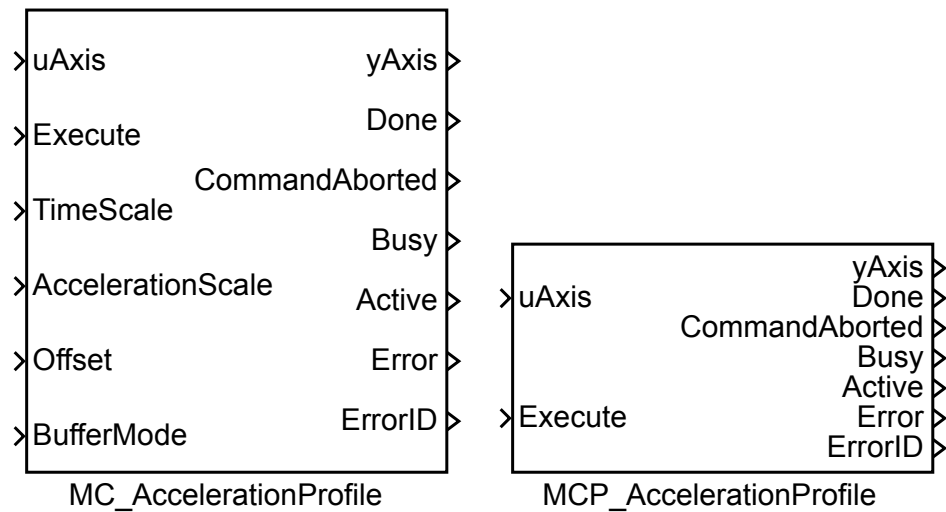


## Illustration of blending modes



## 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

<b>uAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Execute</b>	The block is activated on rising edge	bool
<b>TimeScale</b>	Overall scale factor in time	double
<b>AccelerationScale</b>	Overall scale factor in value	double
<b>Offset</b>	Overall profile offset in value	double
<b>BufferMode</b>	Buffering mode	long
	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

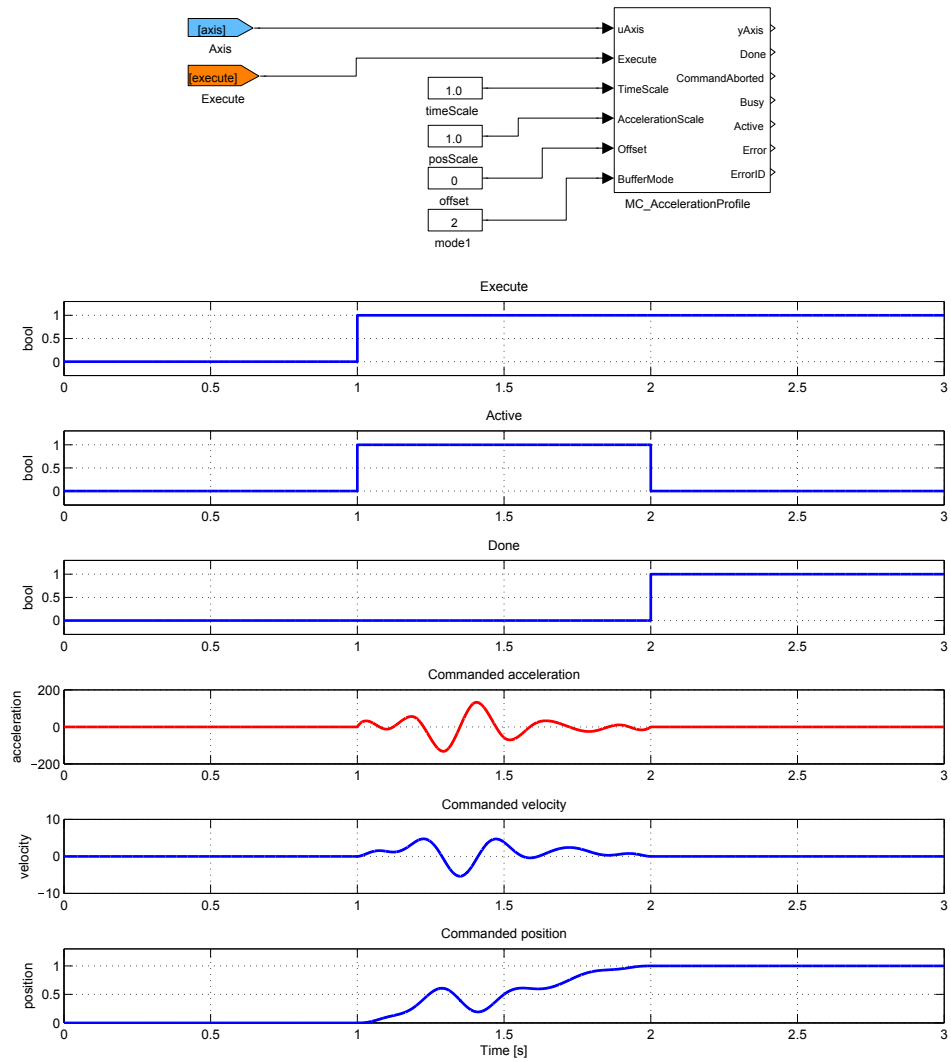
<b>yAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)
--------------	--

Done	Algorithm finished	bool
CommandAborted	Algorithm was aborted	bool
Busy	Algorithm not finished yet	bool
Active	The block is controlling the axis	bool
Error	Error occurred	bool
ErrorID	Result of the last operation	error
	i ..... REXYGEN general error	

## Parameters

alg	Algorithm for interpolation	⊙2	long
	1 ..... Sequence of time/value pairs		
	2 ..... Sequence of equidistant values		
	3 ..... Spline		
	4 ..... Equidistant spline		
cSeg	Number of profile segments	⊙3	long
times	Times when segments are switched	⊙[0 30]	double
values	Values or interpolating polynomial coefficients (a0, a1, a2, ...)		double
		⊙[0 100 100 50]	

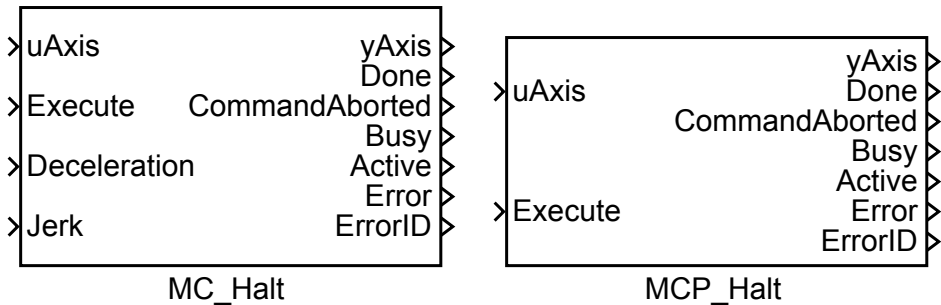
## Example



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 reference connections are allowed)	
Execute	The block is activated on rising edge	bool
Deceleration	Maximal allowed deceleration [unit/s <sup>2</sup> ]	double
Jerk	Maximal allowed jerk [unit/s <sup>3</sup> ]	double

Outputs

yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)	
Done	Algorithm finished	bool
CommandAborted	Algorithm was aborted	bool
Busy	Algorithm not finished yet	bool
Active	The block is controlling the axis	bool
Error	Error occurred	bool

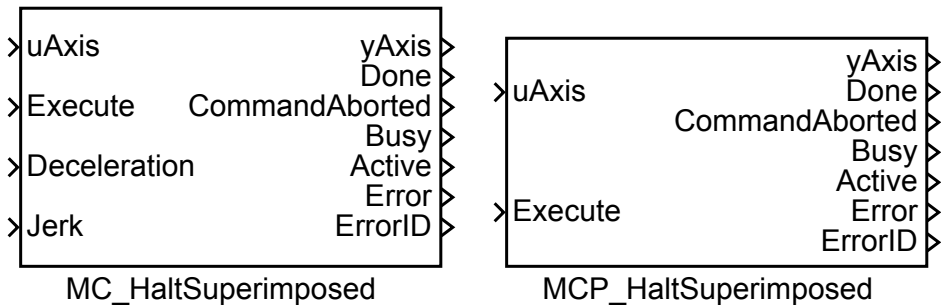
ErrorID	Result of the last operation
i .....	REXYGEN general error

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 reference connections are allowed)	
Execute	The block is activated on rising edge	bool
Deceleration	Maximal allowed deceleration [unit/s <sup>2</sup> ]	double
Jerk	Maximal allowed jerk [unit/s <sup>3</sup> ]	double

**Outputs**

yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)	
Done	Algorithm finished	bool
CommandAborted	Algorithm was aborted	bool
Busy	Algorithm not finished yet	bool
Active	The block is controlling the axis	bool
Error	Error occurred	bool

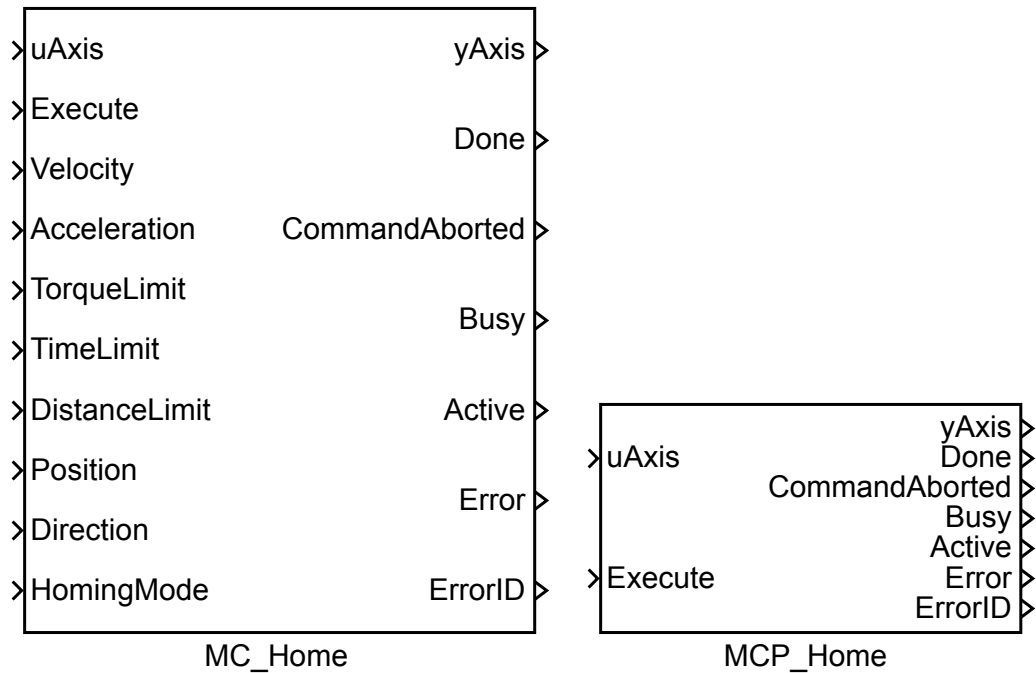


ErrorID	Result of the last operation
i .....	REXYGEN general error

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

<b>uAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Execute</b>	The block is activated on rising edge	bool
<b>Velocity</b>	Maximal allowed velocity [unit/s]	double
<b>Acceleration</b>	Maximal allowed acceleration [unit/s <sup>2</sup> ]	double
<b>TorqueLimit</b>	Maximal allowed torque/force	double
<b>TimeLimit</b>	Maximal allowed time for the whole algorithm [s]	double
<b>DistanceLimit</b>	Maximal allowed distance for the whole algorithm [unit]	double
<b>Position</b>	Requested target position (absolute) [unit]	double
<b>Direction</b>	Direction of movement (cyclic axis or special case only)	long
	1 ..... Positive	
	2 ..... Shortest	
	3 ..... Negative	
	4 ..... Current	
<b>HomingMode</b>	Homing mode algorithm	long
	1 ..... Absolute switch	
	2 ..... Limit switch	
	3 ..... Reference pulse	
	4 ..... Direct (user reference)	
	5 ..... Absolute encoder	
	6 ..... Block	

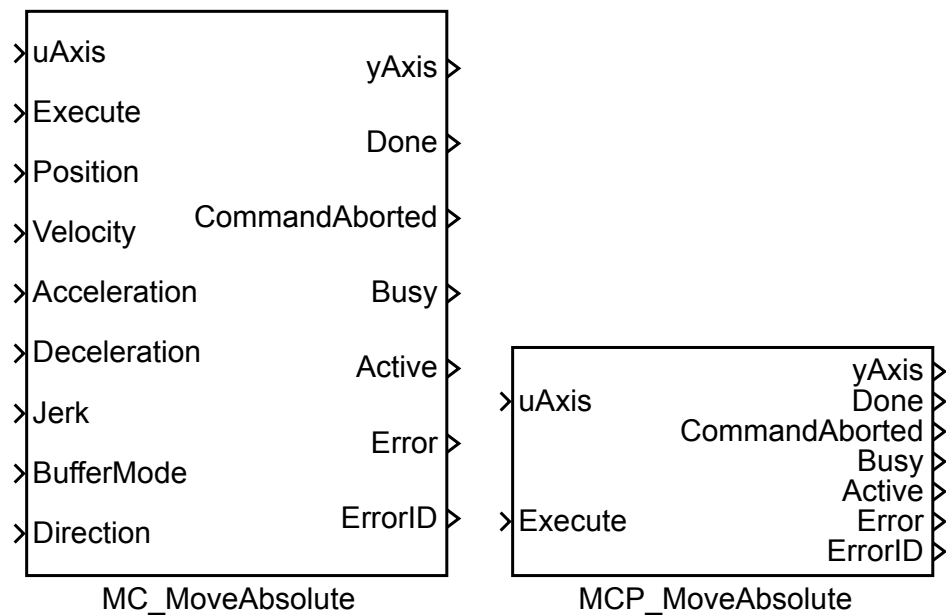
## Outputs

<b>yAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Done</b>	Algorithm finished	bool
<b>CommandAborted</b>	Algorithm was aborted	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Active</b>	The block is controlling the axis	bool
<b>Error</b>	Error occurred	bool
<b>ErrorID</b>	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.

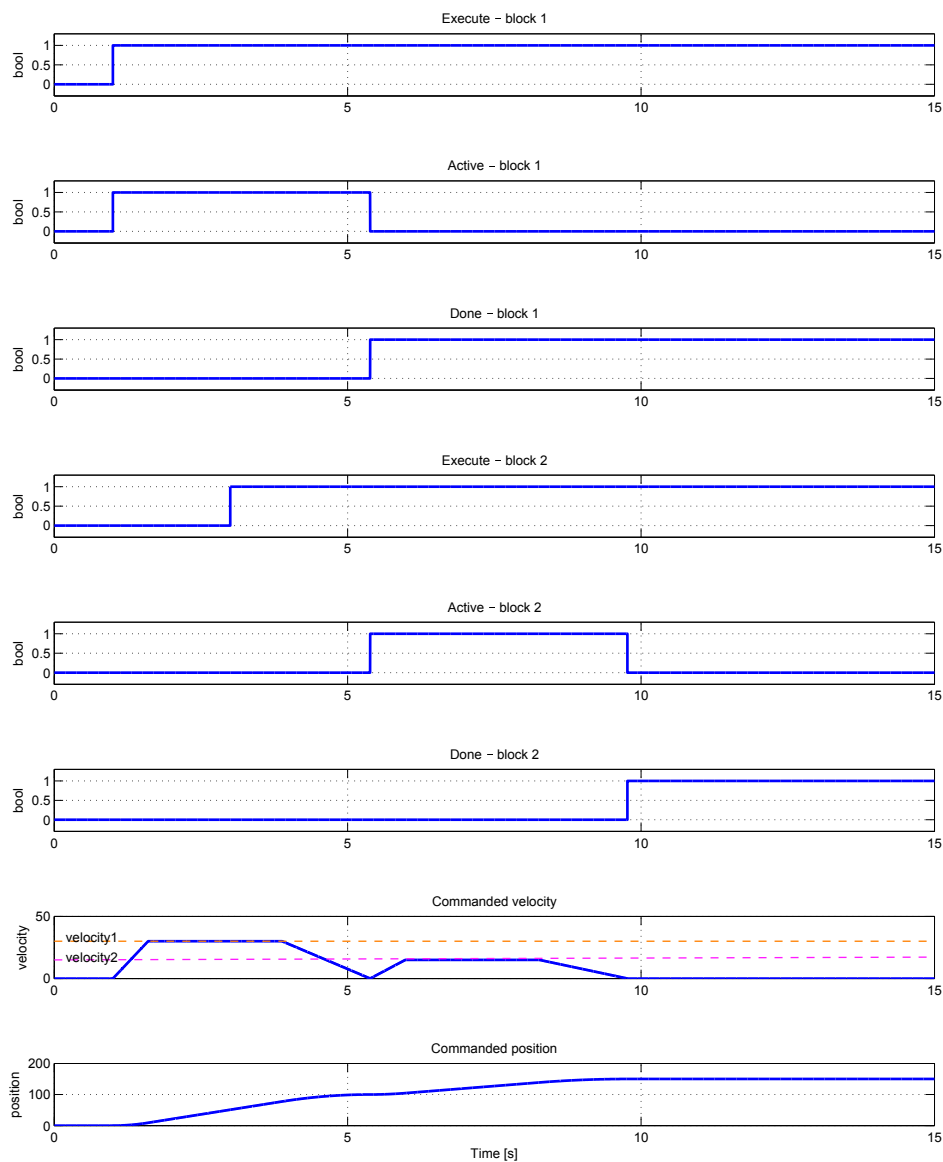
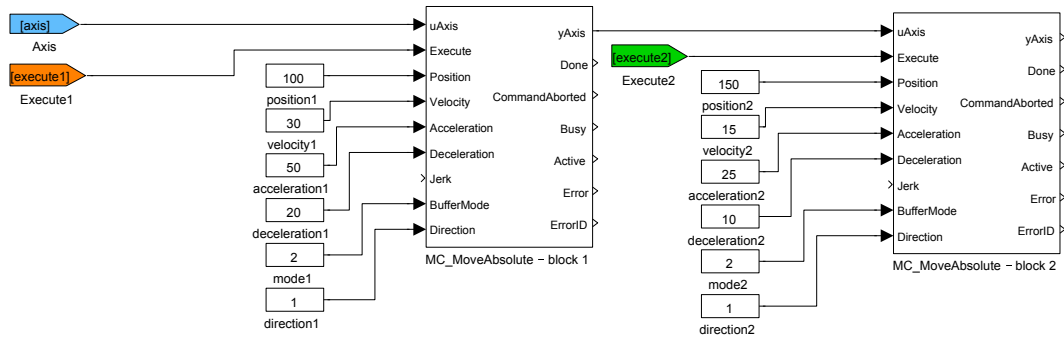
## Inputs

<b>uAxis</b>	AAxis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Execute</b>	The block is activated on rising edge	bool
<b>Position</b>	Requested target position (absolute) [unit]	double
<b>Velocity</b>	Maximal allowed velocity [unit/s]	double
<b>Acceleration</b>	Maximal allowed acceleration [unit/s <sup>2</sup> ]	double
<b>Deceleration</b>	Maximal allowed deceleration [unit/s <sup>2</sup> ]	double
<b>Jerk</b>	Maximal allowed jerk [unit/s <sup>3</sup> ]	double
<b>BufferMode</b>	Buffering mode	long
	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)	
<b>Direction</b>	Direction of movement (cyclic axis or special case only)	long
	1 ..... Positive	
	2 ..... Shortest	
	3 ..... Negative	
	4 ..... Current	

## Outputs

<code>yAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> reference connections are allowed)	
<code>Done</code>	Algorithm finished	<code>bool</code>
<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 the last operation	<code>error</code>
	<i>i</i> ..... REXYGEN general error	

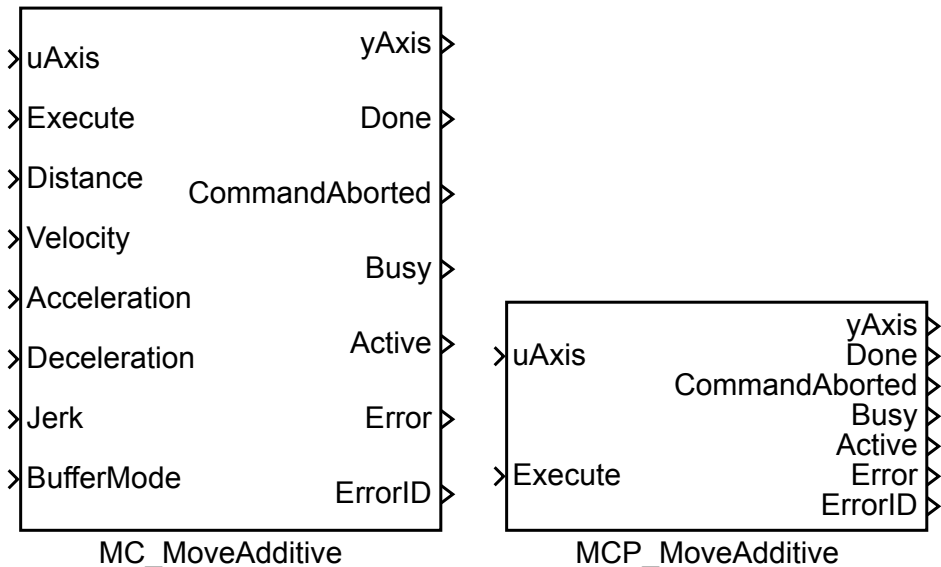
## Example



# 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.

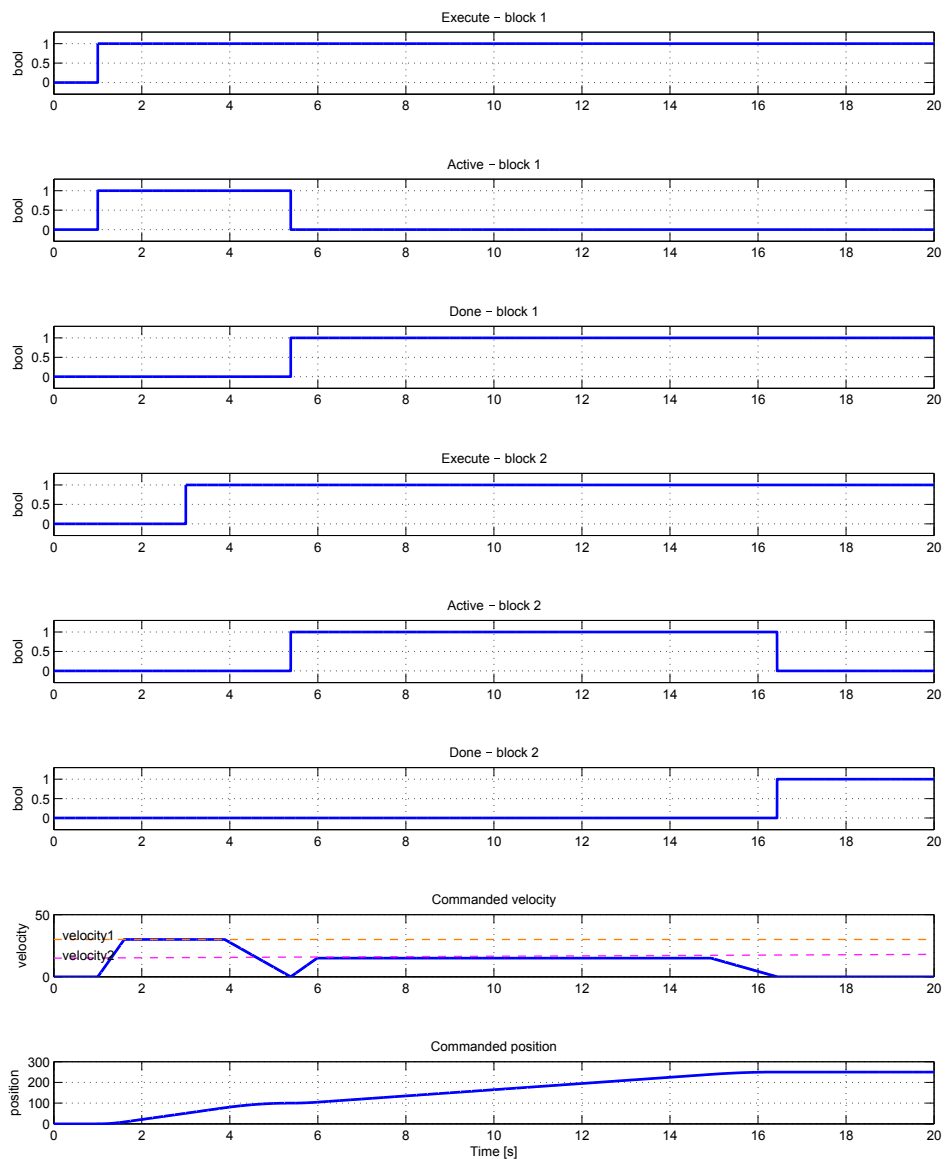
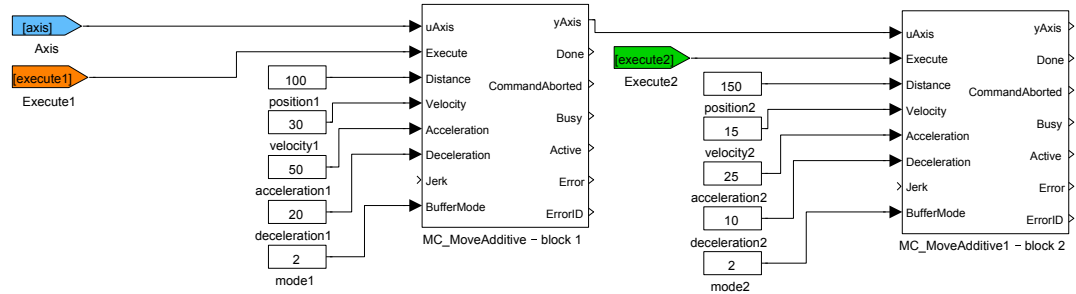
## Inputs

<b>uAxis</b>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> reference connections are allowed)	
<b>Execute</b>	The block is activated on rising edge	bool
<b>Distance</b>	Requested target distance (relative to start point) [unit]	double
<b>Velocity</b>	Maximal allowed velocity [unit/s]	double
<b>Acceleration</b>	Maximal allowed acceleration [unit/s <sup>2</sup> ]	double
<b>Deceleration</b>	Maximal allowed deceleration [unit/s <sup>2</sup> ]	double
<b>Jerk</b>	Maximal allowed jerk [unit/s <sup>3</sup> ]	double
<b>BufferMode</b>	Buffering mode	long
	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

<b>yAxis</b>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> reference connections are allowed)	
<b>Done</b>	Algorithm finished	bool
<b>CommandAborted</b>	Algorithm was aborted	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Active</b>	The block is controlling the axis	bool
<b>Error</b>	Error occurred	bool
<b>ErrorID</b>	Result of the last operation	error
	i ..... REXYGEN general error	

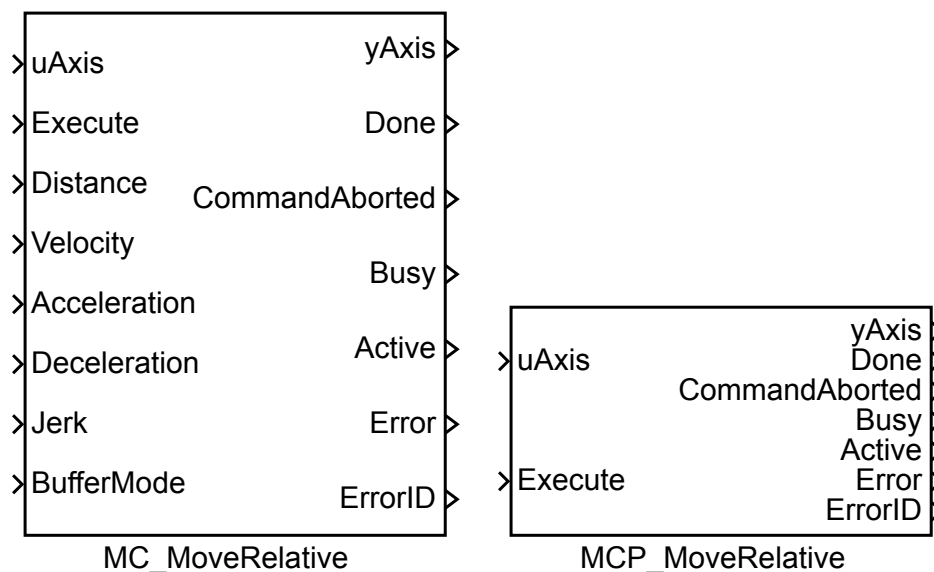
## Example



## 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.

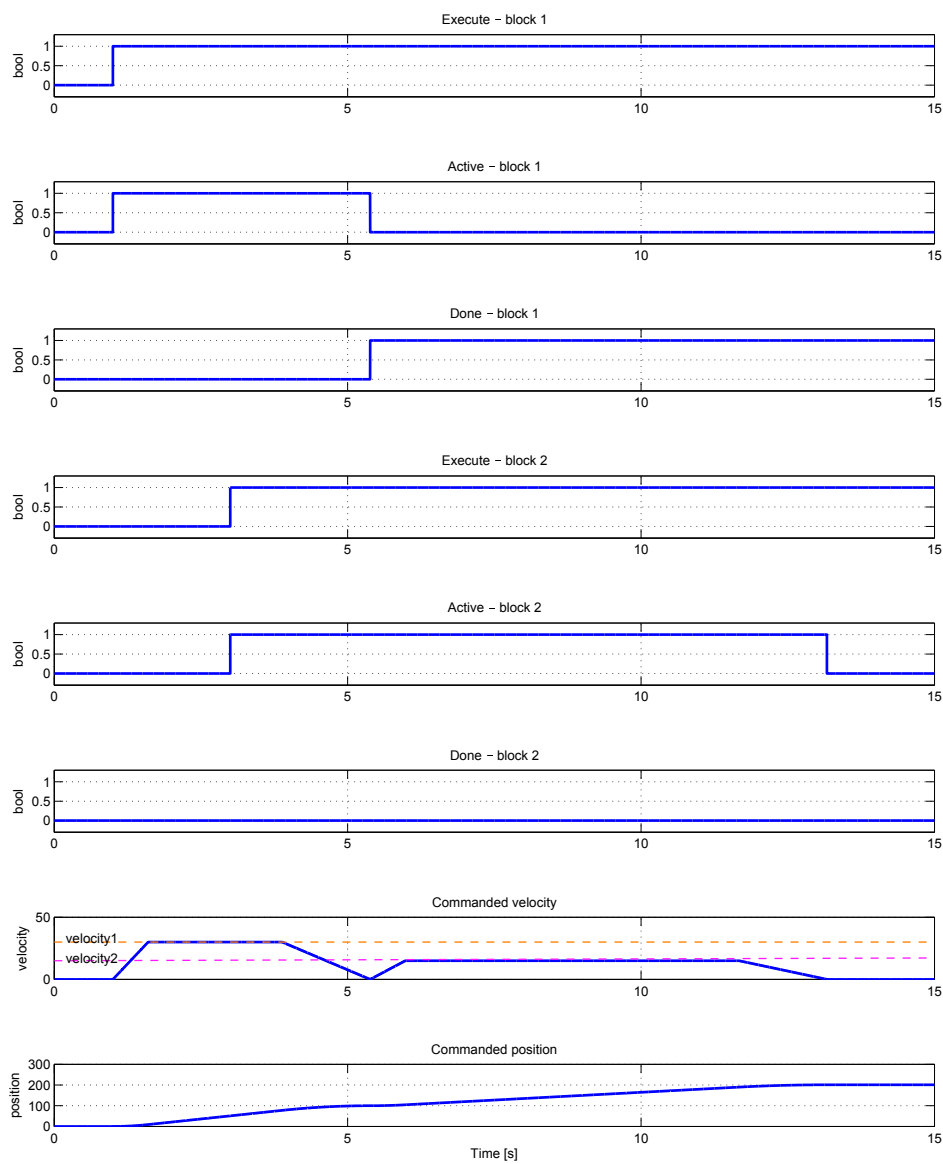
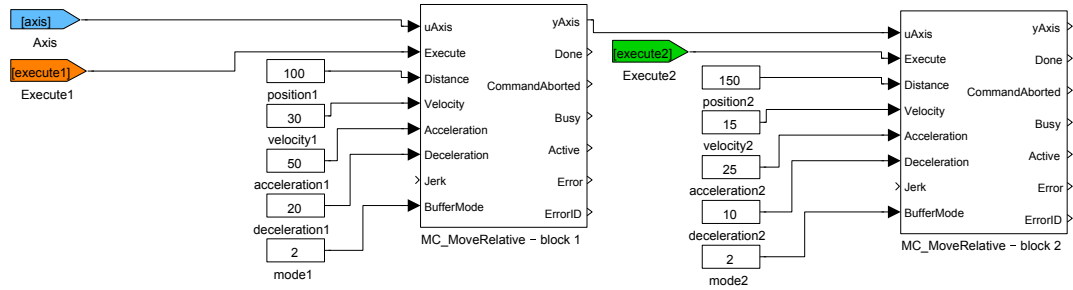
## Inputs

<b>uAxis</b>	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)	
<b>Execute</b>	The block is activated on rising edge	bool
<b>Distance</b>	Requested target distance (relative to execution point) [unit]	double
<b>Velocity</b>	Maximal allowed velocity [unit/s]	double
<b>Acceleration</b>	Maximal allowed acceleration [[unit/s <sup>2</sup> ]	double
<b>Deceleration</b>	Maximal allowed deceleration [[unit/s <sup>2</sup> ]	double
<b>Jerk</b>	Maximal allowed jerk [[unit/s <sup>3</sup> ]	double
<b>BufferMode</b>	Buffering mode	long
	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

<b>yAxis</b>	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)	
<b>Done</b>	Algorithm finished	bool
<b>CommandAborted</b>	Algorithm was aborted	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Active</b>	The block is controlling the axis	bool
<b>Error</b>	Error occurred	bool
<b>ErrorID</b>	Result of the last operation	error
	i ..... REXYGEN general error	

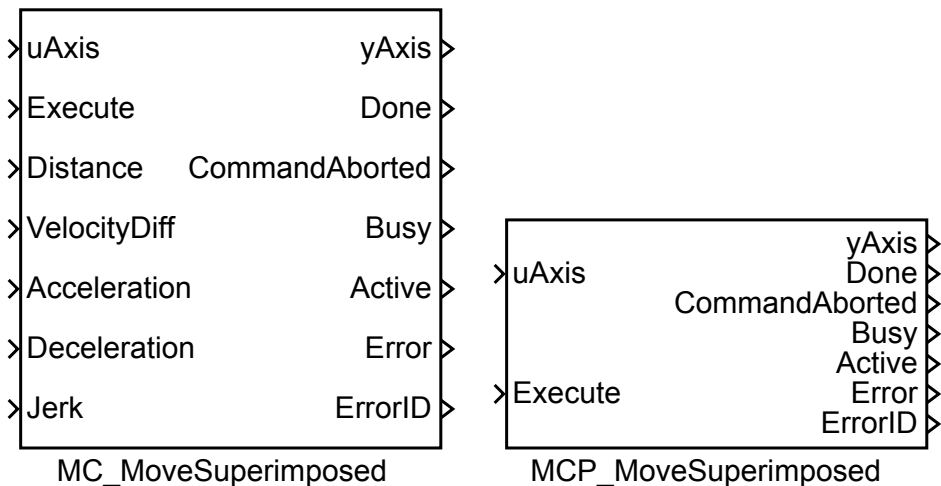
## Example



# 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

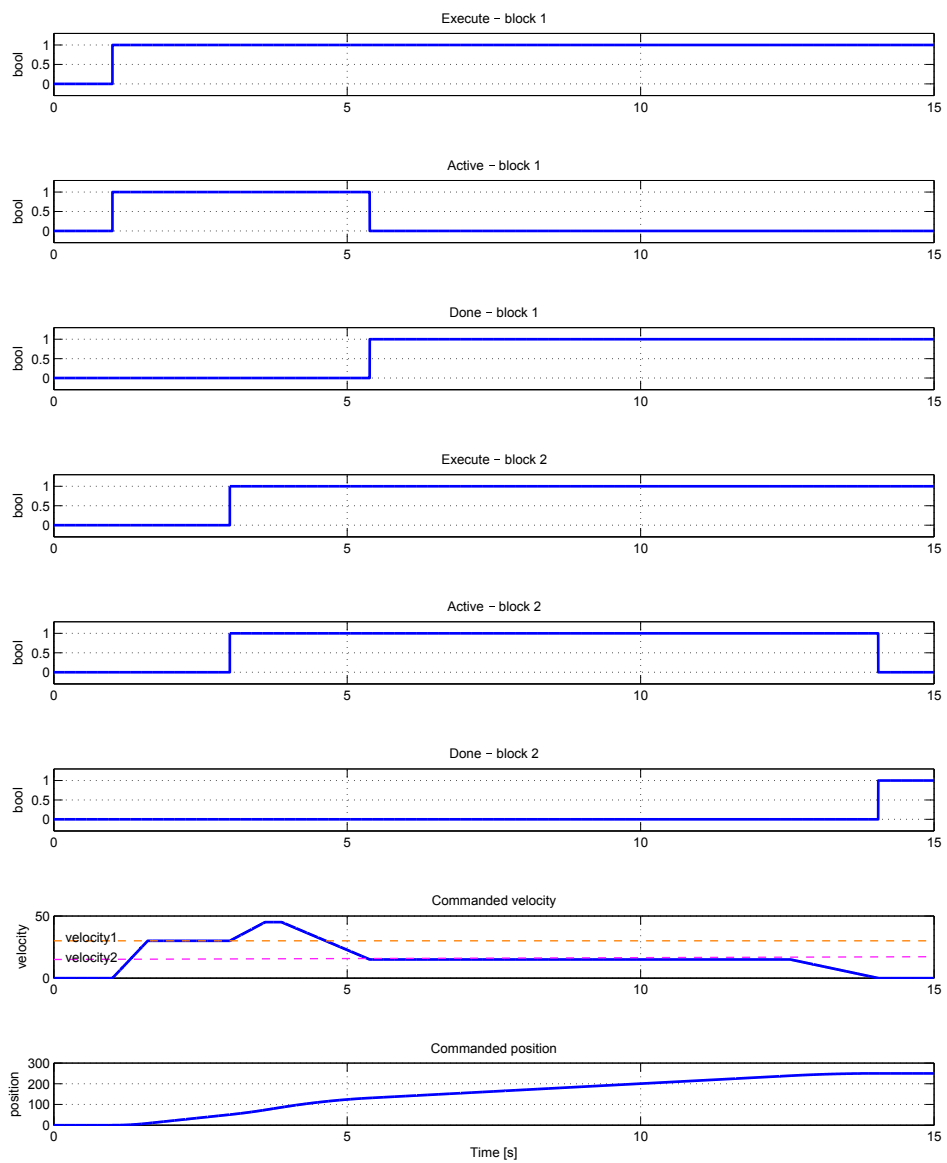
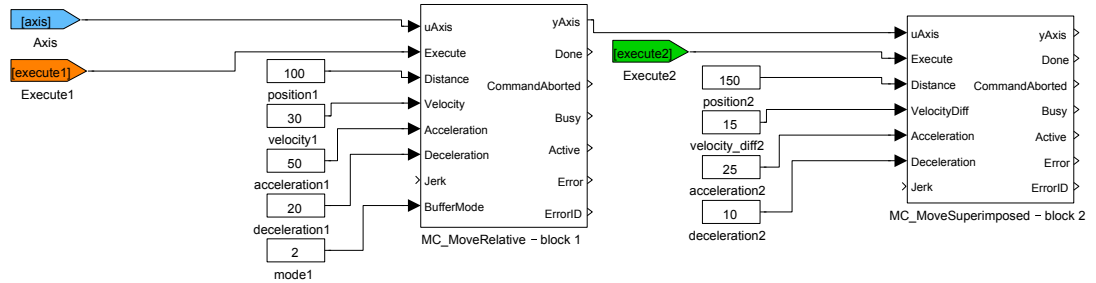
uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)	
Execute	The block is activated on rising edge	bool
Distance	Requested target distance (relative to execution point) [unit]	double

<b>VelocityDiff</b>	Maximal allowed velocity [unit/s]	double
<b>Acceleration</b>	Maximal allowed acceleration [unit/s <sup>2</sup> ]	double
<b>Deceleration</b>	Maximal allowed deceleration [unit/s <sup>2</sup> ]	double
<b>Jerk</b>	Maximal allowed jerk [unit/s <sup>3</sup> ]	double

## Outputs

<b>yAxis</b>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)	reference
<b>Done</b>	Algorithm finished	bool
<b>CommandAborted</b>	Algorithm was aborted	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Active</b>	The block is controlling the axis	bool
<b>Error</b>	Error occurred	bool
<b>ErrorID</b>	Result of the last operation	error
	i ..... REXYGEN general error	

## Example

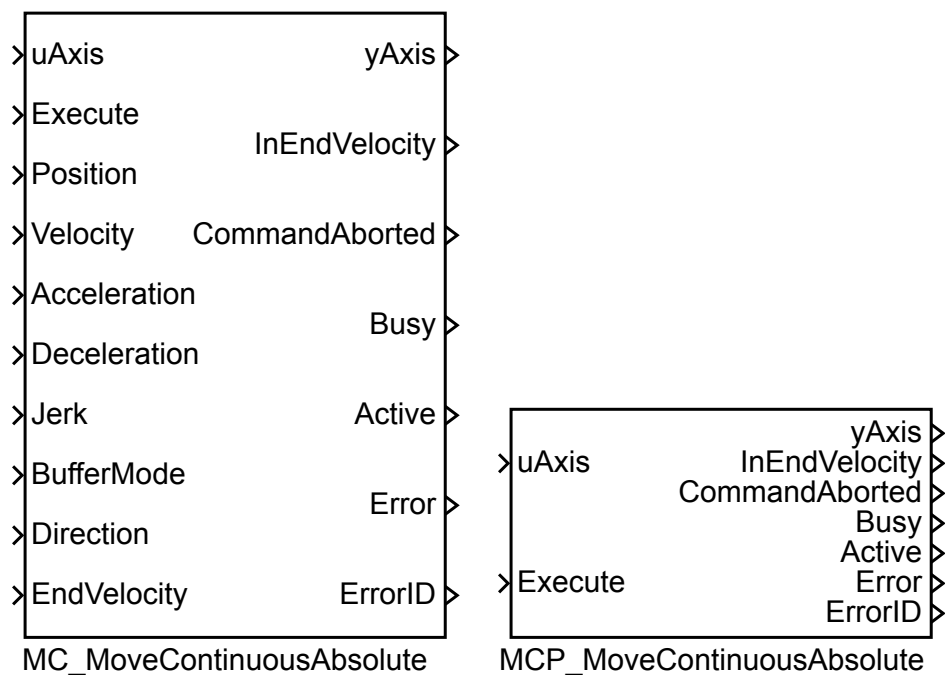




## MC\_MoveContinuousAbsolute, MCP\_MoveContinuousAbsolute – Move to position (absolute coordinate)

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

<b>uAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Execute</b>	The block is activated on rising edge	bool
<b>Position</b>	Requested target position (absolute) [unit]	double
<b>Velocity</b>	Maximal allowed velocity [unit/s]	double
<b>Acceleration</b>	Maximal allowed acceleration [unit/s <sup>2</sup> ]	double
<b>Deceleration</b>	Maximal allowed deceleration [unit/s <sup>2</sup> ]	double
<b>Jerk</b>	Maximal allowed jerk [unit/s <sup>3</sup> ]	double
<b>BufferMode</b>	Buffering mode	long
	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)	
<b>Direction</b>	Direction of movement (cyclic axis or special case only)	long
	1 ..... Positive	
	2 ..... Shortest	
	3 ..... Negative	
	4 ..... Current	
<b>EndVelocity</b>	End velocity	double

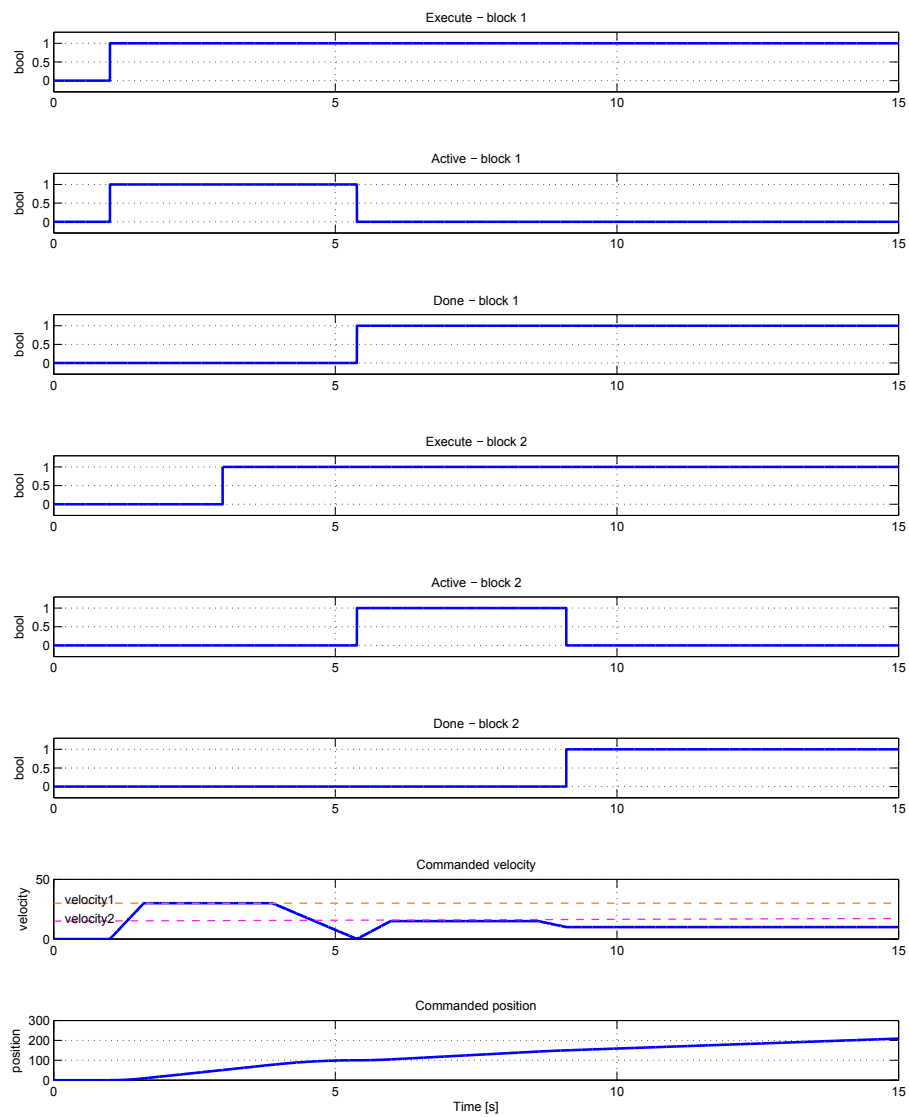
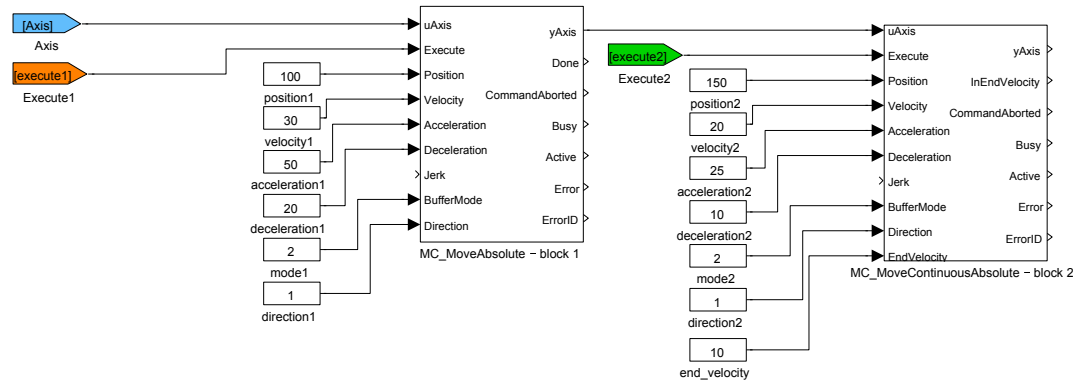
## Outputs

<b>yAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>InEndVelocity</b>	Algorithm finished	bool
<b>CommandAborted</b>	Algorithm was aborted	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Active</b>	The block is controlling the axis	bool
<b>Error</b>	Error occurred	bool

ErrorID	Result of the last operation
i .....	REXYGEN general error

error

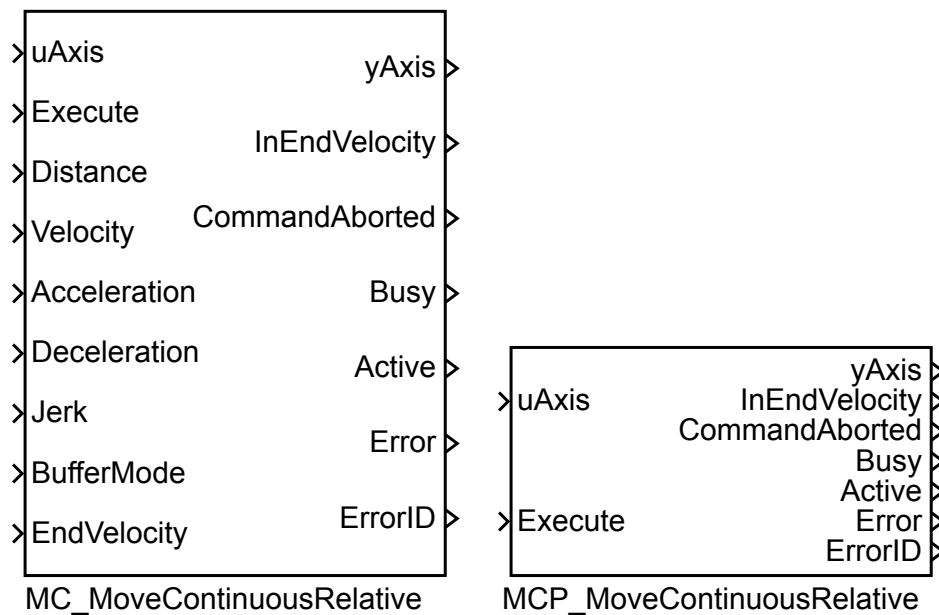
## Example



## 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

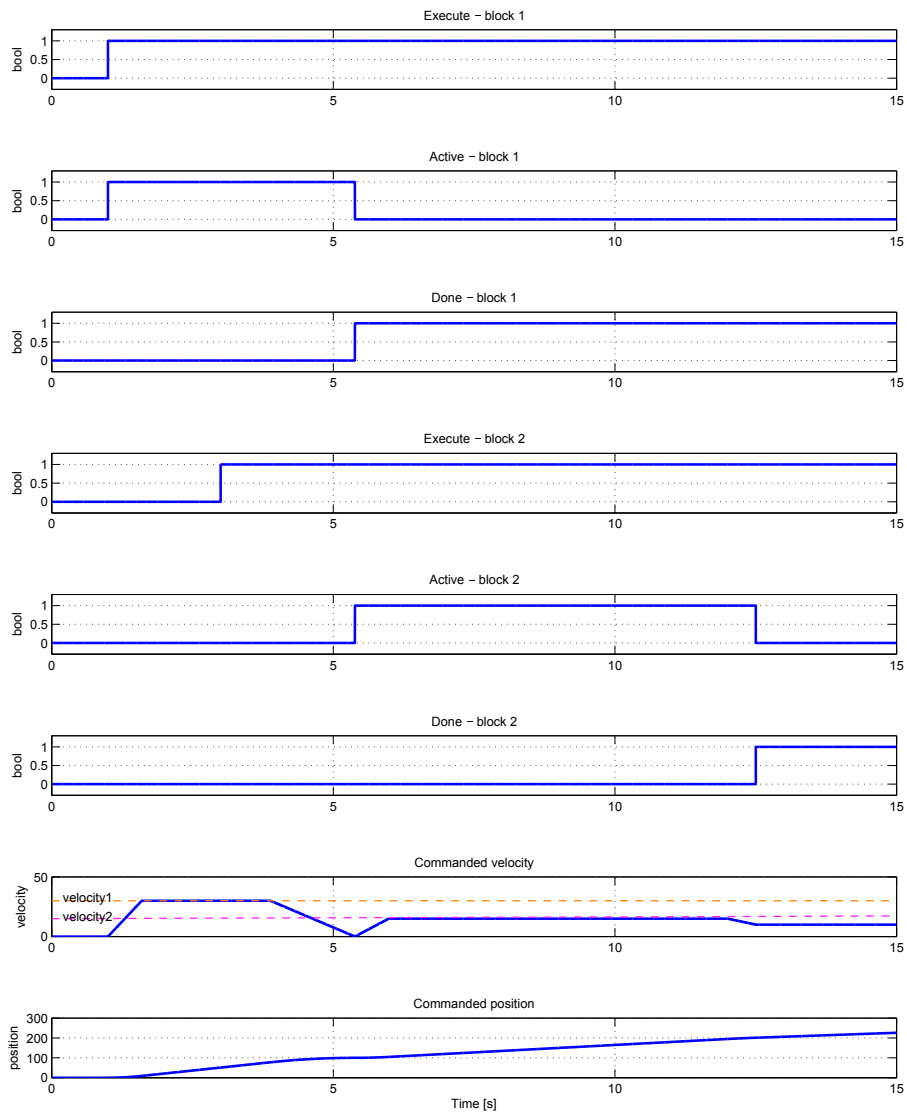
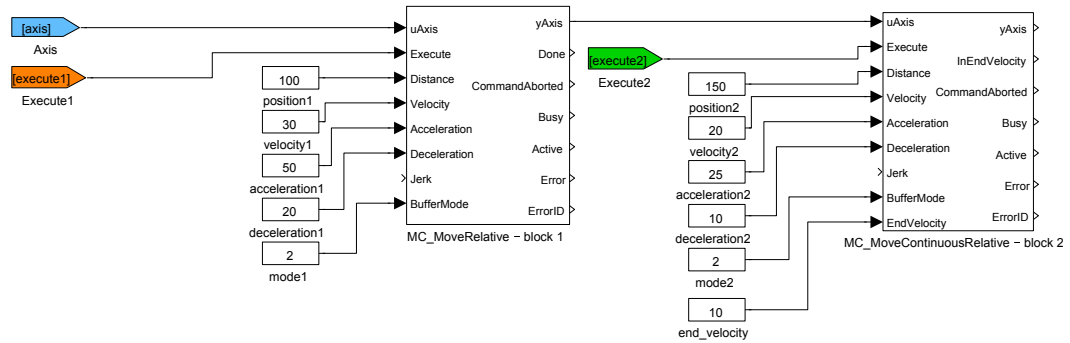
<b>uAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Execute</b>	The block is activated on rising edge	<b>bool</b>
<b>Distance</b>	Requested target distance (relative to execution point) [unit]	<b>double</b>
<b>Velocity</b>	Maximal allowed velocity [unit/s]	<b>double</b>
<b>Acceleration</b>	Maximal allowed acceleration [unit/s <sup>2</sup> ]	<b>double</b>
<b>Deceleration</b>	Maximal allowed deceleration [unit/s <sup>2</sup> ]	<b>double</b>
<b>Jerk</b>	Maximal allowed jerk [unit/s <sup>3</sup> ]	<b>double</b>

<b>BufferMode</b>	Buffering mode	long
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)	
<b>EndVelocity</b>	End velocity	long

## Outputs

<b>yAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> connections are allowed)	reference
<b>InEndVelocity</b>	PLCopen Done (algorithm finished)	bool
<b>CommandAborted</b>	PLCopen CommandAborted (algorithm was aborted)	bool
<b>Busy</b>	PLCopen Busy (algorithm not finished yet)	bool
<b>Active</b>	PLCopen Active (the block is controlling the axis)	bool
<b>Error</b>	PLCopen Error (error occurred)	bool
<b>ErrorID</b>	Result of the last operation	error
i	..... REXYGEN general error	

## Example

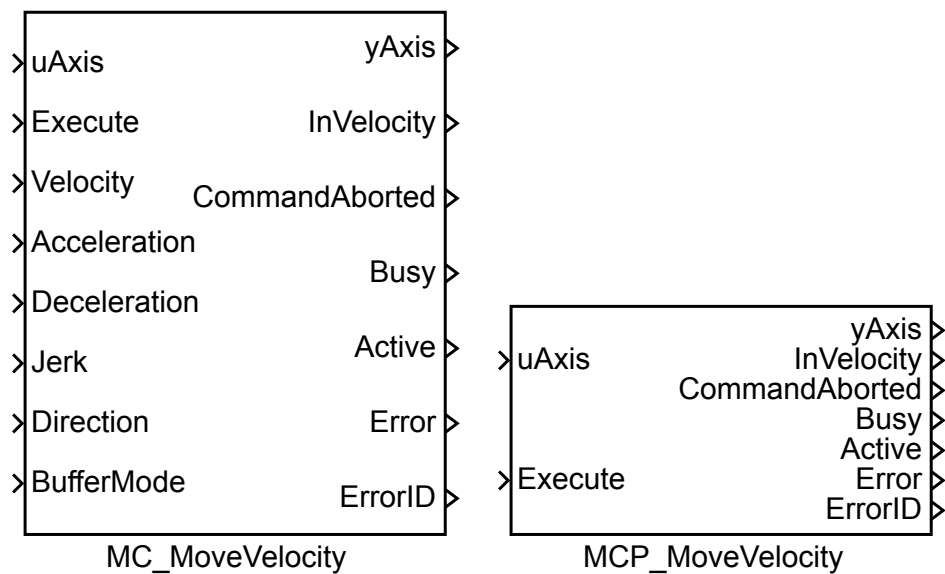




## MC\_MoveVelocity, MCP\_MoveVelocity – Move with constant velocity

### 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

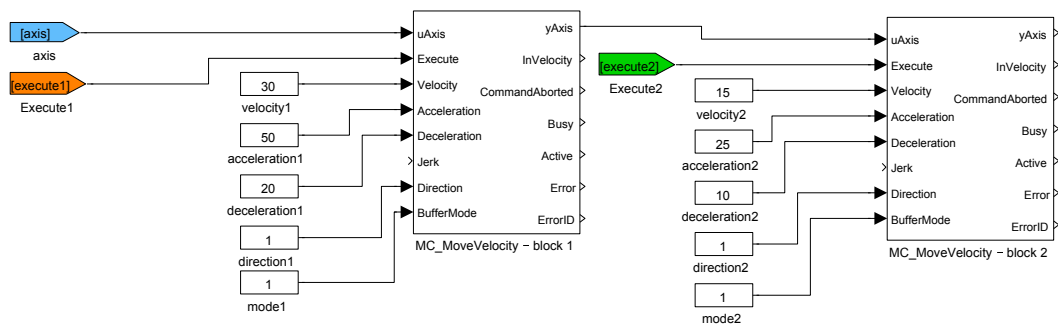
<b>uAxis</b>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> reference connections are allowed)	
<b>Execute</b>	The block is activated on rising edge	bool
<b>Velocity</b>	Maximal allowed velocity [unit/s]	double
<b>Acceleration</b>	Maximal allowed acceleration unit/s <sup>2</sup>	double
<b>Deceleration</b>	Maximal allowed deceleration [unit/s <sup>2</sup> ]	double
<b>Jerk</b>	Maximal allowed jerk [unit/s <sup>3</sup> ]	double

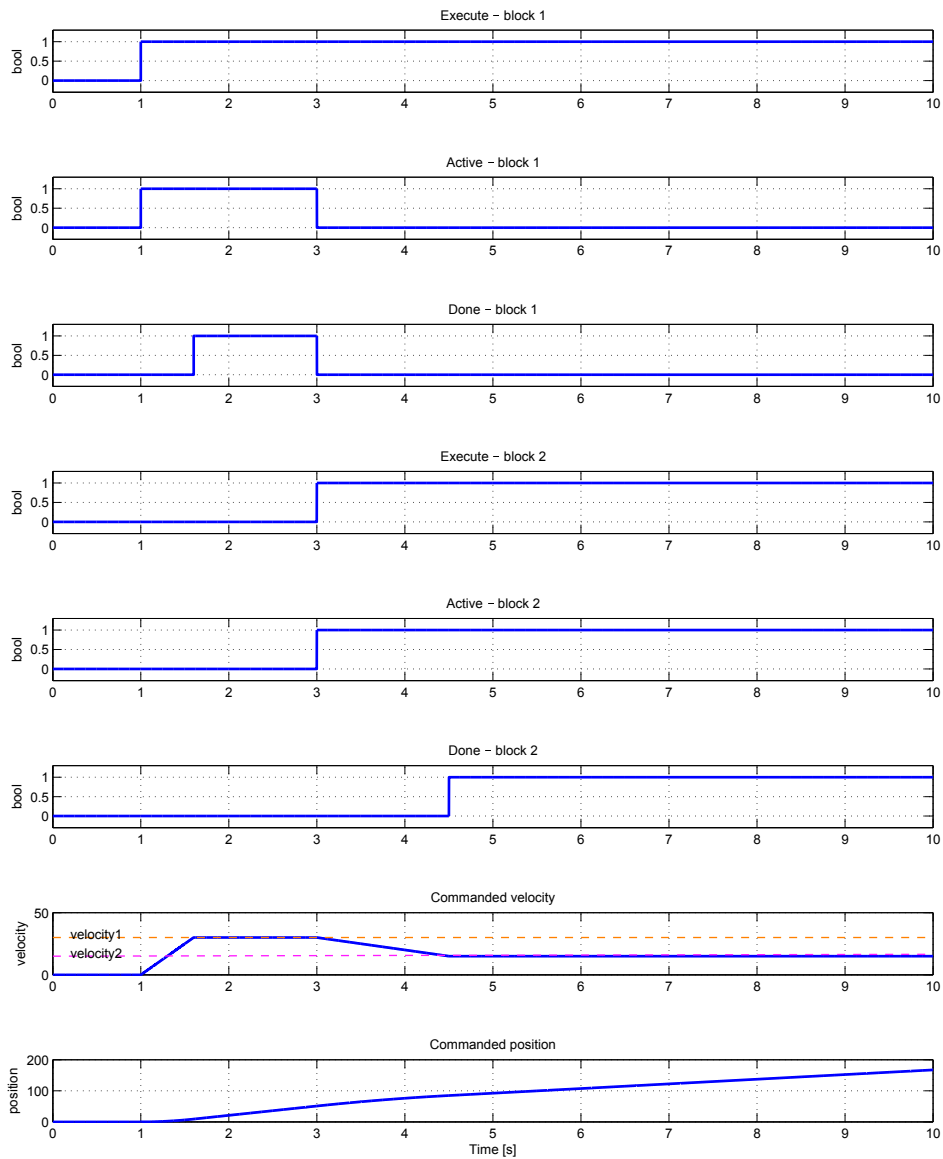
<b>Direction</b>	Direction of movement (cyclic axis or special case only)	<b>long</b>
	1 ..... Positive	
	2 ..... Shortest	
	3 ..... Negative	
	4 ..... Current	
<b>BufferMode</b>	Buffering mode	<b>long</b>
	1 ..... Aborting (start immediately)	
	2 ..... Buffered (start after finish of previous motion)	
	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

<b>yAxis</b>	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	<b>reference</b>
<b>InVelocity</b>	Requested velocity reached	<b>bool</b>
<b>CommandAborted</b>	Algorithm was aborted	<b>bool</b>
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Active</b>	The block is controlling the axis	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation	<b>error</b>
	i ..... REXYGEN general error	

## Example

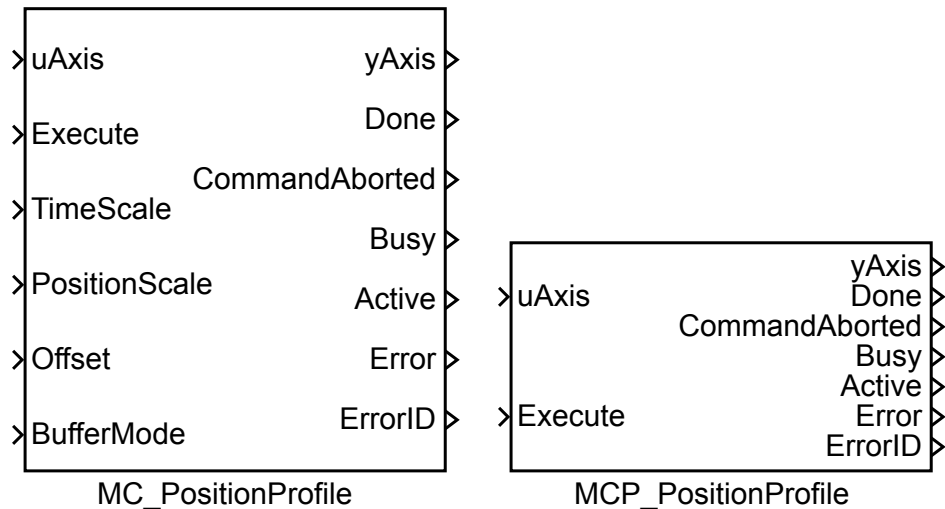




## MC\_PositionProfile, MCP\_PositionProfile – Position profile

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

<b>uAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Execute</b>	The block is activated on rising edge	bool
<b>TimeScale</b>	Overall scale factor in time	double
<b>PositionScale</b>	Overall scale factor in value	double
<b>Offset</b>	Overall profile offset in value	double
<b>BufferMode</b>	Buffering mode	long
	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

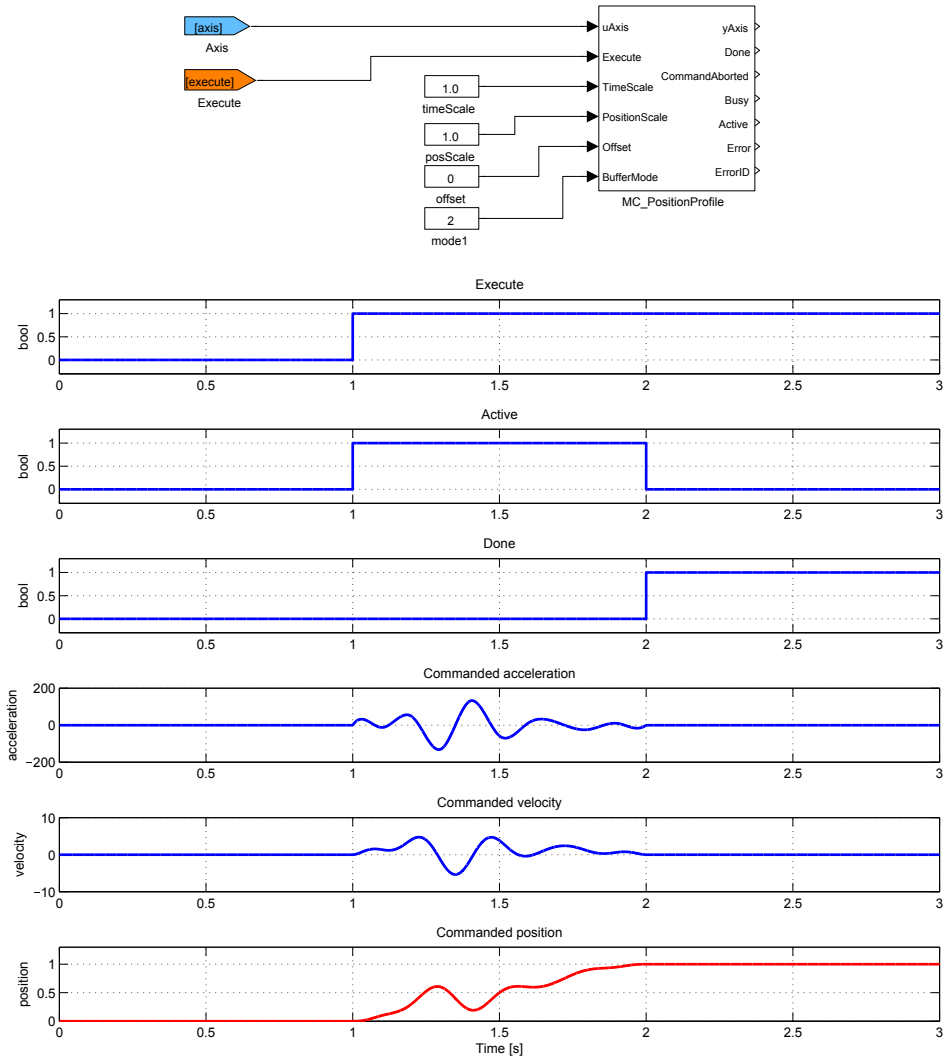
<b>yAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Done</b>	Algorithm finished	bool

<b>CommandAborted</b>	Algorithm was aborted	<b>bool</b>
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Active</b>	The block is controlling the axis	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation	<b>error</b>
	i ..... REXYGEN general error	

## Parameters

<b>alg</b>	Algorithm for interpolation	⊙2	<b>long</b>
	1 ..... Sequence of time/value pairs		
	2 ..... Sequence of equidistant values		
	3 ..... Spline		
	4 ..... Equidistant spline		
<b>cSeg</b>	Number of profile segments	⊙3	<b>long</b>
<b>times</b>	Times when segments are switched	⊙[0 30]	<b>double</b>
<b>values</b>	Values or interpolating polynomial coefficients (a0, a1, a2, ...)		<b>double</b>
		⊙[0 100 100 50]	

Example

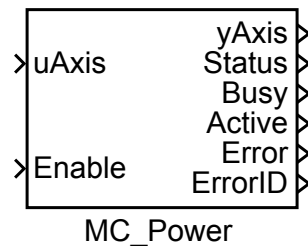




## MC\_Power – Axis activation (power on/off)

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 stoping sequence is activated and the status is switched to off/diabled when the motion stops (commanded velocity reaches zero value).

### Inputs

<b>uAxis</b>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> reference connections are allowed)	
<b>Enable</b>	Block function is enabled	bool

### Outputs

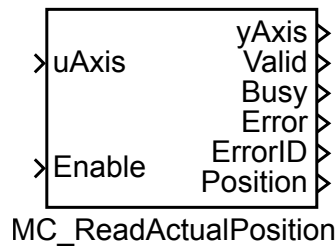
<b>yAxis</b>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> reference connections are allowed)	
<b>Status</b>	Effective state of the power stage	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Active</b>	The block is controlling the axis	bool
<b>Error</b>	Error occurred	bool

ErrorID	Result of the last operation	error
i . . . . .	REXYGEN general 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> reference connections are allowed)	
<code>Enable</code>	Block function is enabled	bool

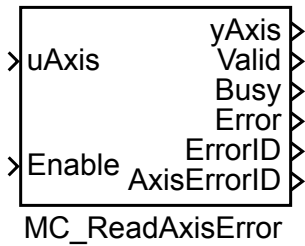
### Outputs

<code>yAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> reference connections are allowed)	
<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

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

uAxis	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
Enable	Block function is enabled	bool

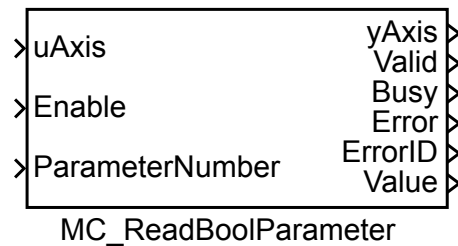
Outputs

yAxis	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
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
AxisErrorID	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

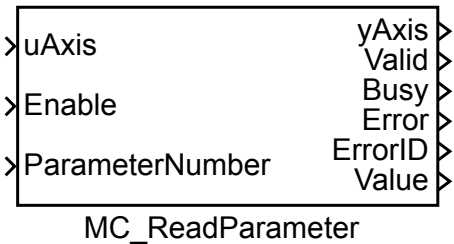
<b>uAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Enable</b>	Block function is enabled	bool
<b>ParameterNumber</b>	Parameter ID	long
	4 ..... Enable sw positive limit	
	5 ..... Enable sw negative limit	
	6 ..... Enable position lag monitoring	

### Outputs

<b>yAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Valid</b>	Output value is valid	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Error</b>	Error occurred	bool
<b>ErrorID</b>	Result of the last operation	error
	i ..... REXYGEN general error	
<b>Value</b>	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

<b>uAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Enable</b>	Block function is enabled	bool
<b>ParameterNumber</b>	Parameter ID	long
	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	

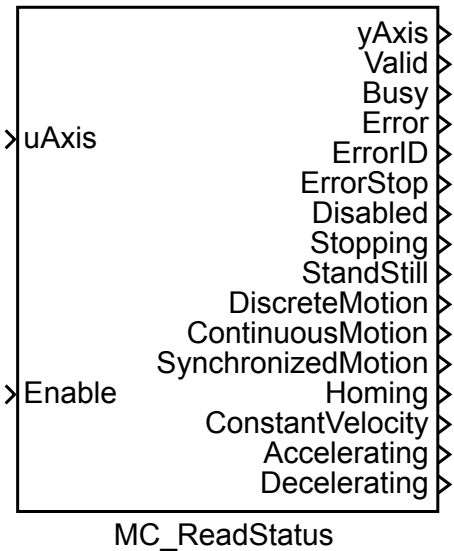
## Outputs

<code>yAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> reference connections are allowed)	
<code>Valid</code>	Output value is valid	<code>bool</code>
<code>Busy</code>	Algorithm not finished yet	<code>bool</code>
<code>Error</code>	Error occurred	<code>bool</code>
<code>ErrorID</code>	Result of the last operation	<code>error</code>
	<code>i . . . . .</code> REXYGEN general error	
<code>Value</code>	Parameter value	<code>double</code>

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

<b>uAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Enable</b>	Block function is enabled	bool

Outputs

<b>yAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Valid</b>	Output value is valid	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Error</b>	Error occurred	bool
<b>ErrorID</b>	Result of the last operation i . . . . . REXYGEN general error	error
<b>ErrorStop</b>	Axis is in the ErrorStop state	bool

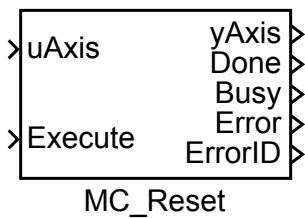


<b>Disabled</b>	Axis is in the Disabled state	bool
<b>Stopping</b>	Axis is in the Stopping state	bool
<b>StandStill</b>	Axis is in the StandStill state	bool
<b>DiscreteMotion</b>	Axis is in the DiscreteMotion state	bool
<b>ContinuousMotion</b>	Axis is in the ContinuousMotion state	bool
<b>SynchronizedMotion</b>	Axis is in the SynchronizedMotion state	bool
<b>Homing</b>	Axis is in the Homing state	bool
<b>ConstantVelocity</b>	Axis is moving with constant velocity	bool
<b>Accelerating</b>	Axis is accelerating	bool
<b>Decelerating</b>	Axis is decelerating	bool

MC\_Reset – **Reset axis errors**

Block Symbol

Licence: [MOTION CONTROL](#)



Function Description

The **MC\_Reset** block makes the transition from the state **ErrorStop** to **StandStill** by resetting all internal axis-related errors.

Inputs

<b>uAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Execute</b>	The block is activated on rising edge	<b>bool</b>

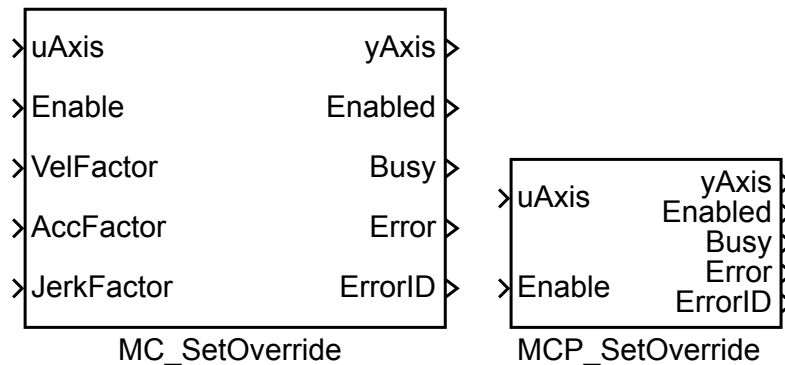
Outputs

<b>yAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Done</b>	Algorithm finished	<b>bool</b>
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation	<b>error</b>
	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** reference connections are allowed)

<b>Enable</b>	Block function is enabled	<b>bool</b>
<b>VelFactor</b>	Velocity multiplication factor	<b>double</b>
<b>AccFactor</b>	Acceleration/deceleration multiplication factor	<b>double</b>
<b>JerkFactor</b>	Jerk multiplication factor	<b>double</b>

## Outputs

<b>yAxis</b>	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	<b>reference</b>
<b>Enabled</b>	Block function is enabled	<b>bool</b>
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation i . . . . . REXYGEN general error	<b>error</b>

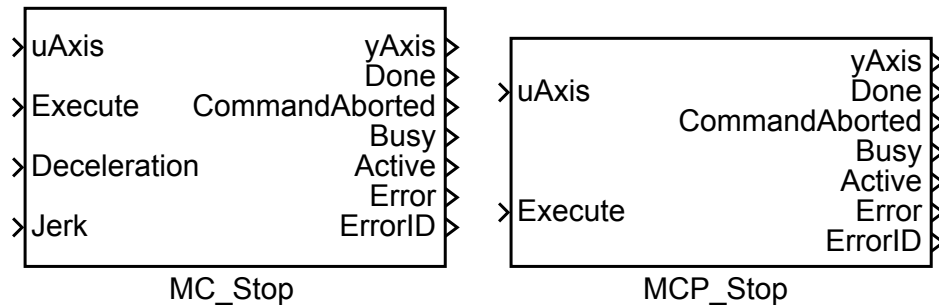
## Parameter

<b>diff</b>	Deadband (difference for recalculation)	↓0.0 ↑1.0 ⊙0.1	<b>double</b>
-------------	---	----------------	---------------

## 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 <a href="#">RM_Axis.axisRef-uAxis</a> or <a href="#">yAxis-uAxis</a> reference connections are allowed)	
Execute	The block is activated on rising edge	bool
Deceleration	Maximal allowed deceleration [unit/s <sup>2</sup> ]	double
Jerk	Maximal allowed jerk [unit/s <sup>3</sup> ]	double

### Outputs

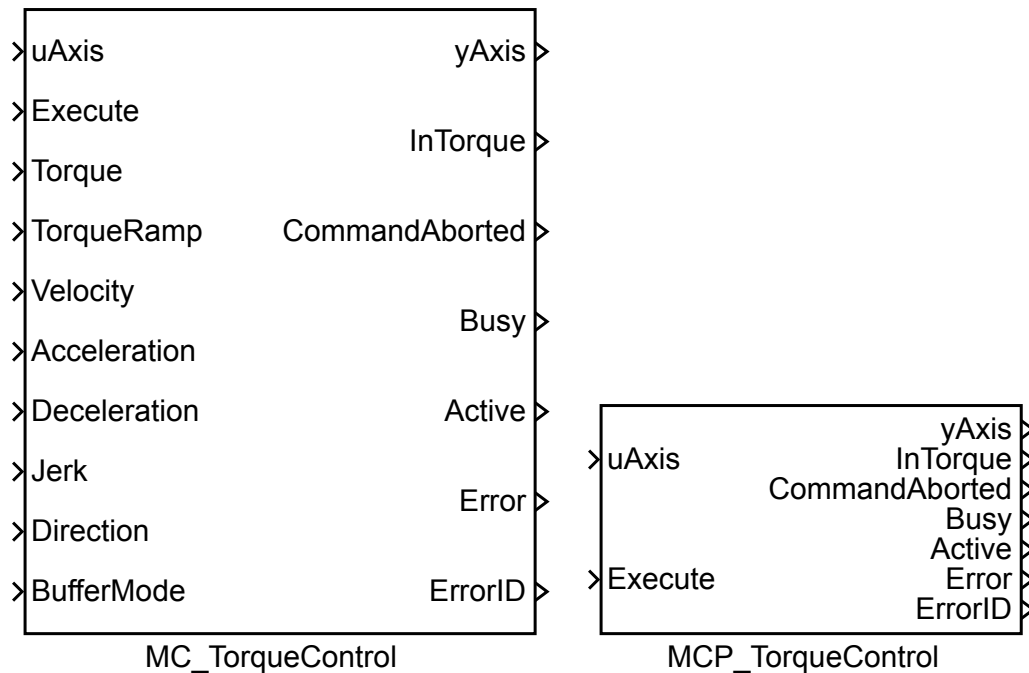
yAxis	Axis reference (only <a href="#">RM_Axis.axisRef-uAxis</a> or <a href="#">yAxis-uAxis</a> reference connections are allowed)	
-------	--	--

<b>Done</b>	Algorithm finished	<b>bool</b>
<b>CommandAborted</b>	Algorithm was aborted	<b>bool</b>
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Active</b>	The block is controlling the axis	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation	<b>error</b>
	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

<code>uAxis</code>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> reference connections are allowed)	
<code>Execute</code>	The block is activated on rising edge	<code>bool</code>
<code>Torque</code>	Maximal allowed torque/force	<code>double</code>
<code>TorqueRamp</code>	Maximal allowed torque/force ramp	<code>double</code>

Velocity	Maximal allowed velocity [unit/s]	double
Acceleration	Maximal allowed acceleration [unit/s <sup>2</sup> ]	double
Deceleration	Maximal allowed deceleration [uunit/s <sup>2</sup> ]	double
Jerk	Maximal allowed jerk [unit/s <sup>3</sup> ]	double
Direction	Direction of movement (cyclic axis or special case only)	long
	1 ..... Positive	
	2 ..... Shortest	
	3 ..... Negative	
	4 ..... Current	
BufferMode	Buffering mode	long
	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

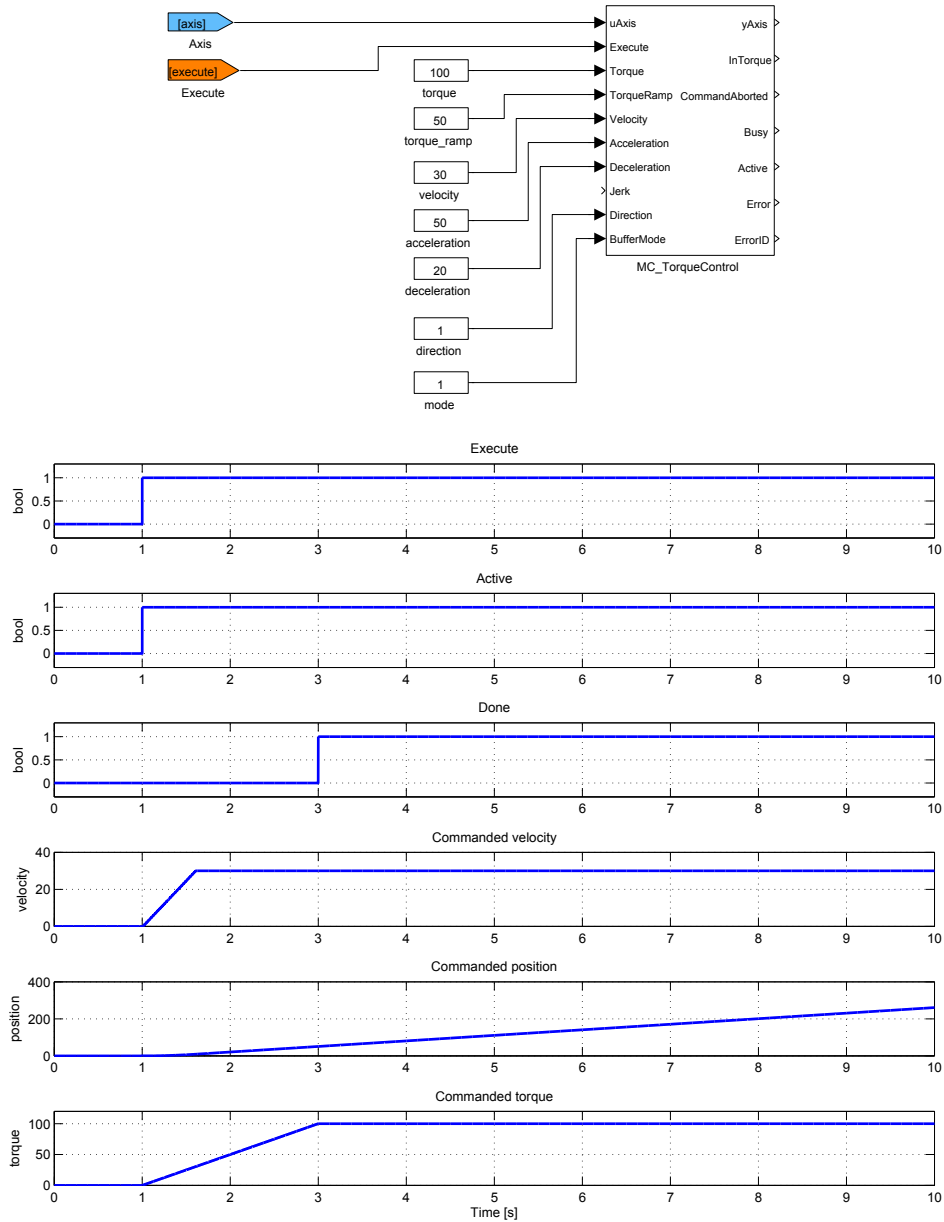
yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	reference
InTorque	Requested torque/force is reached	bool
CommandAborted	Algorithm was aborted	bool
Busy	Algorithm not finished yet	bool
Active	The block is controlling the axis	bool
Error	Error occurred	bool
ErrorID	Result of the last operation	error
	i ..... REXYGEN general error	

## Parameter

kma	Torque/force to acceleration ratio	double
-----	------------------------------------	--------

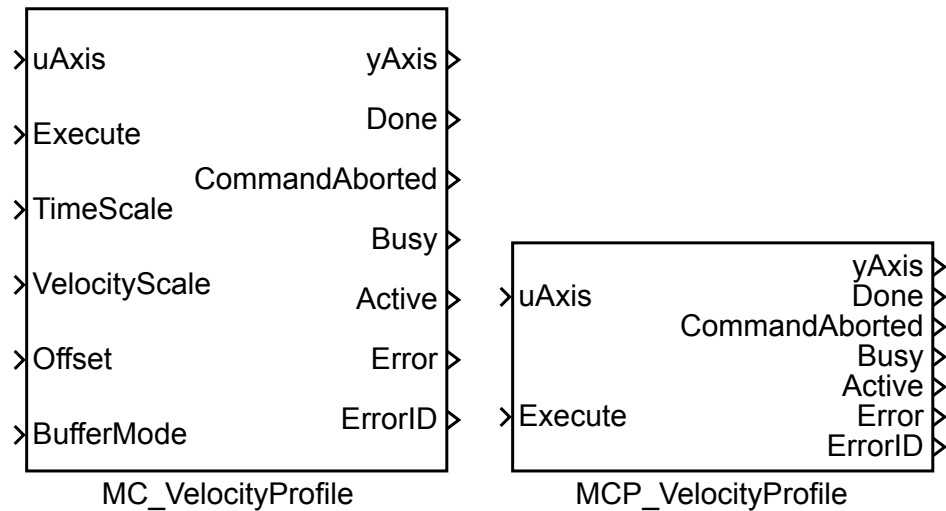


## Example



## 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

<b>uAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Execute</b>	The block is activated on rising edge	bool
<b>TimeScale</b>	Overall scale factor in time	double
<b>VelocityScale</b>	Overall scale factor in value	double
<b>Offset</b>	Overall profile offset in value	double
<b>BufferMode</b>	Buffering mode	long
	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

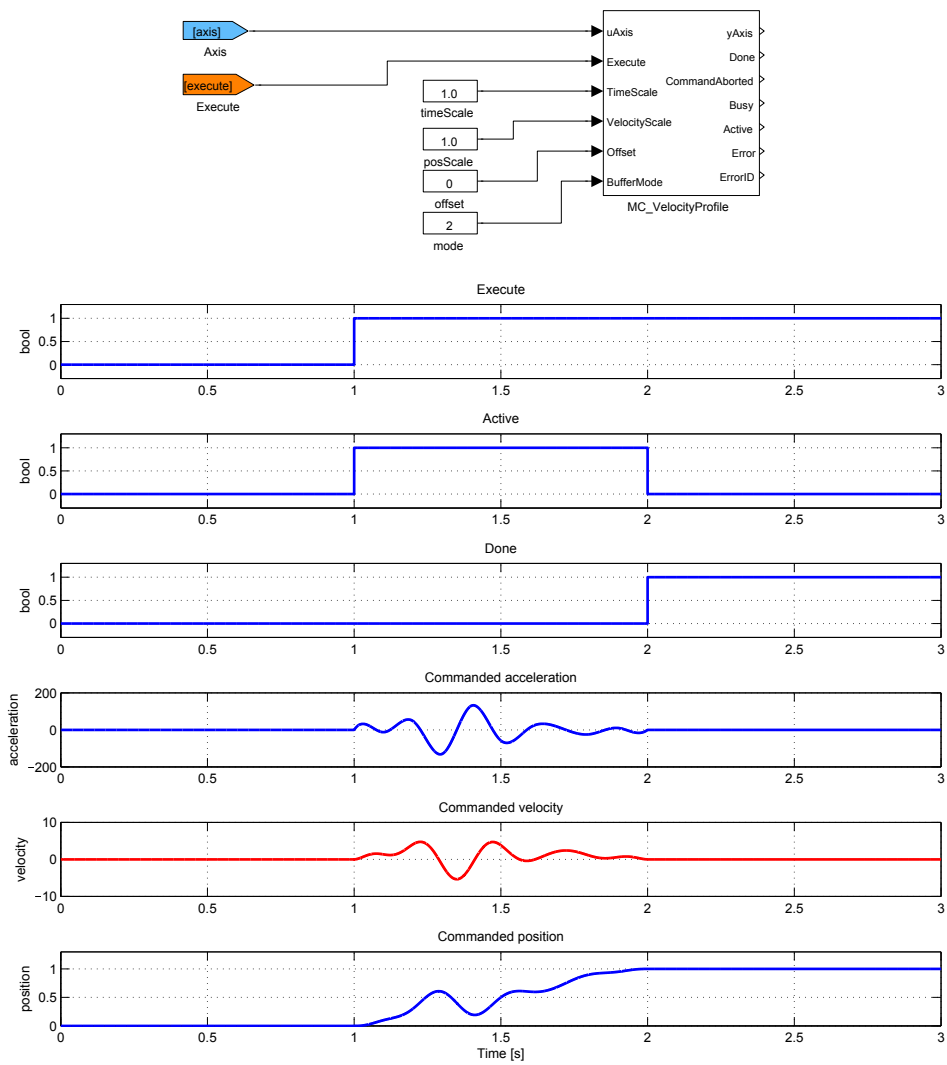
<b>yAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Done</b>	Algorithm finished	bool
<b>CommandAborted</b>	Algorithm was aborted	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Active</b>	The block is controlling the axis	bool
<b>Error</b>	Error occurred	bool

ErrorID	Result of the last operation	error
i	..... REXYGEN general error	

Parameters

alg	Algorithm for interpolation	⊙1	long
	1 ..... Sequence of time/value pairs		
	2 ..... Sequence of equidistant values		
	3 ..... Spline		
	4 ..... Equidistant spline		
cSeg	Number of profile segments	⊙3	long
times	Times when segments are switched	⊙[0 15 25 30]	double
values	Values or interpolating polynomial coefficients (a0, a1, a2, ...)		double
		⊙[0 100 100 50]	

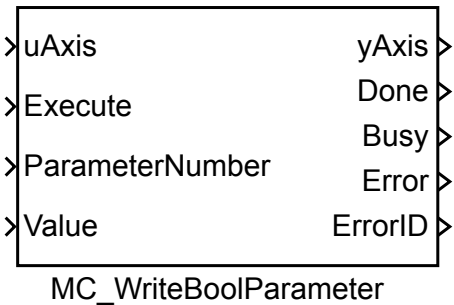
## Example



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 reference connections are allowed)	
Execute	The block is activated on rising edge	bool
ParameterNumber	Parameter ID	long
	4 ..... Enable sw positive limit	
	5 ..... Enable sw negative limit	
	6 ..... Enable position lag monitoring	
Value	Parameter value	bool

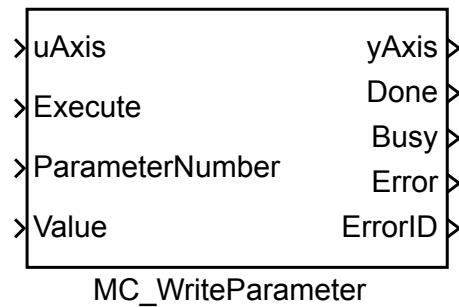
Outputs

yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)	
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

<b>uAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Execute</b>	The block is activated on rising edge	bool
<b>ParameterNumber</b>	Parameter ID	long
	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	
<b>Value</b>	Parameter value	double

### Outputs

<b>yAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> reference connections are allowed)	
<b>Done</b>	Algorithm finished	bool

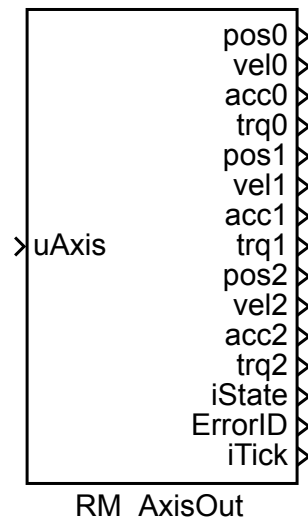
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation	<b>error</b>
	i . . . . . REXYGEN general error	



## 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 1: Control system **REXYGEN** orders blocks primary by flow of signal, secondarily by name of block (ascendent in alphabetical order), so name like "zzz" is good choice. For checking the order, you can use **REXYGEN** Diagnostics tool where the blocks are sorted by execution order.

Note 2: almost all blocks do not work with torque so commanded torque is 0. Commanded acceleration and torque should be used as feed-forward value for position/velocity controller so this value does not make any problem.

## Inputs

<b>uAxis</b>	axis reference that must be connected to <b>axisRef</b> of the <b>RM_Axis</b> block (direct or indirect throw output <b>yAxis</b> of some other block)	<b>reference</b>
--------------	---	------------------

## Input

`uAxis` Axis reference (only `RM_Axis.axisRef-uAxis` or `yAxis-uAxis` reference connections are allowed)

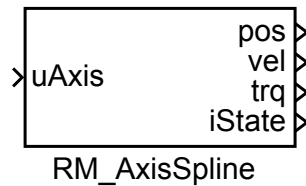
## Outputs

<code>pos0</code>	Current commanded position [unit]	double
<code>vel0</code>	Current commanded velocity [unit/s]	double
<code>acc0</code>	Current commanded acceleration [unit/s <sup>2</sup> ]	double
<code>trq0</code>	Current commanded torque/force (if generated)	double
<code>pos1</code>	Next step commanded position [unit]	double
<code>vel1</code>	Next step commanded velocity [unit/s]	double
<code>acc1</code>	Next step commanded acceleration/deceleration [unit/s <sup>2</sup> ]	double
<code>trq1</code>	Next step commanded torque/force (if generated)	double
<code>pos2</code>	2nd next step commanded position [unit]	double
<code>vel2</code>	2nd next step commanded velocity [unit/s]	double
<code>acc2</code>	2nd next step commanded acceleration/deceleration [unit/s <sup>2</sup> ]	double
<code>trq2</code>	2nd next step commanded torque/force (if generated)	double
<code>iState</code>	State of the axis	long
	0 ..... Disabled	
	1 ..... Stand still	
	2 ..... Homing	
	3 ..... Discrete motion	
	4 ..... Continuous motion	
	5 ..... Synchronized motion	
	6 ..... Coordinated motion	
	7 ..... Stopping	
	8 ..... Error stop	
<code>ErrorID</code>	Result of the last operation	error
	i ..... REXYGEN general error	
<code>iTick</code>	Current tick	long

## RM\_AxisSpline – Commanded values interpolation

Block Symbol

Licence: [MOTION CONTROL](#)



### Function Description

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 200ms). On the other side, the motor driver usually requires small sampling period for smooth/waveless movement. The **RM\_AxisSpline** block solves this problem of multirate execution of motion planning and motion control levels. The block can run in different task than other motion control blocks with highest sampling period possible. It interpolates commanded position, velocity and torque and generates smooth curve which is more suited for motor driver controllers.

There are two possibilities of connection to **RM\_Axis** block: connect all necessary values (outputs of the block **RM\_AxisOut**) as input of interpolating block or use only axis reference and read the state directly. This block uses axis reference. For correct synchronization between two tasks, the block **RM\_Axis** must be executed first followed by all axis related motion control blocks and finally by block **RM\_AxisOut** at the end.

Note 1: For interpolation of position signal, 3rd order polynomial  $p(t)$  is used, where  $p_s(0) = pos0, p_s(t_s) = pos1, \frac{dp_s(t)}{dt}_{t=0} = vel0, \frac{dp_s(t)}{dt}_{t=t_s} = vel1$ . To interpolate velocity, also an 3rd order polynomial  $p_v(t)$  is used, where  $p_v(0) = vel0, p_v(t_s) = vel1, \frac{dp_v(t)}{dt}_{t=0} = acc0, \frac{dp_v(t)}{dt}_{t=t_s} = acc1$ . Torque is interpolated by linear function.

Note 2: Because the time of execution of motion blocks is varying in time, the block uses one or two step prediction for interpolation depending on actual conditions and timing of the motion blocks in slower tasks. The use of predicted values is signalized by states **RUN0**, **RUN1**, **RUN2**.

Note 3: Control system **REXYGEN** orders the blocks primarily by flow of signal, secondarily by name of block (ascendent in alphabetical order), so name like "zzz" is good choice for the block **RM\_AxisOut**. For checking the order, you can use **REXYGEN Diagnostics** tool where the blocks are sorted by execution order.

## Input

`uAxis`      Axis reference (only `RM_Axis.axisRef-uAxis` or `yAxis-uAxis` reference connections are allowed)

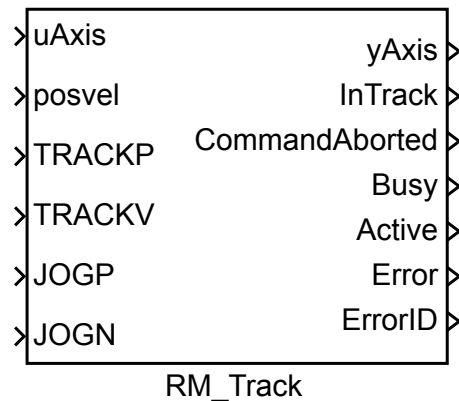
## Outputs

<code>pos</code>	Commanded interpolated position [unit]	double
<code>vel</code>	Commanded interpolated velocity [unit/s]	double
<code>trq</code>	Commanded interpolated torque/force	double
<code>iState</code>	Interpolator state/error	long
	0 ..... Off	
	1 ..... Run0	
	2 ..... Run1	
	3 ..... Run2	
	5 ..... Change1	
	-1 .... Change0	
	-2 .... Late	
	-3 .... Busy	
	-4 .... Slow	

## 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

<b>uAxis</b>	Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> reference connections are allowed)	
<b>posvel</b>	Requested target position or velocity [unit]	double

TRACKP	Position tracking mode	bool
TRACKV	Velocity tracking mode	bool
JOGP	Moving positive direction mode	bool
JOGN	Moving negative direction mode	bool

## Parameters

pv	Maximal allowed velocity [unit/s]	double
pa	Maximal allowed acceleration [unit/s <sup>2</sup> ]	double
pd	Maximal allowed deceleration [unit/s <sup>2</sup> ]	double
pj	Maximal allowed jerk [unit/s <sup>3</sup> ]	double
iLen	Length of buffer for estimation	⊙10 long

## Outputs

yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	reference
InTrack	Requested position 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	

## Chapter 17

# MC\_MULTI – Motion control - multi axis blocks

### Contents

---

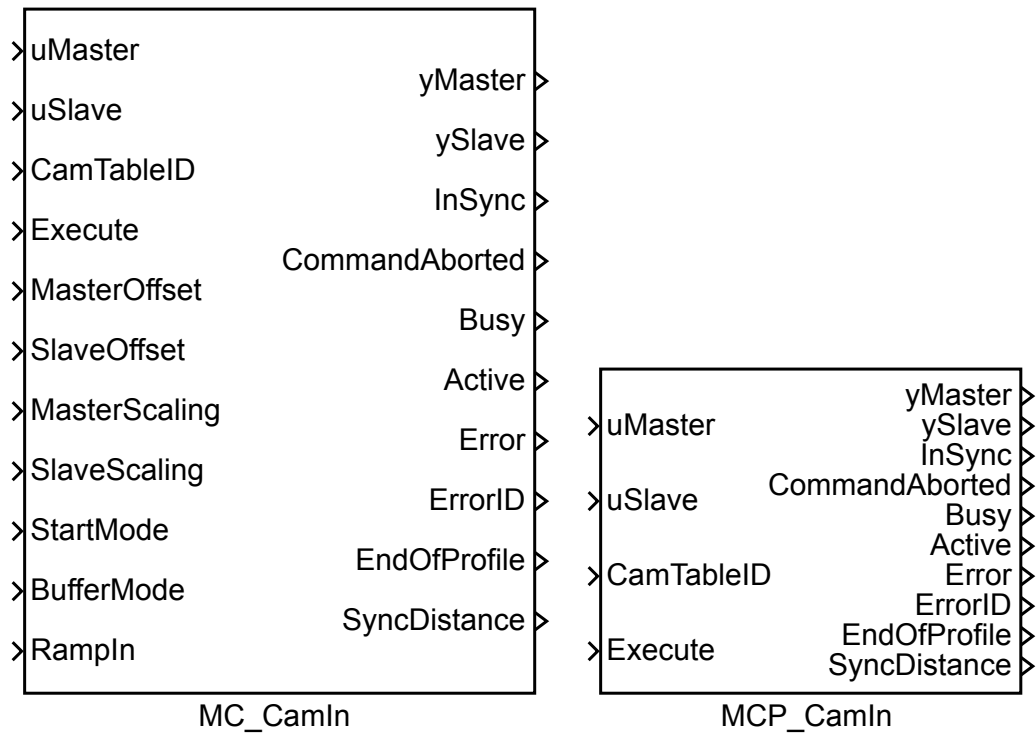
MC_CamIn, MCP_CamIn – Engage the cam . . . . .	556
MC_CamOut – Disengage the cam . . . . .	560
MCP_CamTableSelect – Cam definition . . . . .	562
MC_CombineAxes, MCP_CombineAxes – Combine the motion of 2 axes into a third axis . . . . .	564
MC_GearIn, MCP_GearIn – Engage the master/slave velocity ratio . . . . .	568
MC_GearInPos, MCP_GearInPos – Engage the master/slave velocity ratio in defined position . . . . .	571
MC_GearOut – Disengage the master/slave velocity ratio . . . . .	577
MC_PhasingAbsolute, MCP_PhasingAbsolute – Phase shift in synchronized motion (absolute coordinates) . . . . .	579
MC_PhasingRelative, MCP_PhasingRelative – Phase shift in synchronized motion (relative coordinates) . . . . .	582

---

This block set is the second part of motion control blocks library according to the PLCopen standard for multi axis control. General vendor specific rules are the same as described in chapter 16 (the MC\_SINGLE library, blocks for single axis motion control).

## MC\_CamIn, MCP\_CamIn – Engage the cam

Block Symbols

Licence: [MOTION CONTROL](#)

## Function Description

*The MC\_CamIn and MCP\_CamIn blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP\_ version of the block.*

The MC\_CamIn block switches on a mode in which the slave axis is commanded to position which corresponds to the position of master axis transformed with 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

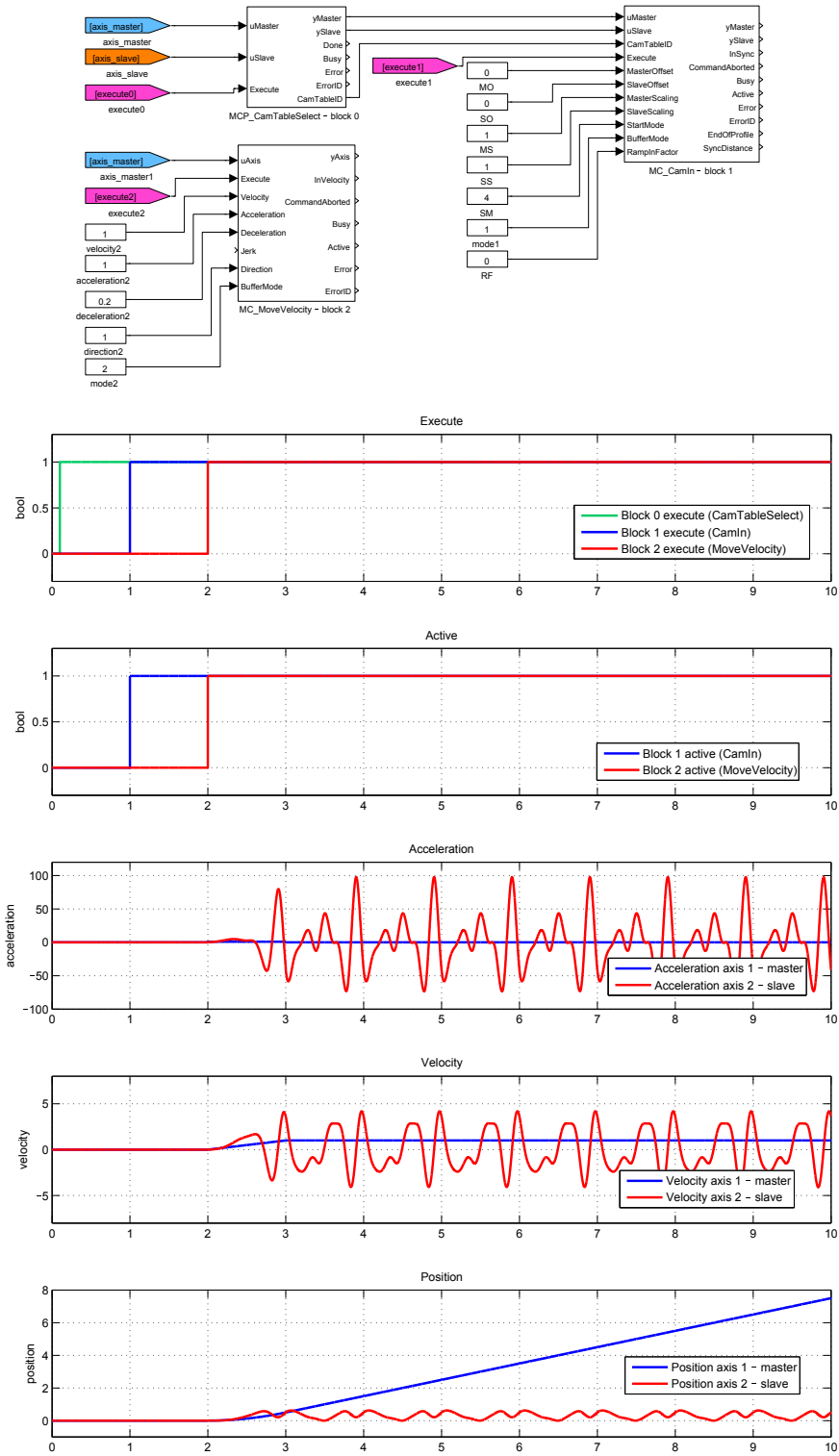
<b>uMaster</b>	Master axis reference	reference
<b>uSlave</b>	Slave axis reference	reference
<b>CamTableID</b>	Cam table reference (connect to <a href="#">MCP_CamTableSelect.CamTableID</a> )	reference
<b>Execute</b>	The block is activated on rising edge	bool
<b>MasterOffset</b>	Offset in cam table on master side [unit]	double
<b>SlaveOffset</b>	Offset in cam table on slave side [unit]	double
<b>MasterScaling</b>	Overall scaling factor in cam table on master side	double
<b>SlaveScaling</b>	Overall scaling factor in cam table on slave side	double
<b>StartMode</b>	Select relative or absolute cam table	long
	1 ..... Master relative	
	2 ..... Slave relative	
	3 ..... Both relative	
	4 ..... Both absolute	
<b>BufferMode</b>	Buffering mode	long
	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)	
<b>RampIn</b>	RampIn factor (0 = RampIn mode not used); average additive velocity (absolute value) during ramp-in process	double

## Outputs

<b>yMaster</b>	Master axis reference	reference
<b>ySlave</b>	Slave axis reference	reference
<b>InSync</b>	Slave axis reached the cam profile	bool

<b>CommandAborted</b>	Algorithm was aborted	<b>bool</b>
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Active</b>	The block is controlling the axis	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation	<b>error</b>
	i . . . . . REXYGEN general error	
<b>EndOfProfile</b>	Indicate end of cam profile ( not periodic cam only)	<b>bool</b>
<b>SyncDistance</b>	Position deviation of the slave axis from synchronized position	<b>double</b>

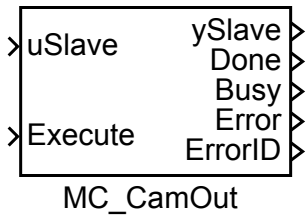
## Example



**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).

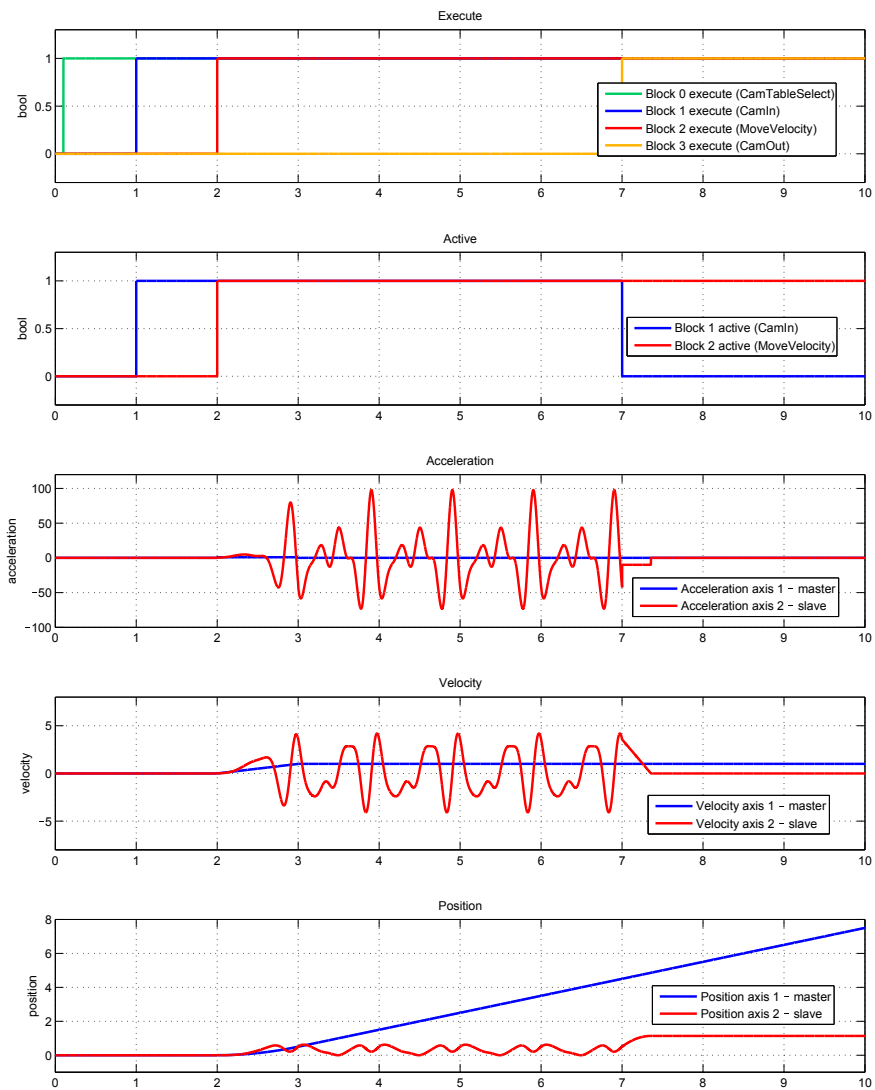
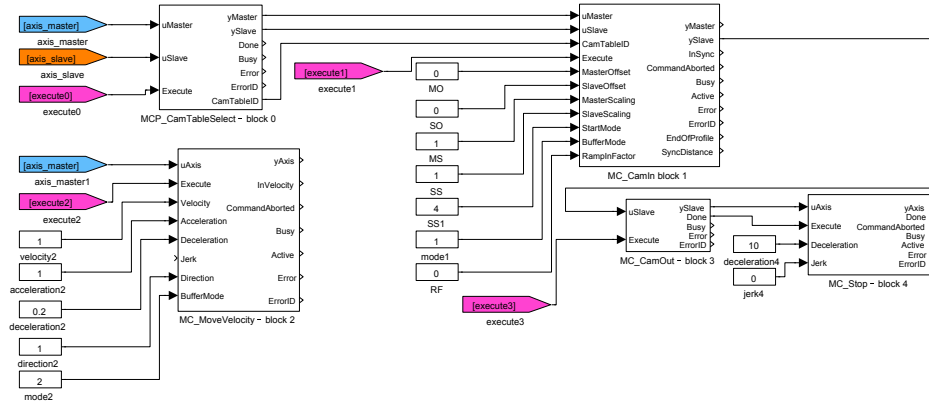
**Inputs**

<b>uSlave</b>	Slave axis reference	<b>reference</b>
<b>Execute</b>	The block is activated on rising edge	<b>bool</b>

**Outputs**

<b>ySlave</b>	Slave axis reference	<b>reference</b>
<b>Done</b>	Algorithm finished	<b>bool</b>
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation	<b>error</b>
	i   ..... REXYGEN general error	

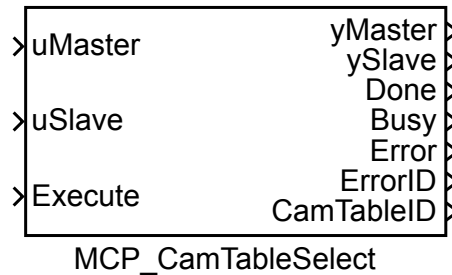
## Example



## 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

<b>uMaster</b>	Master axis reference	reference
<b>uSlave</b>	Slave axis reference	reference
<b>Execute</b>	The block is activated on rising edge	bool

## Outputs

<b>yMaster</b>	Master axis reference	reference
<b>ySlave</b>	Slave axis reference	reference
<b>Done</b>	Algorithm finished	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Error</b>	Error occurred	bool
<b>ErrorID</b>	Result of the last operation	error
	i ..... REXYGEN general error	
<b>CamTableID</b>	Cam table reference (connect to <a href="#">MC_CamIn.CamTableID</a> )	reference

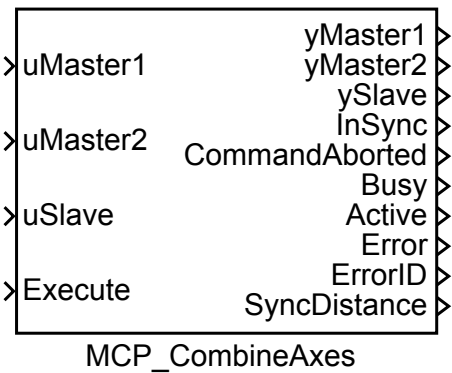
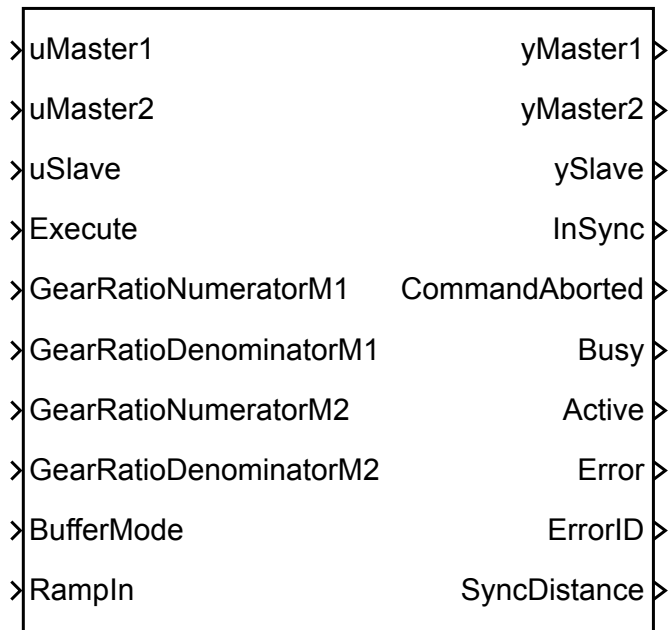
## Parameters

<b>alg</b>	Algorithm for interpolation	⊙2	long
	1 ..... Sequence of time/value pairs		
	2 ..... Sequence of equidistant values		
	3 ..... Spline		
	4 ..... Equidistant spline		
<b>cSeg</b>	Number of profile segments	⊙3	long
<b>Periodic</b>	Indicate periodic cam profile	⊙on	bool
<b>camname</b>	Filename of special editor data file (filename is generated by system if parameter is empty)		string
<b>mvalues</b>	Master positions where segments are switched	⊙[0 30]	double
<b>svalues</b>	Slave positions or interpolating polynomial coefficients (a0, a1, a2, ...)	⊙[0 100 100 0]	double

**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

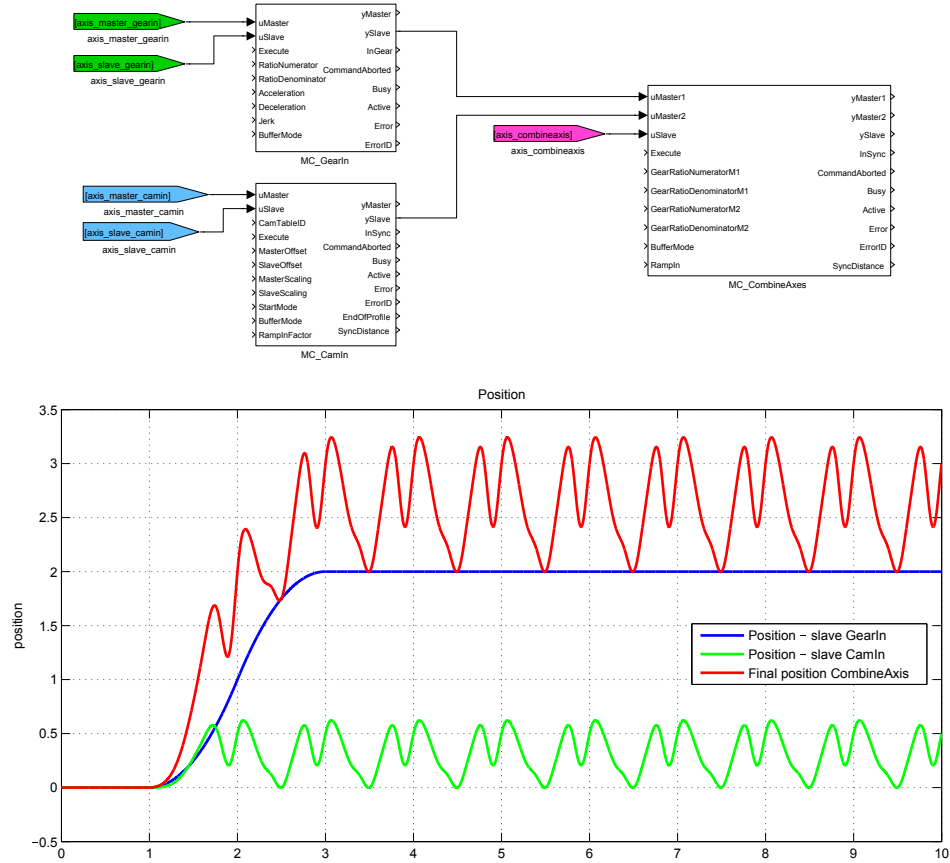
<b>uMaster1</b>	First master axis reference	reference
<b>uMaster2</b>	Second master axis reference	reference
<b>uSlave</b>	Slave axis reference	reference
<b>Execute</b>	The block is activated on rising edge	bool
<b>GearRatioNumeratorM1</b>	Numerator for the gear factor for master axis 1	long
<b>GearRatioDenominatorM1</b>	Denominator for the gear factor for master axis 1	long
<b>GearRatioNumeratorM2</b>	Numerator for the gear factor for master axis 2	long
<b>GearRatioDenominatorM2</b>	Denominator for the gear factor for master axis 2	long
<b>BufferMode</b>	Buffering mode	long
	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)	
<b>RampIn</b>	RampIn factor (0 = RampIn mode not used)	double

## Outputs

<b>yMaster1</b>	First master axis reference	reference
<b>yMaster2</b>	Second master axis reference	reference
<b>ySlave</b>	Slave axis reference	reference
<b>InSync</b>	Slave axis reached the cam profile	bool
<b>CommandAborted</b>	Algorithm was aborted	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Active</b>	The block is controlling the axis	bool
<b>Error</b>	Error occurred	bool

<b>ErrorID</b>	Result of the last operation	<b>error</b>
<b>i</b> .....	REXYGEN general error	
<b>SyncDistance</b>	Position deviation of the slave axis from synchronized position	<b>double</b>

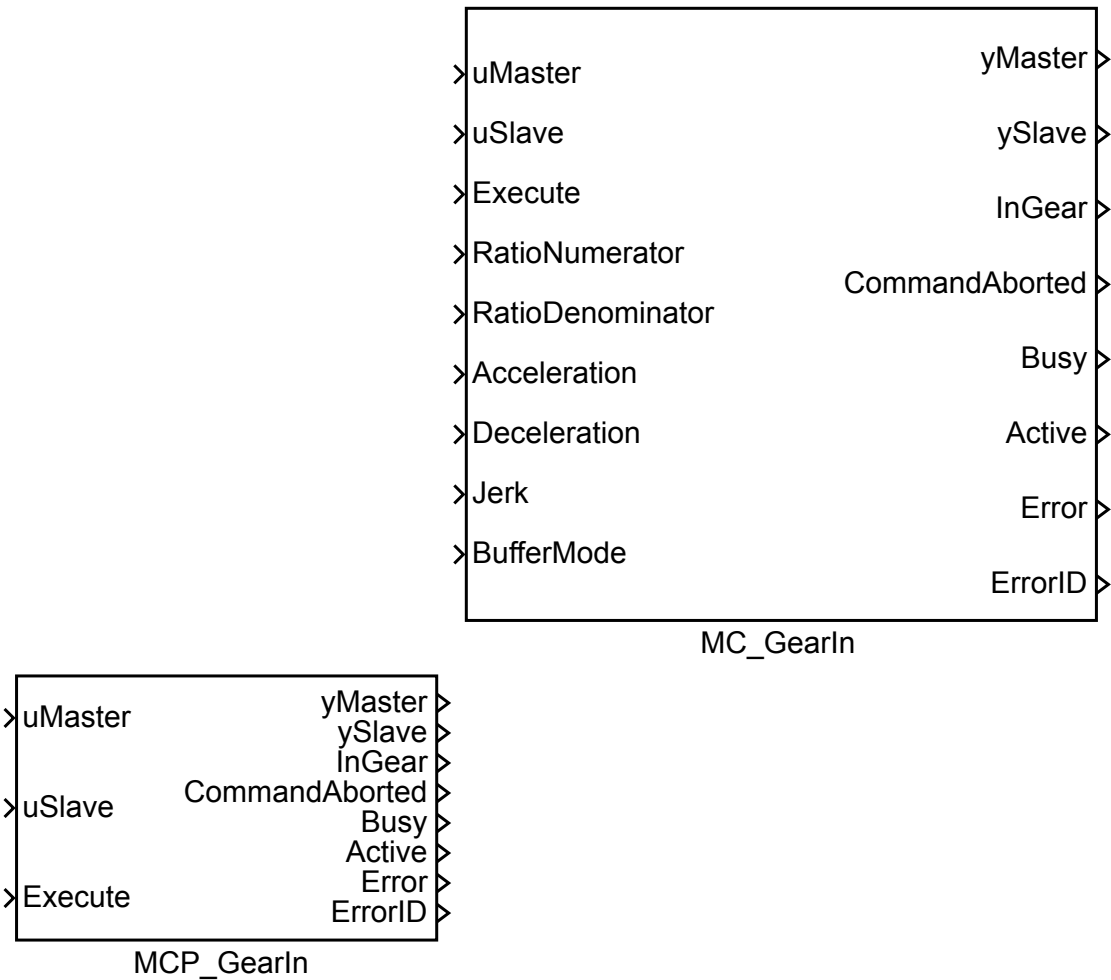
## Example



# 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 phas-

ing):  $VelS = VelM * RatioNumerator / RatioDenominator$ . 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.

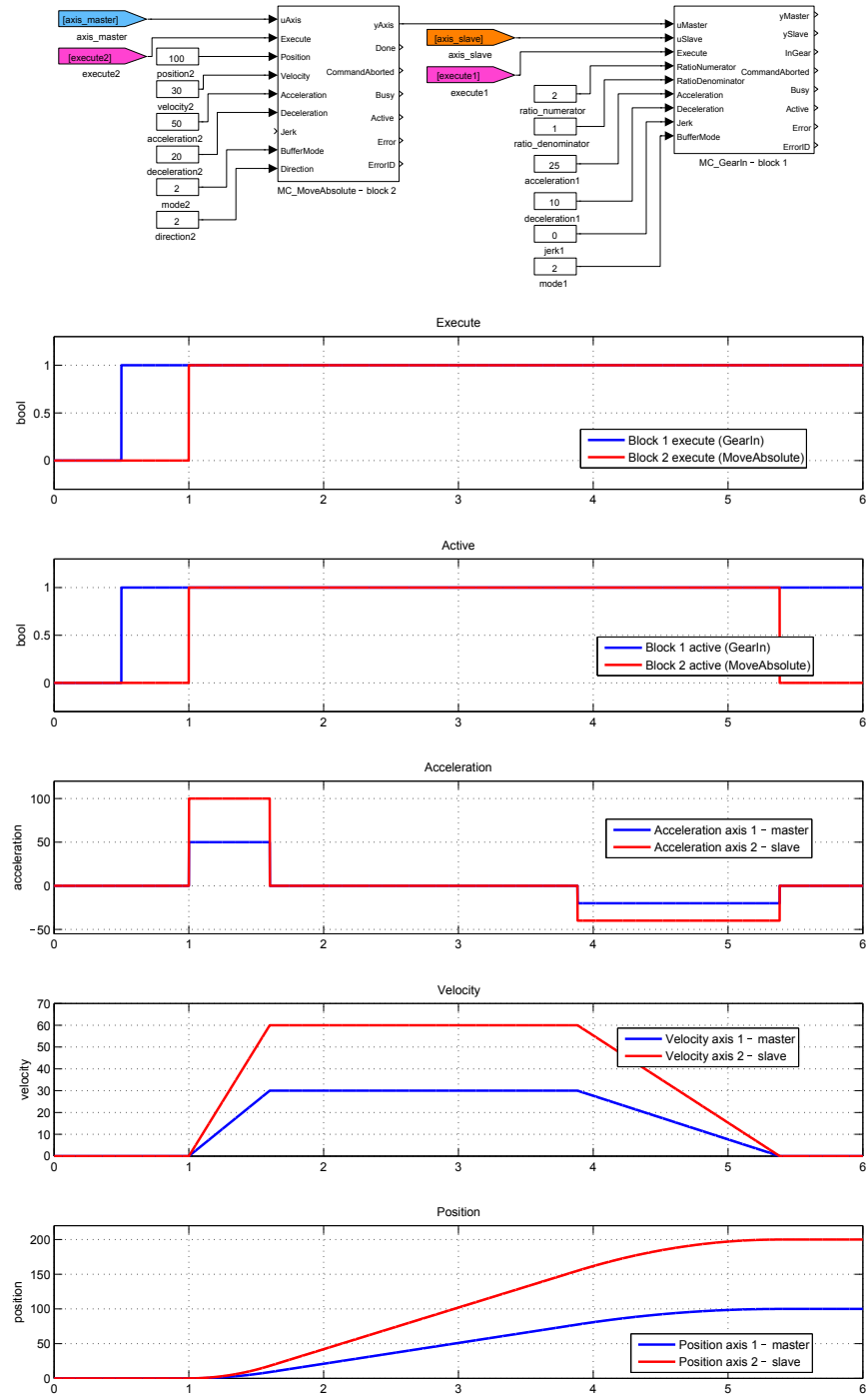
## Inputs

<b>uMaster</b>	Master axis reference	reference
<b>uSlave</b>	Slave axis reference	reference
<b>Execute</b>	The block is activated on rising edge	bool
<b>RatioNumerator</b>	Gear ratio Numerator	long
<b>RatioDenominator</b>	Gear ratio Denominator	long
<b>Acceleration</b>	Maximal allowed acceleration [unit/s <sup>2</sup> ]	double
<b>Deceleration</b>	Maximal allowed deceleration [unit/s <sup>2</sup> ]	double
<b>Jerk</b>	Maximal allowed jerk [unit/s <sup>3</sup> ]	double
<b>BufferMode</b>	Buffering mode	long
	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

<b>yMaster</b>	Master axis reference	reference
<b>ySlave</b>	Slave axis reference	reference
<b>InGear</b>	Slave axis reached gearing ratio	bool
<b>CommandAborted</b>	Algorithm was aborted	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Active</b>	The block is controlling the axis	bool
<b>Error</b>	Error occurred	bool
<b>ErrorID</b>	Result of the last operation	error
	i ..... REXYGEN general error	

Example

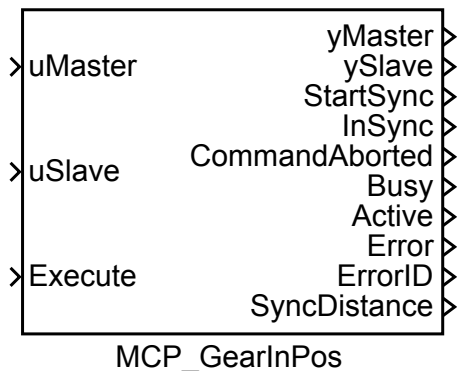


MC\_GearInPos, MCP\_GearInPos – Engage the master/slave velocity ratio in defined position

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.

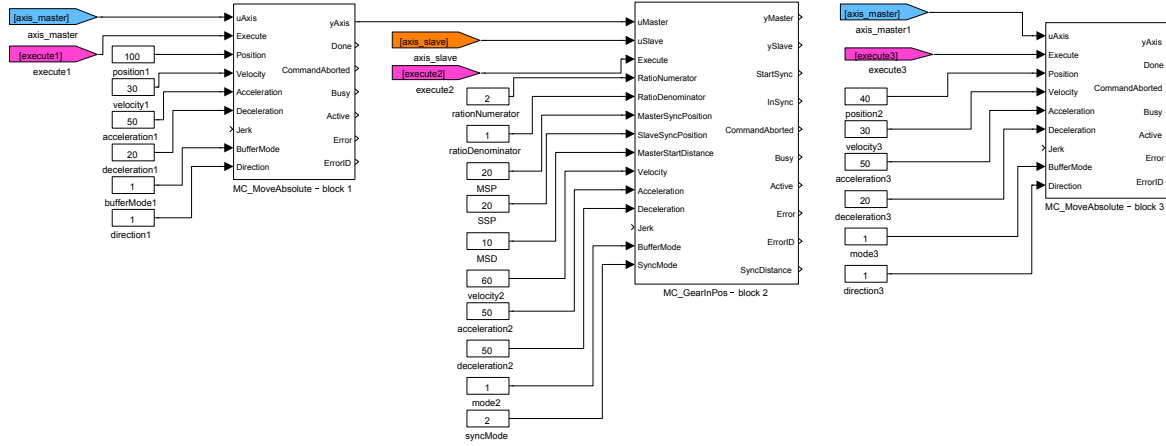
## Inputs

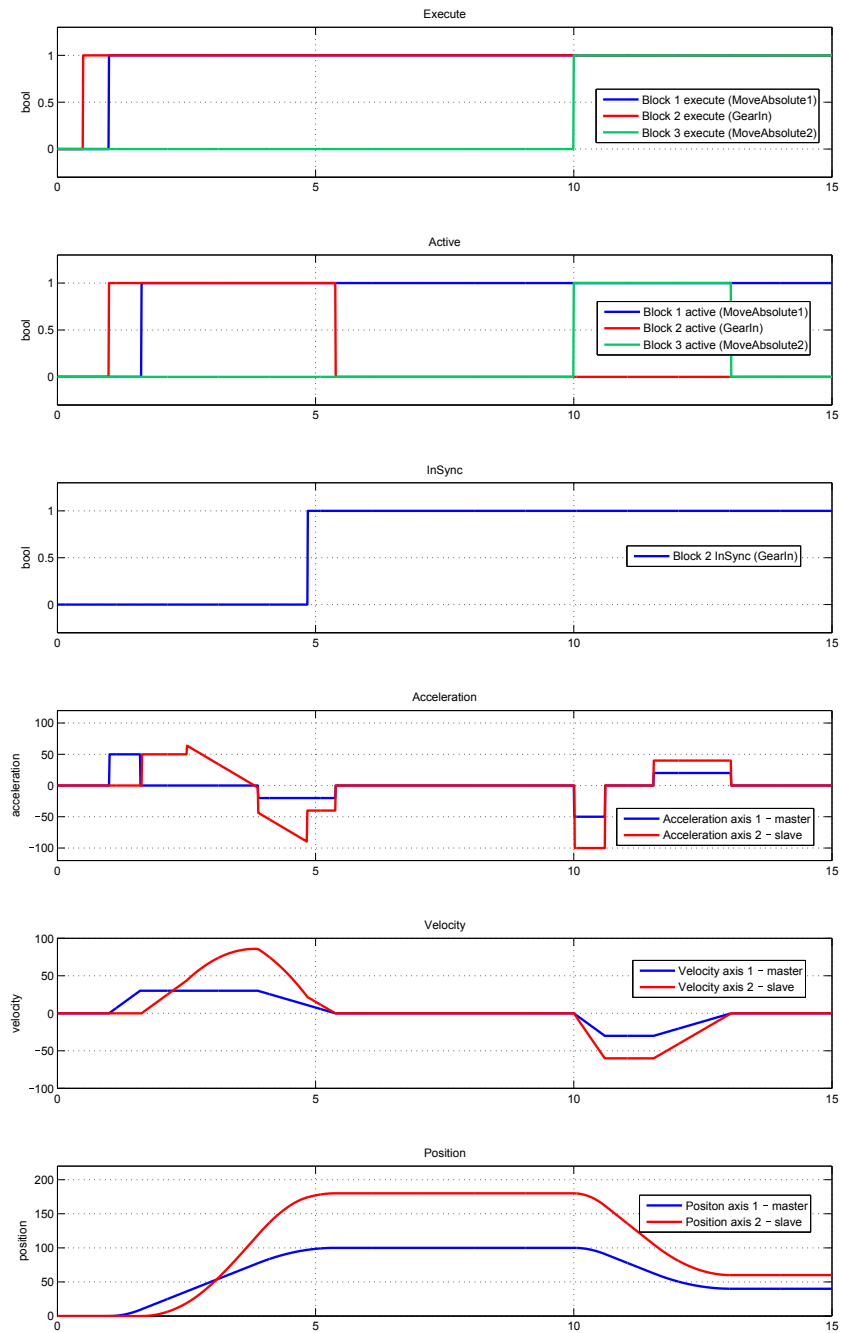
<b>uMaster</b>	Master axis reference	reference
<b>uSlave</b>	Slave axis reference	reference
<b>Execute</b>	The block is activated on rising edge	bool
<b>RatioNumerator</b>	Gear ratio Numerator	long
<b>RatioDenominator</b>	Gear ratio Denominator	long
<b>MasterSyncPosition</b>	Master position for synchronization	double
<b>SlaveSyncPosition</b>	Slave position for synchronization	double
<b>MasterStartDistance</b>	Master distance for starting gear in procedure	double
<b>Velocity</b>	Maximal allowed velocity [unit/s]	double
<b>Acceleration</b>	Maximal allowed acceleration [unit/s <sup>2</sup> ]	double
<b>Deceleration</b>	Maximal allowed deceleration [unit/s <sup>2</sup> ]	double
<b>Jerk</b>	Maximal allowed jerk [unit/s <sup>3</sup> ]	double
<b>BufferMode</b>	Buffering mode	long
	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)	
<b>SyncMode</b>	Synchronization mode (cyclic axes only)	long
	1 ..... CatchUp	
	2 ..... Shortest	
	3 ..... SlowDown	

## Outputs

<b>yMaster</b>	Master axis reference	<b>reference</b>
<b>ySlave</b>	Slave axis reference	<b>reference</b>
<b>StartSync</b>	Commanded gearing starts	<b>bool</b>
<b>InSync</b>	Slave axis reached the cam profile	<b>bool</b>
<b>CommandAborted</b>	Algorithm was aborted	<b>bool</b>
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Active</b>	The block is controlling the axis	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation	<b>error</b>
	i . . . . . REXYGEN general error	
<b>SyncDistance</b>	Position deviation of the slave axis from synchronized position	<b>double</b>

## Example

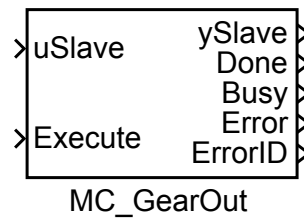




## MC\_GearOut – Disengage the master/slave velocity ratio

### 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).

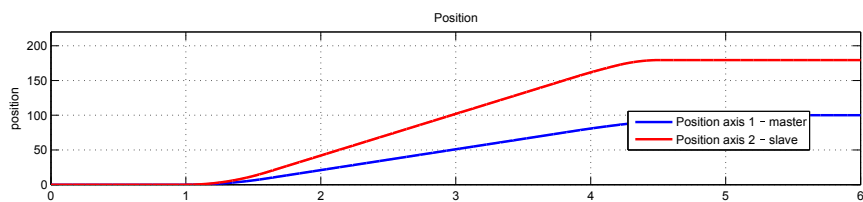
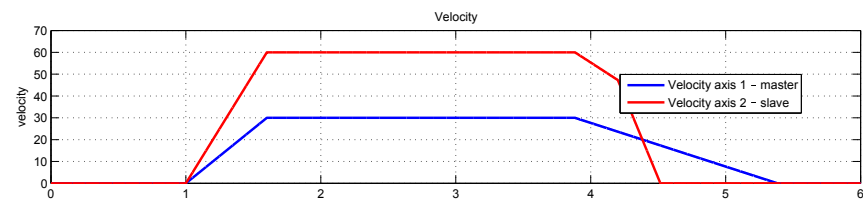
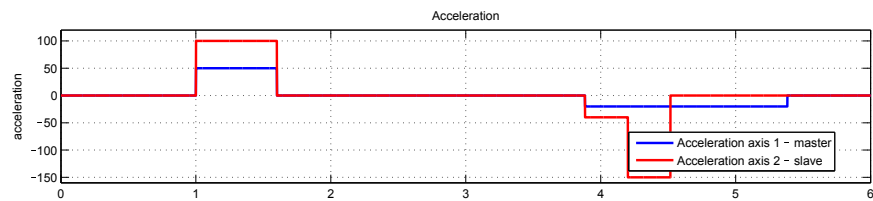
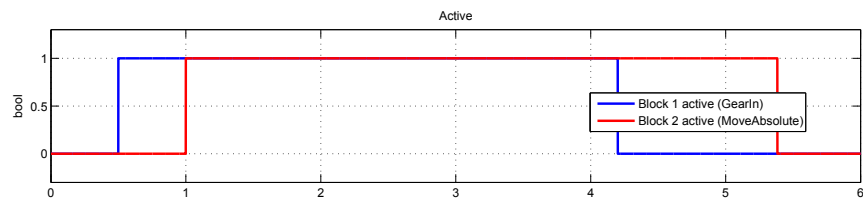
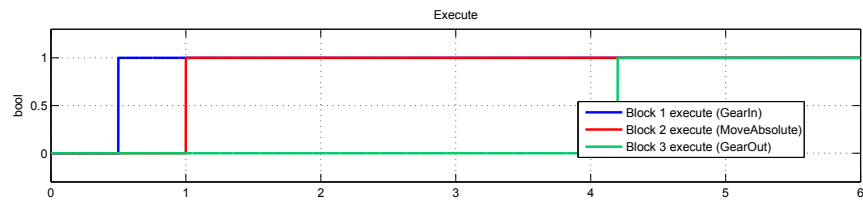
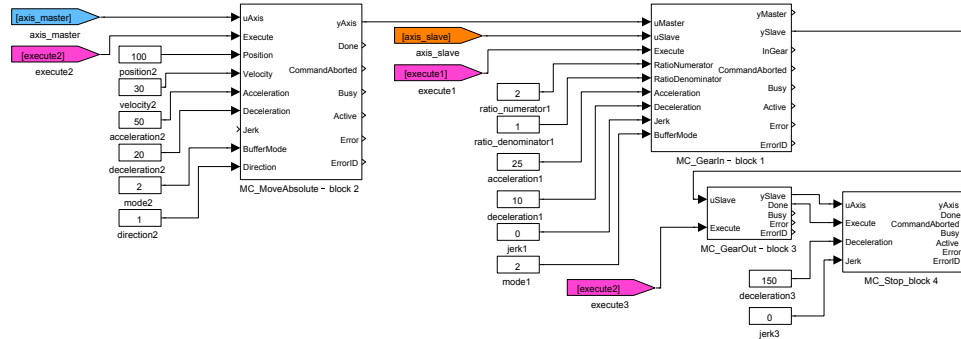
### Inputs

<b>uSlave</b>	Slave axis reference	<b>reference</b>
<b>Execute</b>	The block is activated on rising edge	<b>bool</b>

### Outputs

<b>ySlave</b>	Slave axis reference	<b>reference</b>
<b>Done</b>	Algorithm finished	<b>bool</b>
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation	<b>error</b>
	i . . . . . REXYGEN general error	

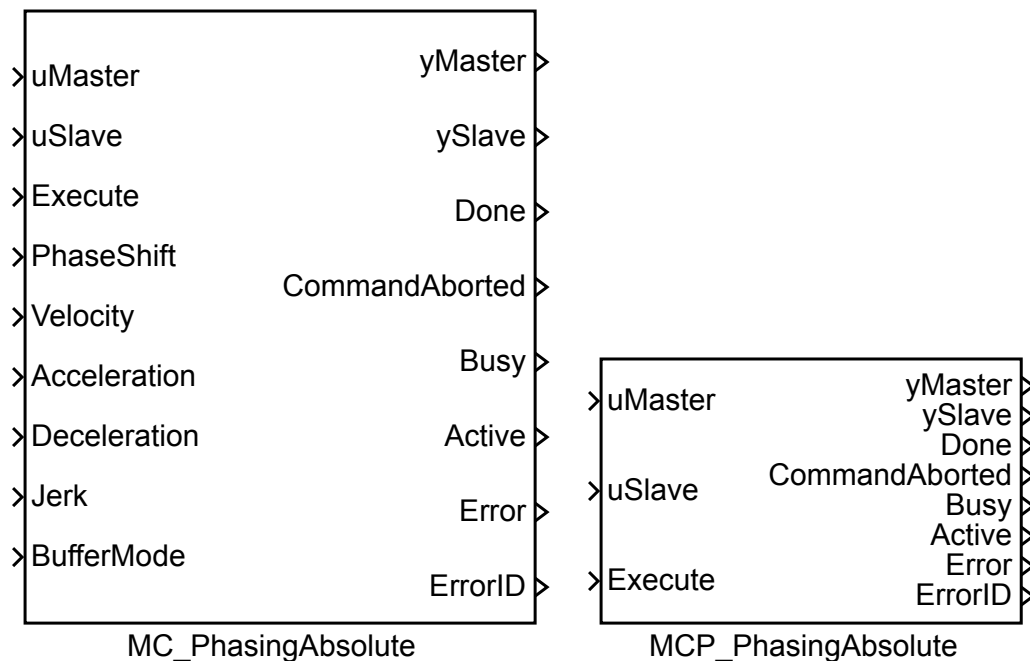
## Example



## 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

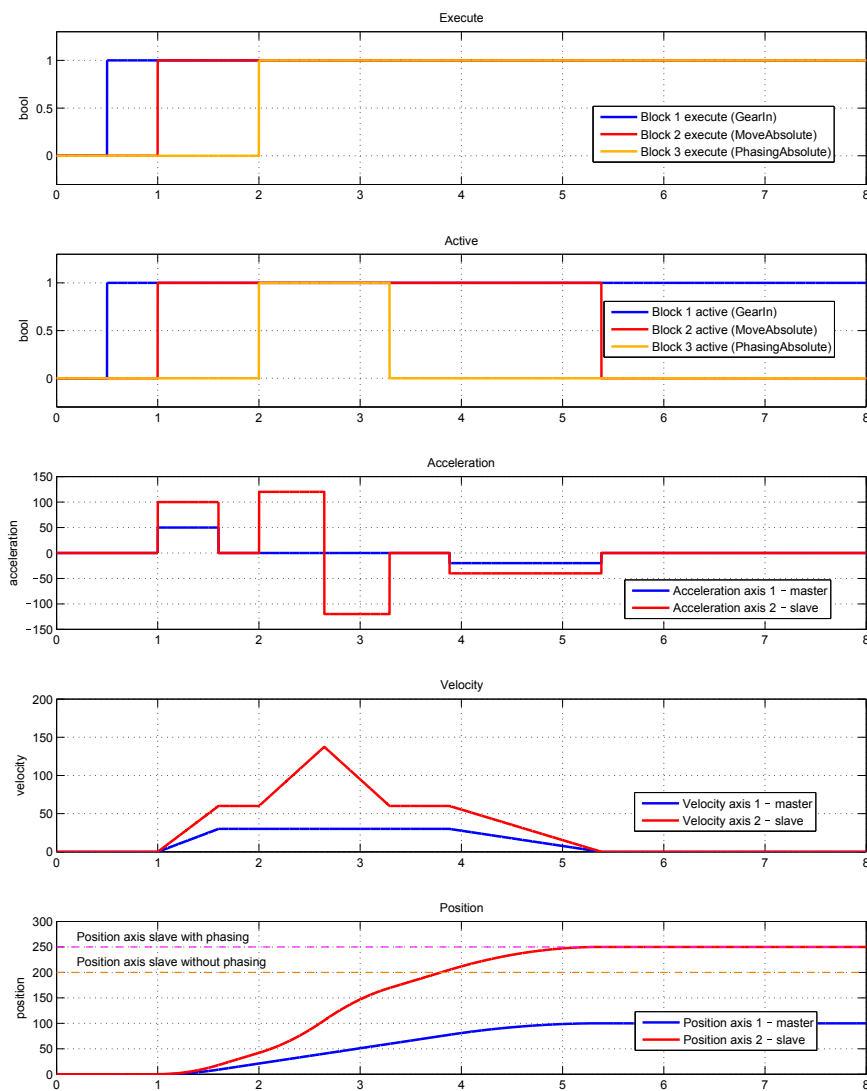
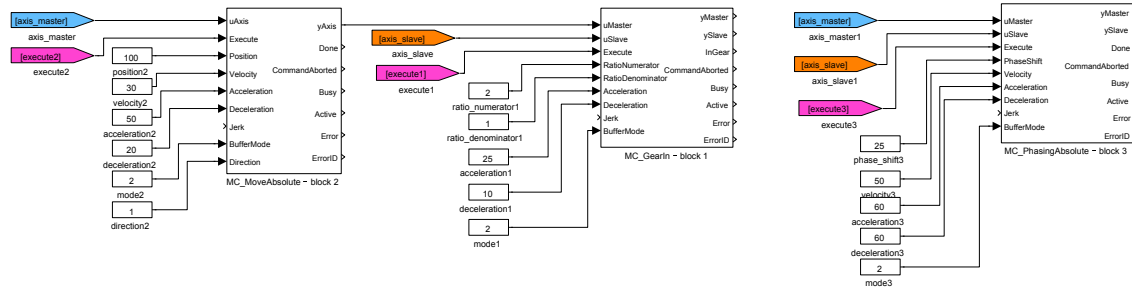
<b>uMaster</b>	Master axis reference	<b>reference</b>
<b>uSlave</b>	Slave axis reference	<b>reference</b>
<b>Execute</b>	The block is activated on rising edge	<b>bool</b>
<b>PhaseShift</b>	Requested phase shift (distance on master axis) for cam	<b>double</b>
<b>Velocity</b>	Maximal allowed velocity [unit/s]	<b>double</b>
<b>Acceleration</b>	Maximal allowed acceleration [unit/s <sup>2</sup> ]	<b>double</b>
<b>Deceleration</b>	Maximal allowed deceleration [unit/s <sup>2</sup> ]	<b>double</b>
<b>Jerk</b>	Maximal allowed jerk [unit/s <sup>3</sup> ]	<b>double</b>
<b>BufferMode</b>	Buffering mode	<b>long</b>
	1 ..... Aborting	
	2 ..... Buffered	

## Outputs

<b>yMaster</b>	Master axis reference	<b>reference</b>
<b>ySlave</b>	Slave axis reference	<b>reference</b>
<b>Done</b>	Algorithm finished	<b>bool</b>
<b>CommandAborted</b>	Algorithm was aborted	<b>bool</b>
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Active</b>	The block is controlling the axis	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation	<b>error</b>
	i ..... REXYGEN general error	



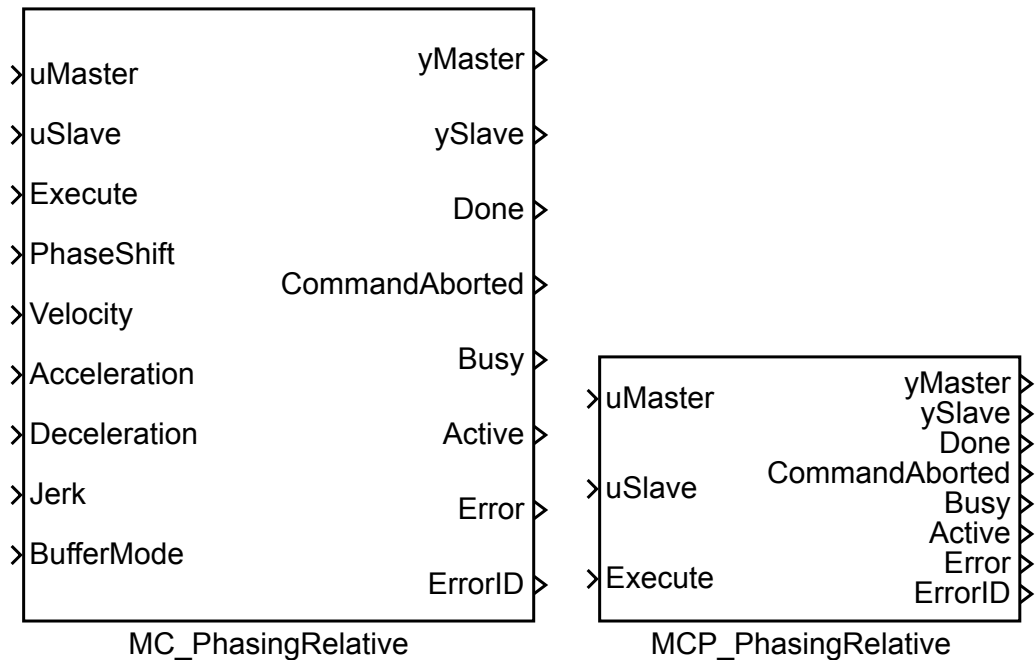
## Example



# 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

<b>uMaster</b>	Master axis reference	reference
<b>uSlave</b>	Slave axis reference	reference
<b>Execute</b>	The block is activated on rising edge	bool
<b>PhaseShift</b>	Requested phase shift (distance on master axis) for cam	double
<b>Velocity</b>	Maximal allowed velocity [unit/s]	double
<b>Acceleration</b>	Maximal allowed acceleration [unit/s <sup>2</sup> ]	double
<b>Deceleration</b>	Maximal allowed deceleration [unit/s <sup>2</sup> ]	double
<b>Jerk</b>	Maximal allowed jerk [unit/s <sup>3</sup> ]	double
<b>BufferMode</b>	Buffering mode	long
	1 ..... Aborting	
	2 ..... Buffered	

## Outputs

<b>yMaster</b>	Master axis reference	reference
<b>ySlave</b>	Slave axis reference	reference
<b>Done</b>	Algorithm finished	bool
<b>CommandAborted</b>	Algorithm was aborted	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Active</b>	The block is controlling the axis	bool
<b>Error</b>	Error occurred	bool
<b>ErrorID</b>	Result of the last operation	error
	i ..... REXYGEN general error	



## Chapter 18

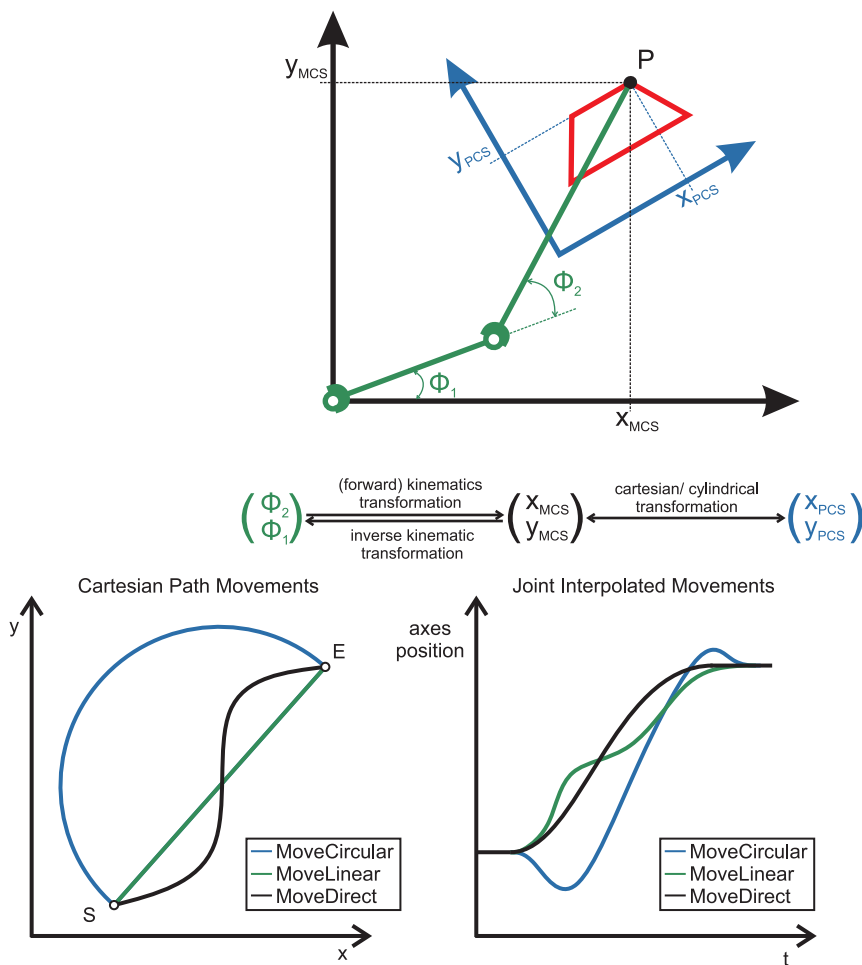
# MC\_COORD – Motion control - coordinated movement blocks

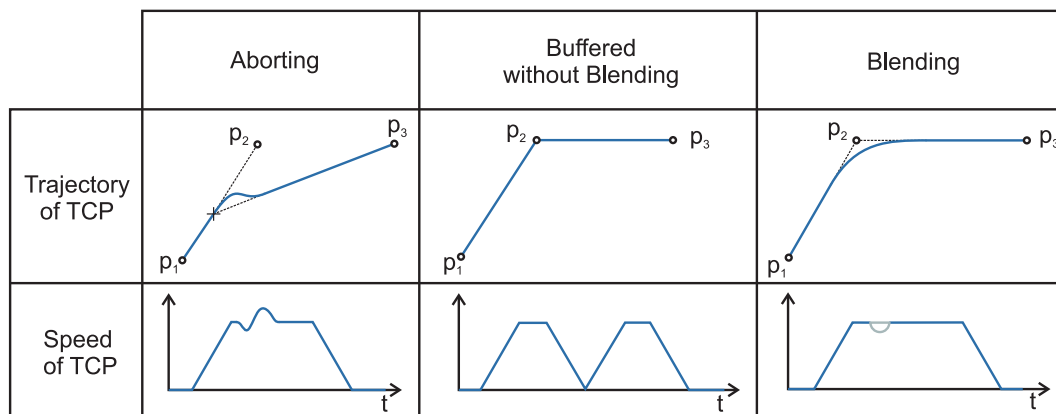
### Contents

---

RM_AxesGroup – Axes group for coordinated motion control . . .	588
RM_Feed – * MC Feeder ??? . . . . .	591
RM_Gcode – * CNC motion control . . . . .	593
MC_AddAxisToGroup – Adds one axis to a group . . . . .	595
MC_UngroupAllAxes – Removes all axes from the group . . . . .	596
MC_GroupEnable – Changes the state of a group to GroupEnable	597
MC_GroupDisable – Changes the state of a group to GroupDisabled	598
MC_SetCartesianTransform – Sets Cartesian transformation . . . .	599
MC_ReadCartesianTransform – Reads the parameter of the cartesian transformation . . . . .	601
MC_GroupSetPosition, MCP_GroupSetPosition – Sets the position of all axes in a group . . . . .	603
MC_GroupReadActualPosition – Read actual position in the selected coordinate system . . . . .	605
MC_GroupReadActualVelocity – Read actual velocity in the selected coordinate system . . . . .	606
MC_GroupReadActualAcceleration – Read actual acceleration in the selected coordinate system . . . . .	607
MC_GroupStop – Stopping a group movement . . . . .	608
MC_GroupHalt – Stopping a group movement (interruptible) . . .	611
MC_GroupInterrupt, MCP_GroupInterrupt – Read a group interrupt	616
MC_GroupContinue – Continuation of interrupted movement . . .	618
MC_GroupReadStatus – Read a group status . . . . .	619
MC_GroupReadError – Read a group error . . . . .	621
MC_GroupReset – Reset axes errors . . . . .	622

MC_MoveLinearAbsolute – Linear move to position (absolute coordinates) . . . . .	623
MC_MoveLinearRelative – Linear move to position (relative to execution point) . . . . .	627
MC_MoveCircularAbsolute – Circular move to position (absolute coordinates) . . . . .	631
MC_MoveCircularRelative – Circular move to position (relative to execution point) . . . . .	635
MC_MoveDirectAbsolute – Direct move to position (absolute coordinates) . . . . .	639
MC_MoveDirectRelative – Direct move to position (relative to execution point) . . . . .	642
MC_MovePath – General spatial trajectory generation . . . . .	645
MC_GroupSetOverride – Set group override factors . . . . .	647

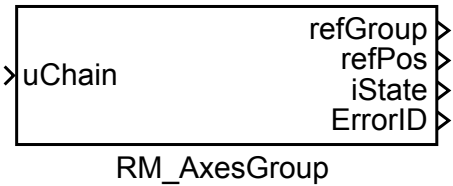




RM\_AxesGroup – Axes group for coordinated motion control

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
AcsCount	Number of axis in ACS	↓1 ↑16 ⊙6	long
PosCount	Number of position axis	↓1 ↑6 ⊙3	long
Velocity	Maximal allowed velocity [unit/s]		double
Acceleration	Maximal allowed acceleration [unit/s <sup>2</sup> ]		double
Jerk	Maximal allowed jerk [unit/s <sup>3</sup> ]		double

Outputs

refGroup	Axes group reference	reference
refPos	Position, velocity and acceleration vector	reference
iState	Group status	long
	0 ..... Disabled	
	1 ..... Standby	
	2 ..... Homing	
	6 ..... Moving	
	7 ..... Stopping	
	8 ..... Error stop	



```

stateDiagram-v2
    state GroupMoving
    state GroupStopping
    state GroupErrorStop
    state GroupStandby
    state GroupDisabled
    state GroupHoming

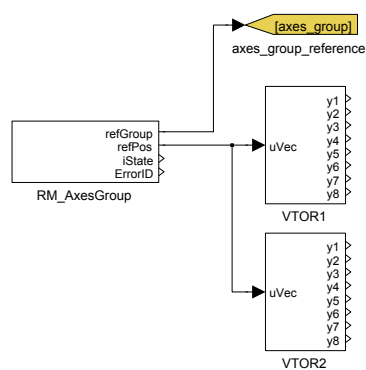
    GroupMoving --> GroupMoving : Note 1 and MC_GroupHalt
    GroupMoving --> GroupStopping : MC_GroupStop
    GroupMoving --> GroupStandby : Done
    GroupMoving --> GroupErrorStop : Note 3
    GroupStopping --> GroupStopping : Note 2
    GroupStopping --> GroupErrorStop : Error
    GroupErrorStop --> GroupErrorStop : Note 2
    GroupErrorStop --> GroupStandby : MC_GroupReset
    GroupStandby --> GroupStandby : Note 5
    GroupStandby --> GroupDisabled : MC_GroupEnable
    GroupStandby --> GroupHoming : MC_GroupHome
    GroupDisabled --> GroupDisabled : Note 4
    GroupHoming --> GroupHoming : Note 1
    GroupHoming --> GroupStandby : Done
    GroupHoming --> GroupErrorStop : Error
    GroupHoming --> GroupStopping : MC_GroupStop
  
```

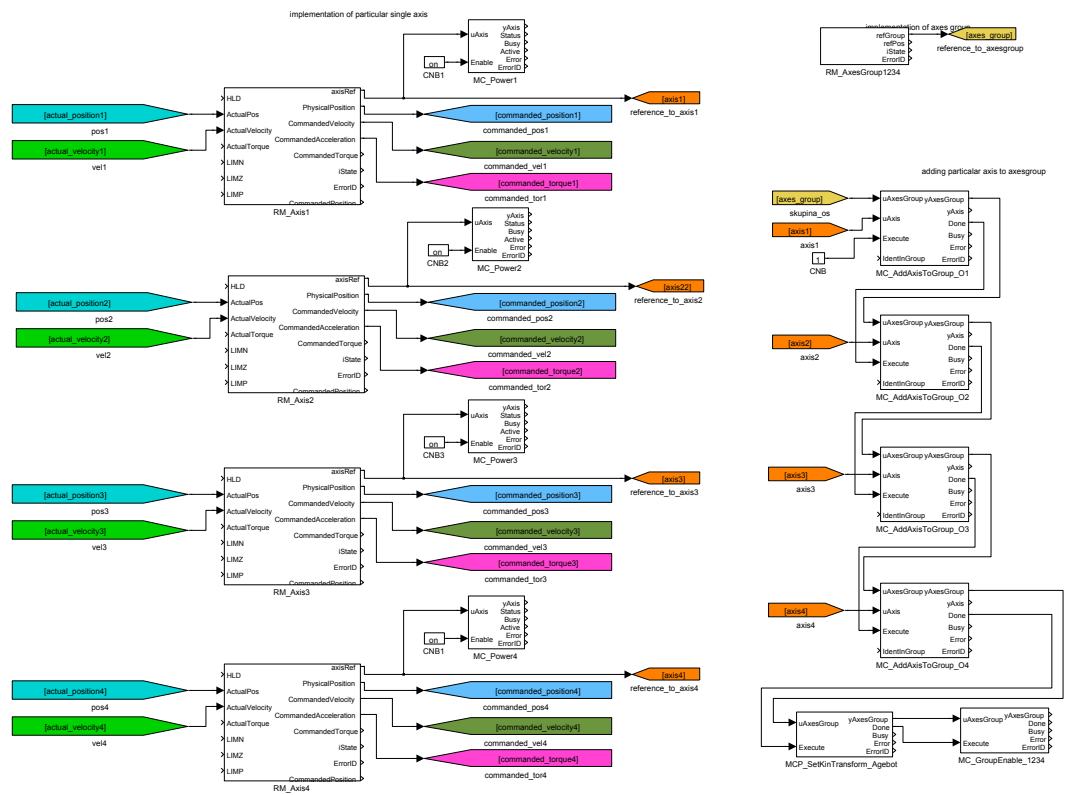
MC\_Group control system state transitions:

- GroupMoving** (Start State)
  - Self-loop: Note 1 and MC\_GroupHalt
  - Transition to **GroupStopping**: MC\_GroupStop
  - Transition to **GroupStandby**: Done
  - Transition to **GroupErrorStop**: Note 3
- GroupStopping**
  - Self-loop: Note 2
  - Transition to **GroupErrorStop**: Error
- GroupErrorStop**
  - Self-loop: Note 2
  - Transition to **GroupStandby**: MC\_GroupReset
- GroupStandby**
  - Self-loop: Note 5
  - Transition to **GroupDisabled**: MC\_GroupEnable
  - Transition to **GroupHoming**: MC\_GroupHome
- GroupDisabled**
  - Self-loop: Note 4
- GroupHoming**
  - Self-loop: Note 1
  - Transition to **GroupStandby**: Done
  - Transition to **GroupErrorStop**: Error
  - Transition to **GroupStopping**: MC\_GroupStop

MC\_Group actions:

- GroupMoving**: MC\_GroupHalt
- GroupStopping**: Error
- GroupErrorStop**: MC\_GroupReset
- GroupStandby**: MC\_GroupEnable, MC\_GroupHome
- GroupDisabled**: MC\_GroupDisable, MC\_UngroupAllAxes, MC\_RemoveAxisFromGroup
- GroupHoming**: MC\_AssAxisToGroup, MC\_RemoveAxisFromGroup

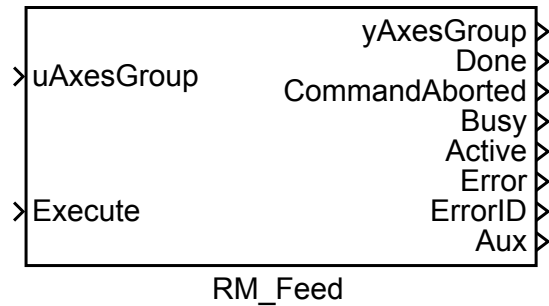




## RM\_Feed — \* MC Feeder ???

### 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
Relative	0		bool
CoordSystem	0	↓1 ↑3 ⊙2	long
BufferMode	0	↓1 ↑6 ⊙1	long
TransitionMode	0	↓0 ↑15 ⊙1	long
TransitionParameter	0		double

### Outputs

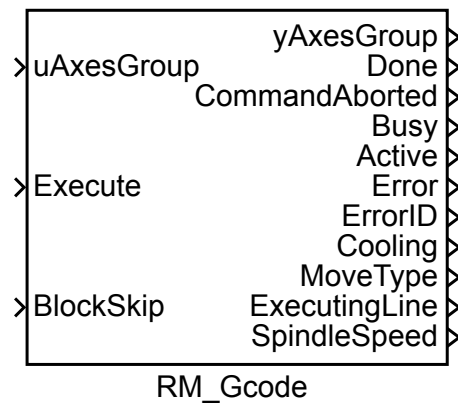
yAxesGroup	Axes group reference	reference
Done	Algorithm finished	bool
CommandAborted	Algorithm was aborted	bool
Busy	Algorithm not finished yet	bool
Active	The block is controlling the axis	bool
Error	Error occurred	bool
ErrorID	Result of the last operation	error

Aux	0	double
-----	---	--------

## 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

<b>uAxesGroup</b>	Axes group reference	reference
<b>Execute</b>	The block is activated on rising edge	bool
<b>BlockSkip</b>	MILAN	bool

### Parameters

<b>BaseDir</b>	Directory of the G-code files	string
<b>MainFile</b>	Source file number	long
<b>CoordSystem</b>	0	↓1 ↑3 ⊙3 long
<b>BufferMode</b>	Buffering mode	⊙1 long
	1 ..... Aborting	
	2 ..... Buffered	
	3 ..... Blending low	
	4 ..... Blending high	
	5 ..... Blending previous	
	6 ..... Blending next	

<b>TransitionMode</b>	Transition mode in blending mode	⊙1	long
1	..... TMNone		
2	..... TMStartVelocity		
3	..... TMConstantVelocity		
4	..... TMCornerDistance		
5	..... TMMaxCornerDeviation		
11	.... Smooth		
<b>TransitionParameter</b>	Parametr for transition (depends on transition mode)		double
<b>workOffsets</b>	Sets with initial coordinate		double
	⊙[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]		
<b>toolOffsets</b>	Sets of tool offset	⊙[0 0 0]	double
<b>cutterOffsets</b>	Tool radii	⊙[0 0 0]	double

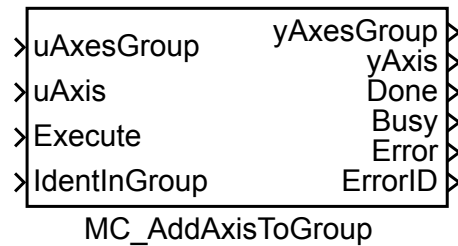
## Outputs

<b>yAxesGroup</b>	Axes group reference	reference
<b>Done</b>	Algorithm finished	bool
<b>CommandAborted</b>	Algorithm was aborted	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Active</b>	The block is controlling the axis	bool
<b>Error</b>	Error occurred	bool
<b>ErrorID</b>	Result of the last operation	error
i	..... REXYGEN general error	
<b>Cooling</b>	Cooling	bool
<b>MoveType</b>	Command execution	long
<b>ExecutingLine</b>	Current line of G-code	long
<b>SpindleSpeed</b>	Spindle speed	long

## 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

<b>uAxesGroup</b>	Axes group reference	<b>reference</b>
<b>uAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> connections are allowed)	<b>reference</b>
<b>Execute</b>	The block is activated on rising edge	<b>bool</b>
<b>IdentInGroup</b>	The order of axes in the group (0 = first unassigned)	<b>long</b>

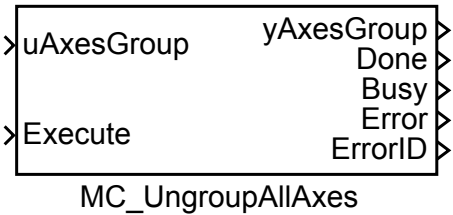
## Outputs

<b>yAxesGroup</b>	Axes group reference	<b>reference</b>
<b>yAxis</b>	Axis reference (only <b>RM_Axis.axisRef-uAxis</b> or <b>yAxis-uAxis</b> connections are allowed)	<b>reference</b>
<b>Done</b>	Algorithm finished	<b>bool</b>
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation	<b>error</b>
	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**

<b>uAxesGroup</b>	Axes group reference	<b>reference</b>
<b>Execute</b>	The block is activated on rising edge	<b>bool</b>

**Outputs**

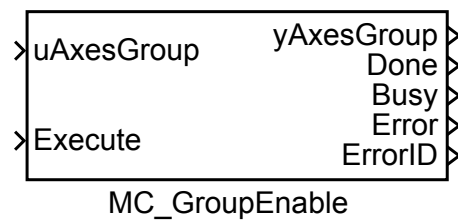
<b>yAxesGroup</b>	Axes group reference	<b>reference</b>
<b>Done</b>	Algorithm finished	<b>bool</b>
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation	<b>error</b>
	i . . . . . REXYGEN general error	



## 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

<b>uAxesGroup</b>	Axes group reference	<b>reference</b>
<b>Execute</b>	The block is activated on rising edge	<b>bool</b>

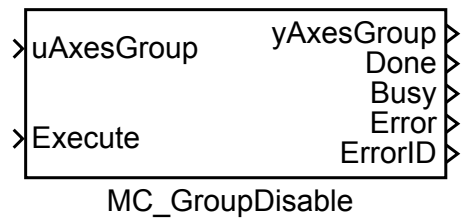
### Outputs

<b>yAxesGroup</b>	Axes group reference	<b>reference</b>
<b>Done</b>	Algorithm finished	<b>bool</b>
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation	<b>error</b>
	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**

<b>uAxesGroup</b>	Axes group reference	<b>reference</b>
<b>Execute</b>	The block is activated on rising edge	<b>bool</b>

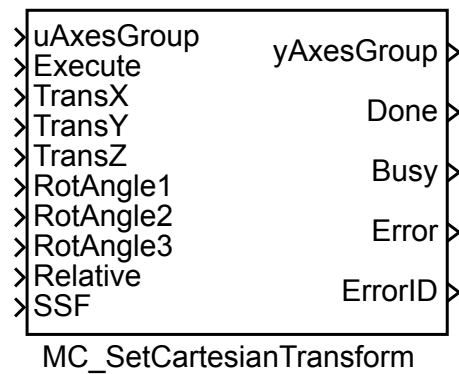
**Outputs**

<b>yAxesGroup</b>	Axes group reference	<b>reference</b>
<b>Done</b>	Algorithm finished	<b>bool</b>
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation	<b>error</b>
	i . . . . . REXYGEN general error	

## MC\_SetCartesianTransform – Sets Cartesian transformation

Block Symbol

Licence: [COORDINATED MOTION](#)



### Function Description

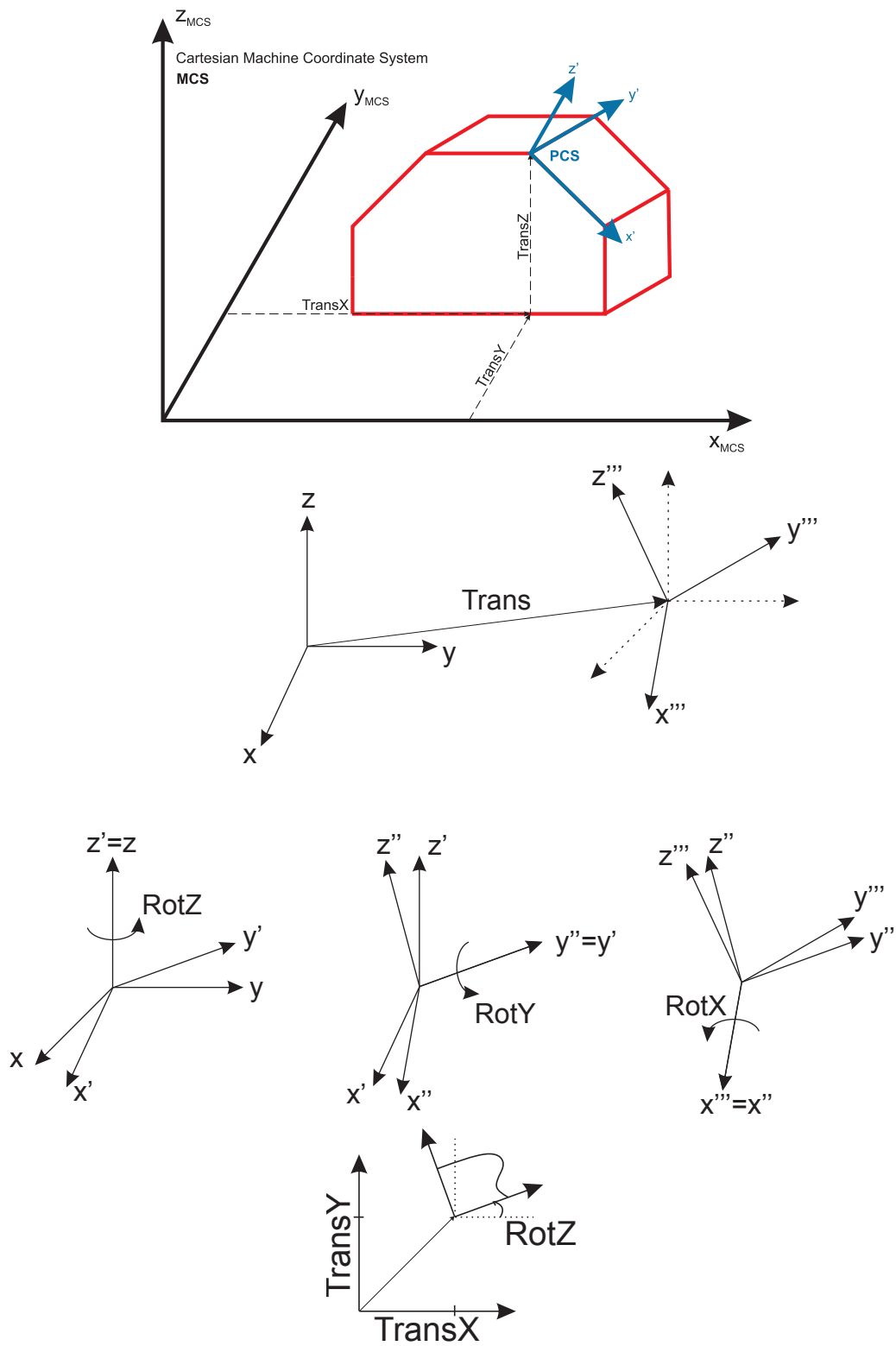
The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

### Inputs

<b>uAxesGroup</b>	Axes group reference	<b>reference</b>
<b>Execute</b>	The block is activated on rising edge	<b>bool</b>
<b>TransX</b>	X-component of translation vector	<b>double</b>
<b>TransY</b>	Y-component of translation vector	<b>double</b>
<b>TransZ</b>	Z-component of translation vector	<b>double</b>
<b>RotAngle1</b>	Rotation angle component	<b>double</b>
<b>RotAngle2</b>	Rotation angle component	<b>double</b>
<b>RotAngle3</b>	Rotation angle component	<b>double</b>
<b>Relative</b>	Mode of position inputs	<b>bool</b>

### Outputs

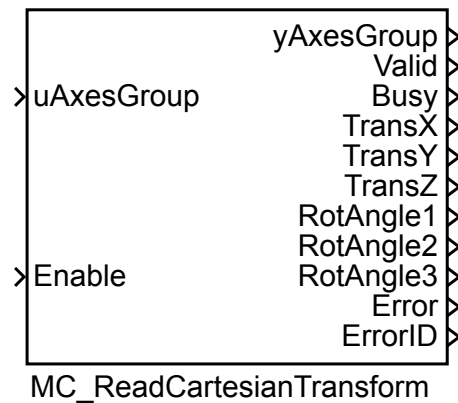
<b>yAxesGroup</b>	Axes group reference	<b>reference</b>
<b>Done</b>	Algorithm finished	<b>bool</b>
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation	<b>error</b>
	i   ..... REXYGEN general error	



## 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

<code>uAxesGroup</code>	Axes group reference	reference
<code>Enable</code>	Block function is enabled	bool

### Outputs

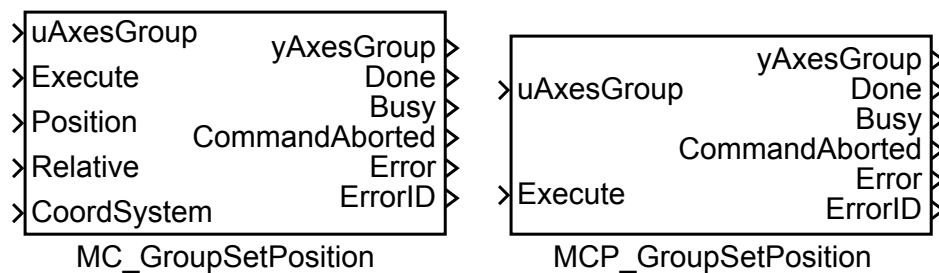
<code>yAxesGroup</code>	Axes group reference	reference
<code>Valid</code>	Output value is valid	bool
<code>Busy</code>	Algorithm not finished yet	bool
<code>TransX</code>	X-component of translation vector	double
<code>TransY</code>	Y-component of translation vector	double
<code>TransZ</code>	Z-component of translation vector	double
<code>RotAngle1</code>	Rotation angle component	double
<code>RotAngle2</code>	Rotation angle component	double
<code>RotAngle3</code>	Rotation angle component	double
<code>Error</code>	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
	1 ..... ACS	
	2 ..... MCS	
	3 ..... PCS	

### Outputs

<code>yAxesGroup</code>	Axes group reference	reference
-------------------------	----------------------	-----------

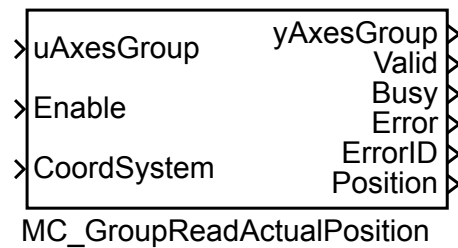
Done	Algorithm finished	bool
Busy	Algorithm not finished yet	bool
CommandAborted	Algorithm was aborted	bool
Error	Error occurred	bool
ErrorID	Result of the last operation	error
	i ..... REXYGEN general error	



## MC\_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

<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
	1 ..... ACS	
	2 ..... MCS	
	3 ..... PCS	

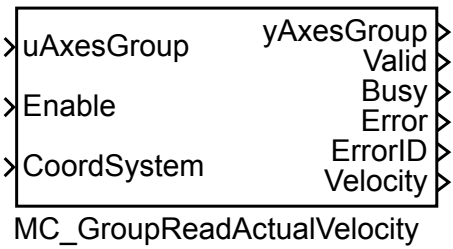
### Outputs

<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>Position</code>	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

uAxesGroup	Axes group reference	reference
Enable	Block function is enabled	bool
CoordSystem	Reference to the coordinate system used	long
	1 ..... ACS	
	2 ..... MCS	
	3 ..... PCS	

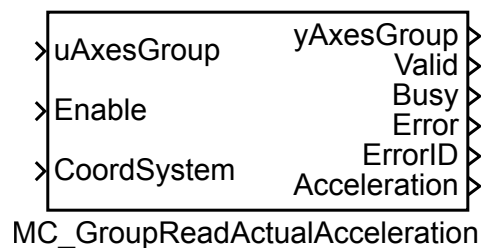
Outputs

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	
Velocity	xxx	reference

## MC\_GroupReadActualAcceleration – Read actual acceleration in the selected coordinate system

Block Symbol

Licence: [COORDINATED MOTION](#)



### Function Description

The function block `MC_GroupReadActualAcceleration` returns the actual velocity in the selected coordinate system of an axes group. The position is valid only if the output `Valid` is true which is achieved by setting the input `Enable` on true.

### Inputs

<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
	1 ..... ACS	
	2 ..... MCS	
	3 ..... PCS	

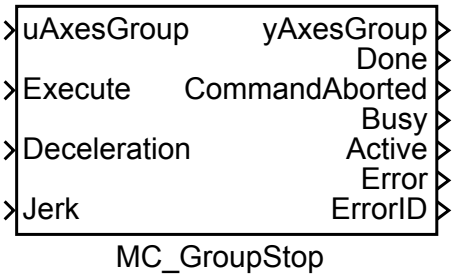
### Outputs

<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\_GroupStop – Stopping a group movement

Block Symbol

Licence: [COORDINATED MOTION](#)



Function Description

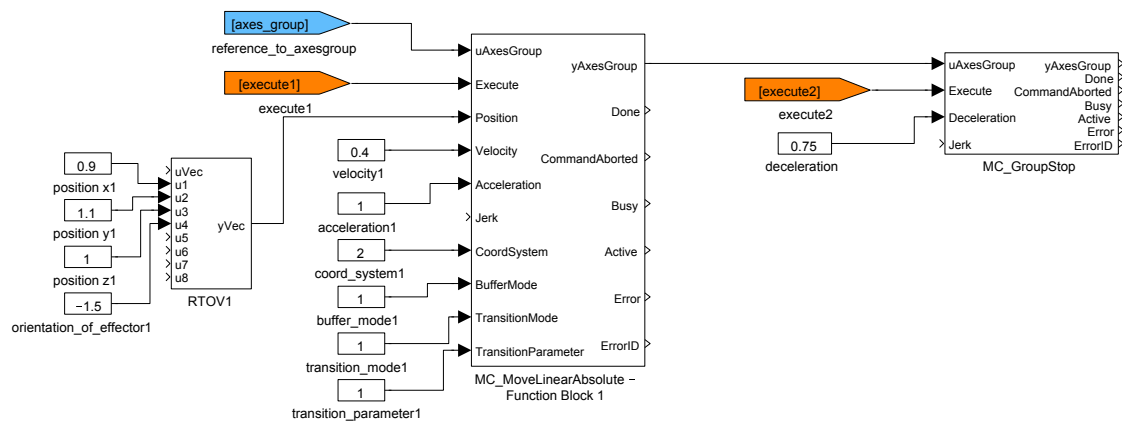
The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

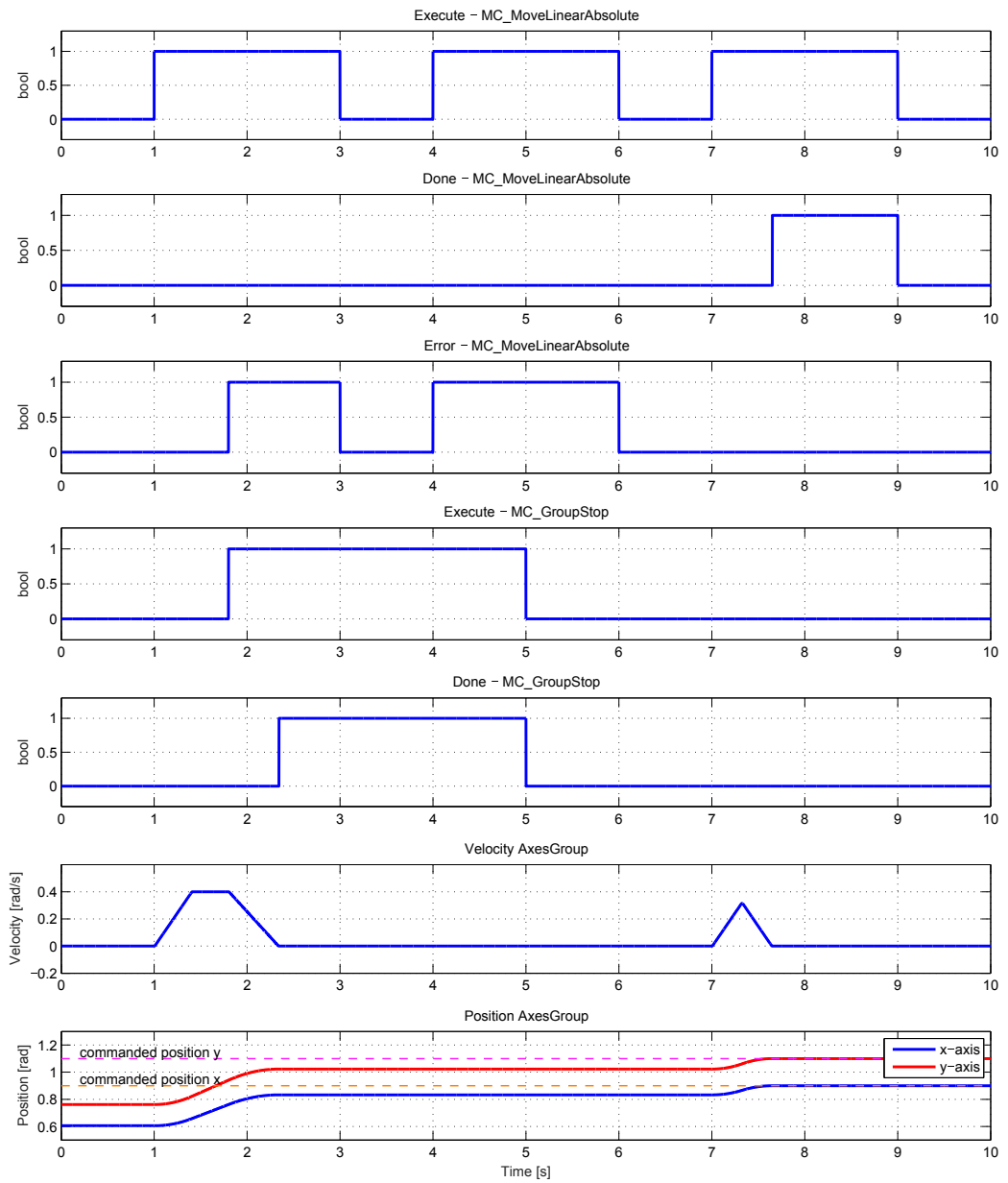
Inputs

uAxesGroup	Axes group reference	reference
Execute	The block is activated on rising edge	bool
Deceleration	Maximal allowed deceleration [unit/s <sup>2</sup> ]	double
Jerk	Maximal allowed jerk [unit/s <sup>3</sup> ]	double

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	

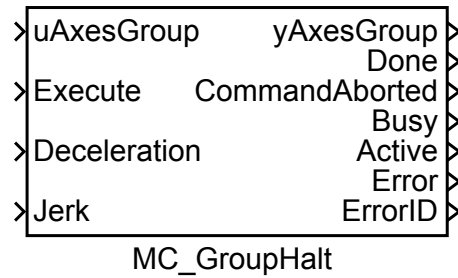




## MC\_GroupHalt – Stopping a group movement (interruptible)

Block Symbol

Licence: [COORDINATED MOTION](#)



### Function Description

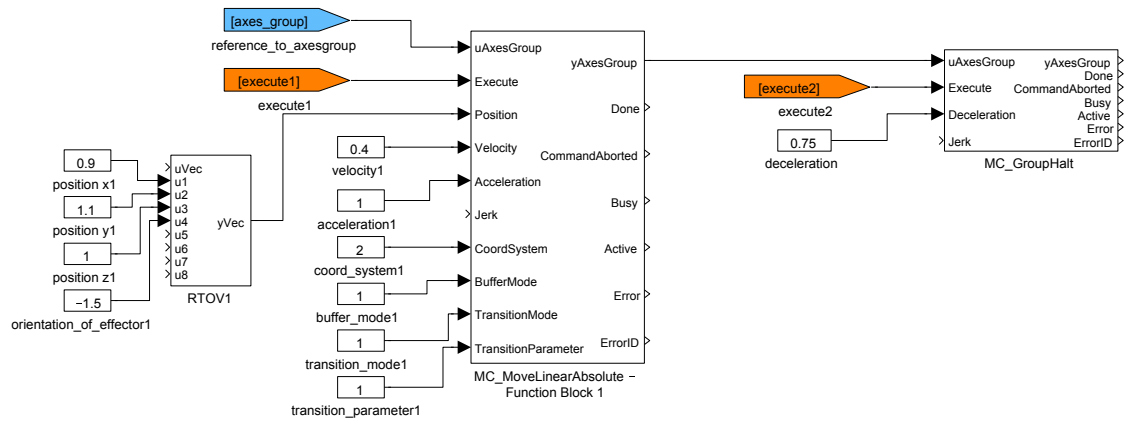
The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

### Inputs

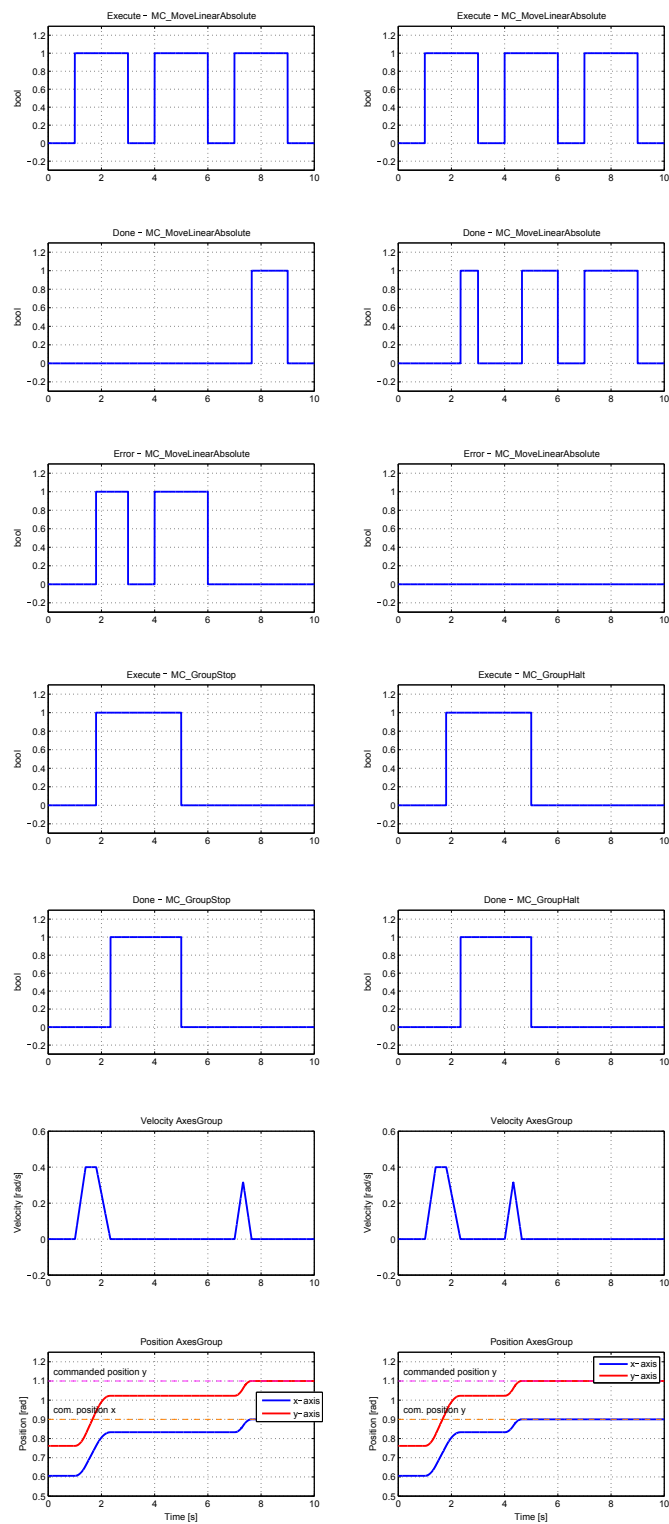
<b>uAxesGroup</b>	Axes group reference	reference
<b>Execute</b>	The block is activated on rising edge	bool
<b>Deceleration</b>	Maximal allowed deceleration [unit/s <sup>2</sup> ]	double
<b>Jerk</b>	Maximal allowed jerk [unit/s <sup>3</sup> ]	double

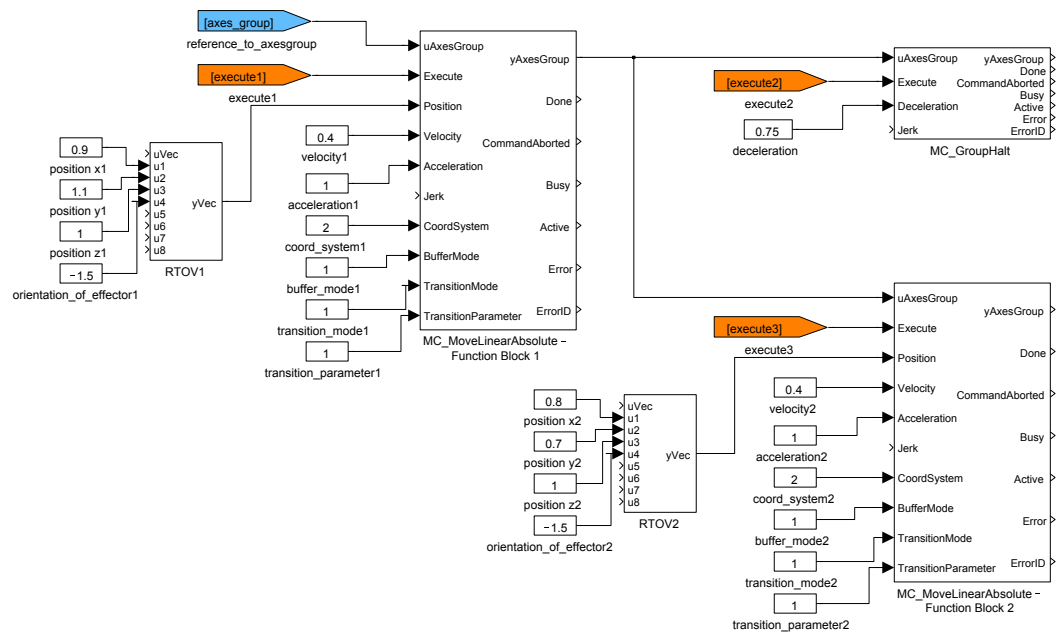
### Outputs

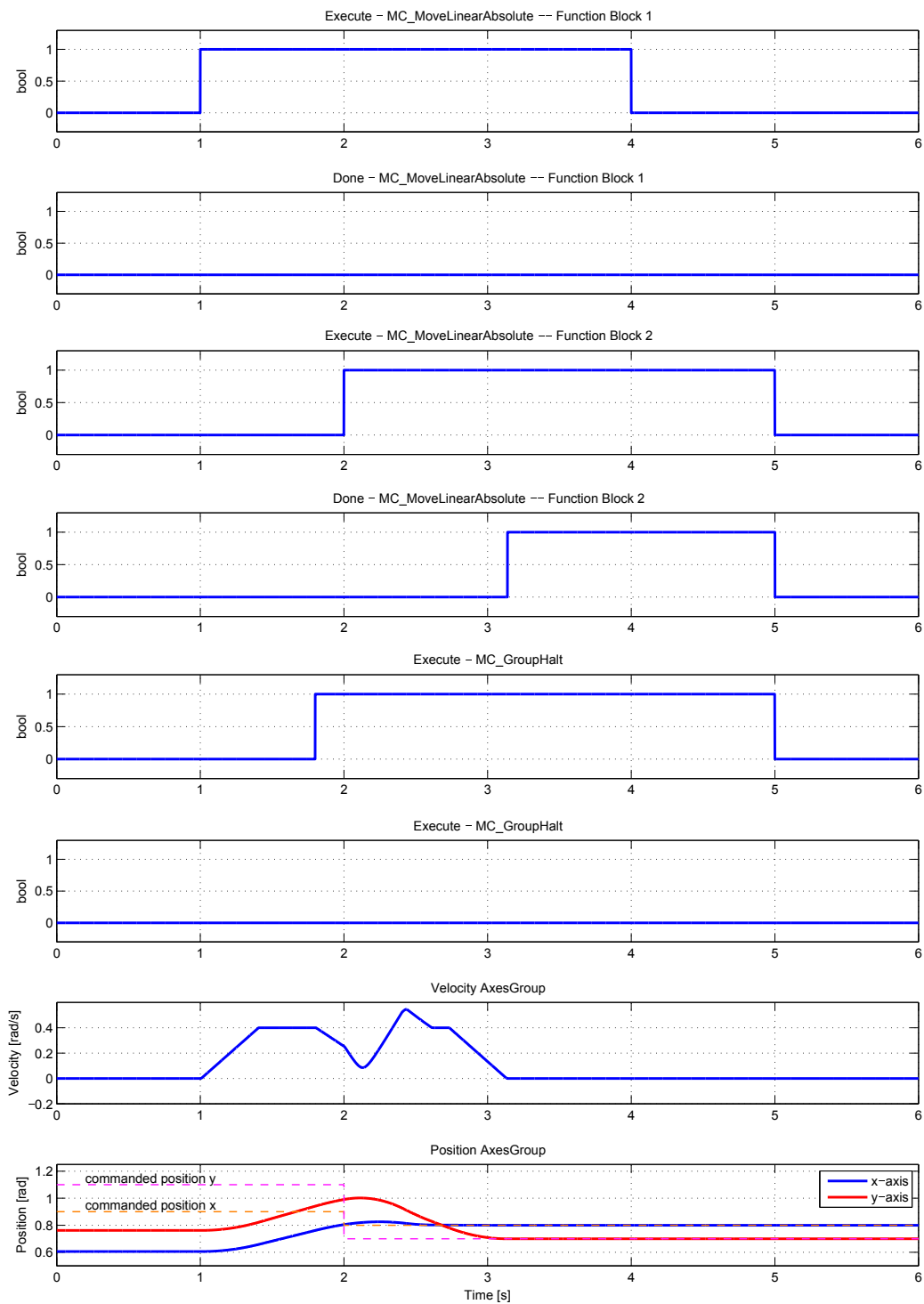
<b>yAxesGroup</b>	Axes group reference	reference
<b>Done</b>	Algorithm finished	bool
<b>CommandAborted</b>	Algorithm was aborted	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Active</b>	The block is controlling the axis	bool
<b>Error</b>	Error occurred	bool
<b>ErrorID</b>	Result of the last operation	error
	i . . . . . REXYGEN general error	







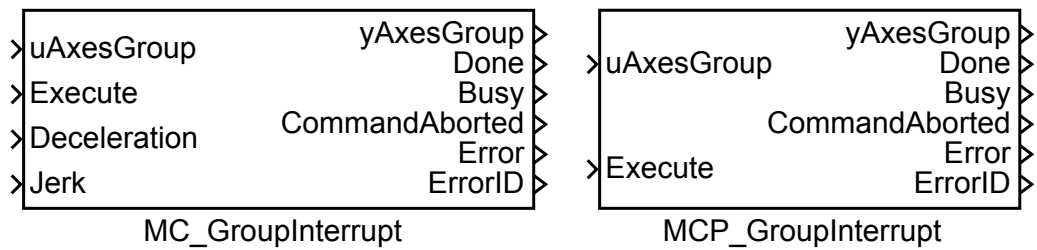




MC\_GroupInterrupt, MCP\_GroupInterrupt – Read a group interrupt

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

<b>uAxesGroup</b>	Axes group reference	reference
<b>Execute</b>	The block is activated on rising edge	bool
<b>Deceleration</b>	Maximal allowed deceleration [unit/s <sup>2</sup> ]	double
<b>Jerk</b>	Maximal allowed jerk [unit/s <sup>3</sup> ]	double

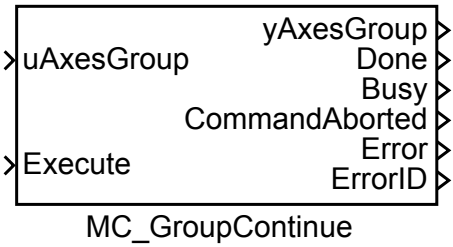
Outputs

<b>yAxesGroup</b>	Axes group reference	reference
<b>Done</b>	Algorithm finished	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>CommandAborted</b>	Algorithm was aborted	bool

Error	Error occurred	bool
ErrorID	Result of the last operation	error
	i ..... REXYGEN general error	

MC\_GroupContinue – Continuation of interrupted movement

Block SymbolLicence: 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

<code>uAxesGroup</code>	Axes group reference	reference
<code>Execute</code>	The block is activated on rising edge	bool

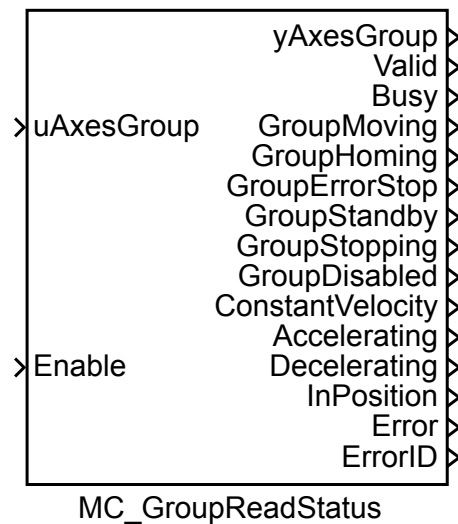
Outputs

<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
<code>ErrorID</code>	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

<code>uAxesGroup</code>	Axes group reference	reference
<code>Enable</code>	Block function is enabled	bool

### 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>GroupMoving</code>	State GroupMoving	bool
<code>GroupHoming</code>	State GroupHoming	bool
<code>GroupErrorStop</code>	State ErrorStop	bool
<code>GroupStandby</code>	State Standby	bool
<code>GroupStopping</code>	State Stopping	bool
<code>GroupDisabled</code>	State Disabled	bool

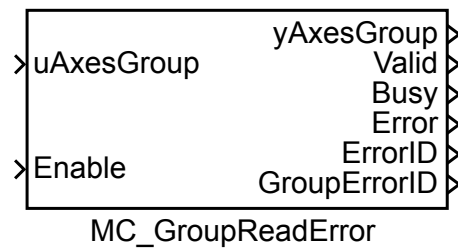
ConstantVelocity	Constant velocity motion	bool
Accelerating	Accelerating	bool
Decelerating	Decelerating	bool
InPosition	Symptom achieve the desired position	bool
Error	Error occurred	bool
ErrorID	Result of the last operation	error
	i . . . . . REXYGEN general error	



## MC\_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

<b>uAxesGroup</b>	Axes group reference	reference
<b>Enable</b>	Block function is enabled	bool

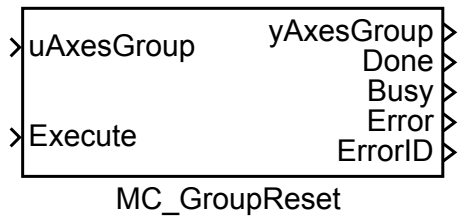
## Outputs

<b>yAxesGroup</b>	Axes group reference	reference
<b>Valid</b>	Output value is valid	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Error</b>	Error occurred	bool
<b>ErrorID</b>	Result of the last operation i ..... REXYGEN general error	error
<b>GroupErrorID</b>	Result of the last operation i ..... REXYGEN general error	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

<b>uAxesGroup</b>	Axes group reference	<b>reference</b>
<b>Execute</b>	The block is activated on rising edge	<b>bool</b>

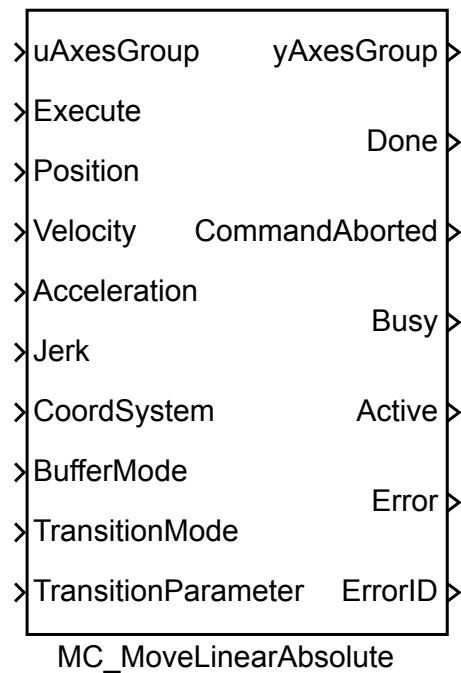
Outputs

<b>yAxesGroup</b>	Axes group reference	<b>reference</b>
<b>Done</b>	Algorithm finished	<b>bool</b>
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation	<b>error</b>
	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.

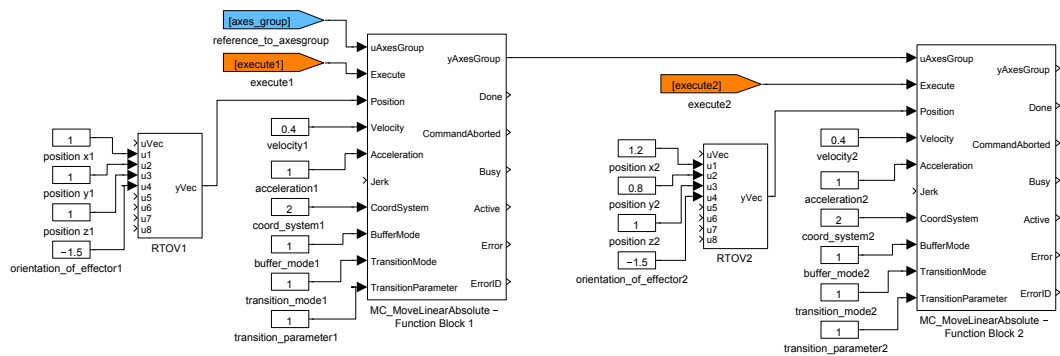
### Inputs

<b>uAxesGroup</b>	Axes group reference	<b>reference</b>
<b>Execute</b>	The block is activated on rising edge	<b>bool</b>
<b>Position</b>	Array of coordinates (positions and orientations)	<b>reference</b>
<b>Velocity</b>	Maximal allowed velocity [unit/s]	<b>double</b>
<b>Acceleration</b>	Maximal allowed acceleration [unit/s <sup>2</sup> ]	<b>double</b>
<b>Jerk</b>	Maximal allowed jerk [unit/s <sup>3</sup> ]	<b>double</b>
<b>CoordSystem</b>	Reference to the coordinate system used	<b>long</b>
	1 ..... ACS	
	2 ..... MCS	
	3 ..... PCS	

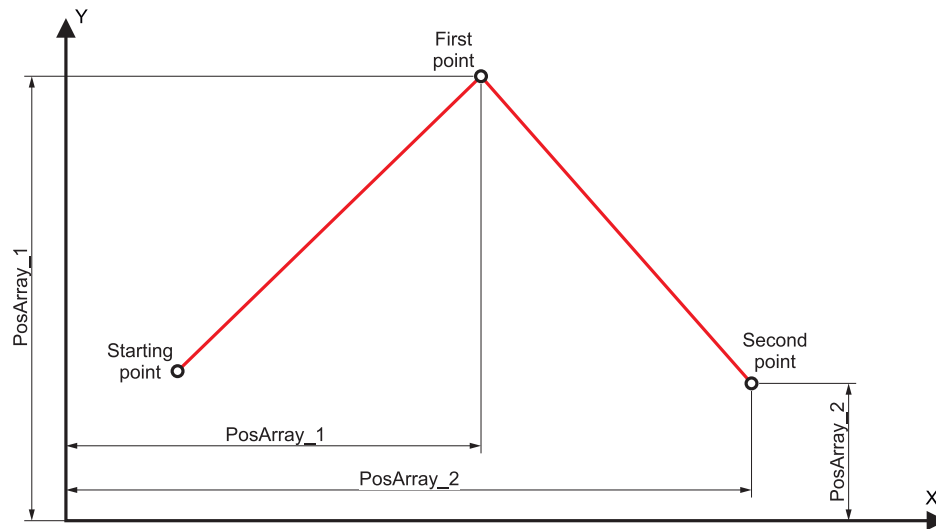
BufferMode	Buffering mode	long
1	Aborting	
2	Buffered	
3	Blending low	
4	Blending high	
5	Blending previous	
6	Blending next	
TransitionMode	Transition mode in blending mode	long
1	TMNone	
2	TMStartVelocity	
3	TMConstantVelocity	
4	TMCornerDistance	
5	TMMaxCornerDeviation	
11	Smooth	
TransitionParameter	Parametr for transition (depends on transition mode)	double

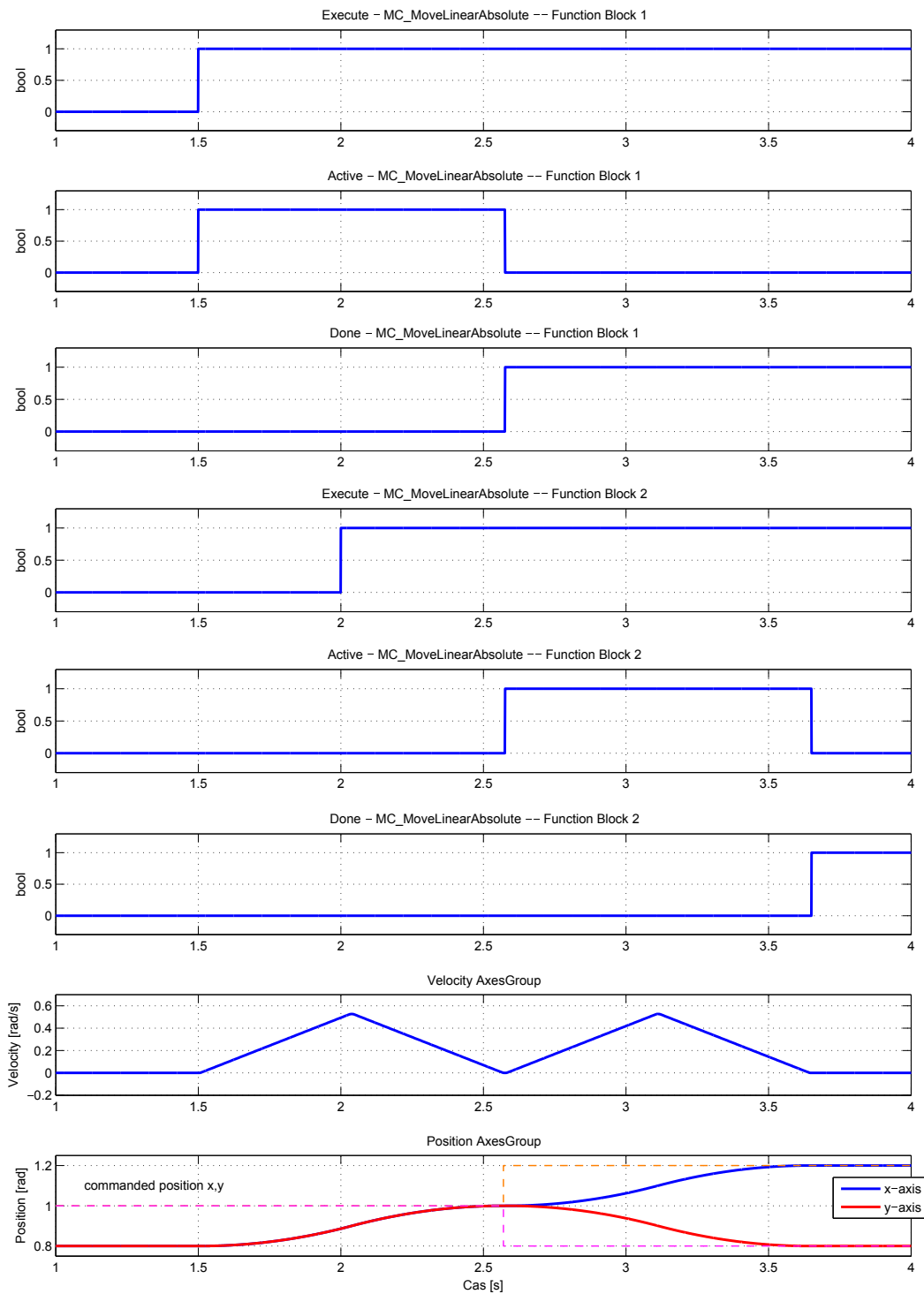
Outputs

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&gt;Execute)

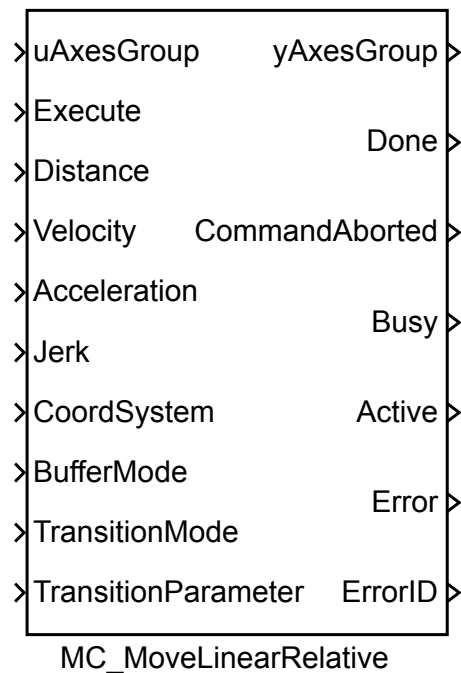




## MC\_MoveLinearRelative – Linear move to position (relative to execution point)

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.

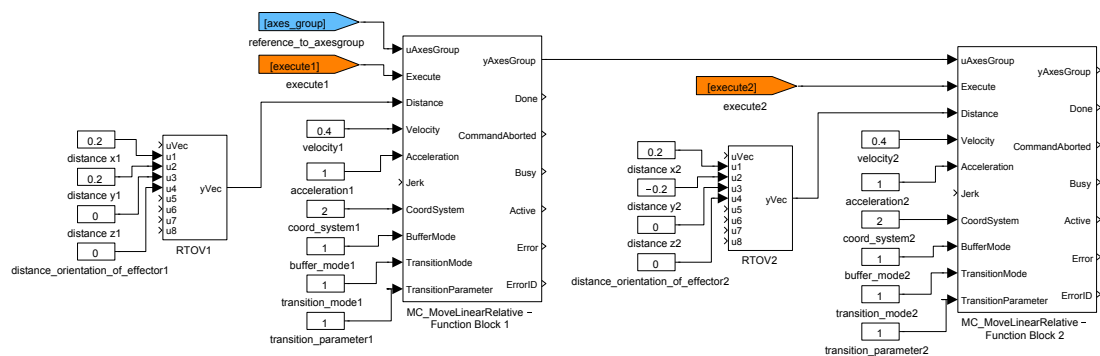
### Inputs

<b>uAxesGroup</b>	Axes group reference	<b>reference</b>
<b>Execute</b>	The block is activated on rising edge	<b>bool</b>
<b>Distance</b>	Array of coordinates (relative distances and orientations)	<b>reference</b>
<b>Velocity</b>	Maximal allowed velocity [unit/s]	<b>double</b>
<b>Acceleration</b>	Maximal allowed acceleration [unit/s <sup>2</sup> ]	<b>double</b>
<b>Jerk</b>	Maximal allowed jerk [unit/s <sup>3</sup> ]	<b>double</b>
<b>CoordSystem</b>	Reference to the coordinate system used	<b>long</b>
	1 ..... ACS	
	2 ..... MCS	
	3 ..... PCS	

<b>BufferMode</b>	Buffering mode	long
1	..... Aborting	
2	..... Buffered	
3	..... Blending low	
4	..... Blending high	
5	..... Blending previous	
6	..... Blending next	
<b>TransitionMode</b>	Transition mode in blending mode	long
1	..... TMNone	
2	..... TMStartVelocity	
3	..... TMConstantVelocity	
4	..... TMCornerDistance	
5	..... TMMaxCornerDeviation	
11	.... Smooth	
<b>TransitionParameter</b>	Parametr for transition (depends on transition mode)	double

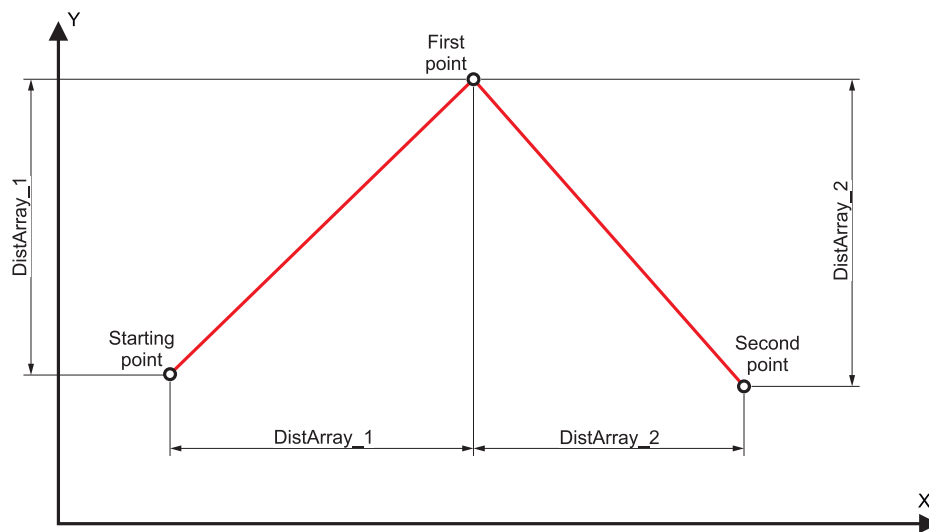
## Outputs

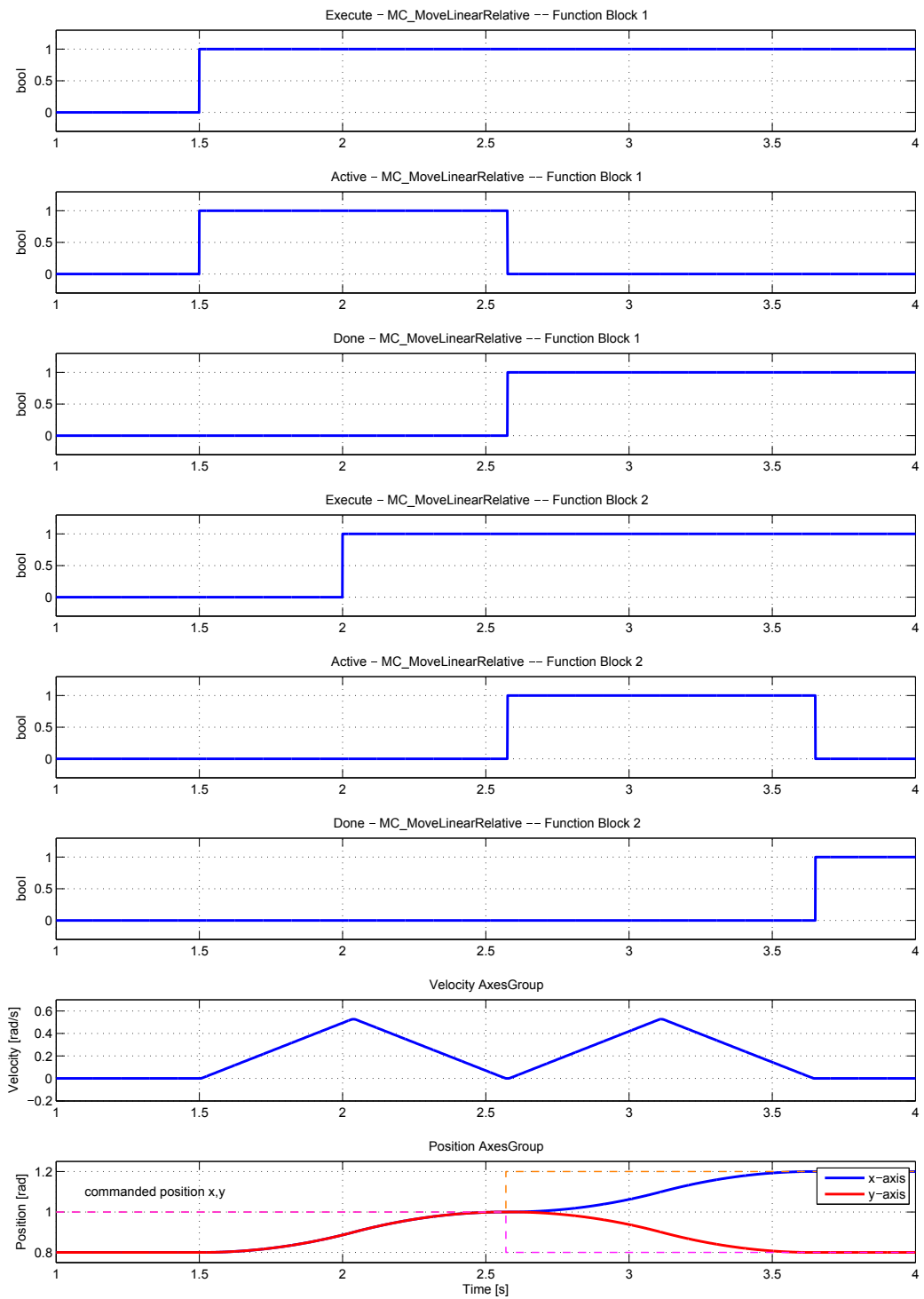
<b>yAxesGroup</b>	Axes group reference	reference
<b>Done</b>	Algorithm finished	bool
<b>CommandAborted</b>	Algorithm was aborted	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Active</b>	The block is controlling the axis	bool
<b>Error</b>	Error occurred	bool
<b>ErrorID</b>	Result of the last operation	error
i	..... REXYGEN general error	





Sequence of two complete motions (Done&gt;Execute)

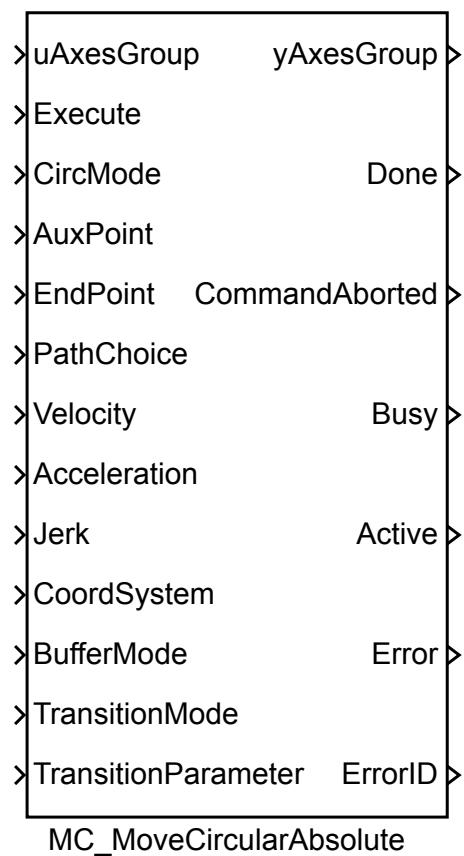




## MC\_MoveCircularAbsolute – Circular move to position (absolute coordinates)

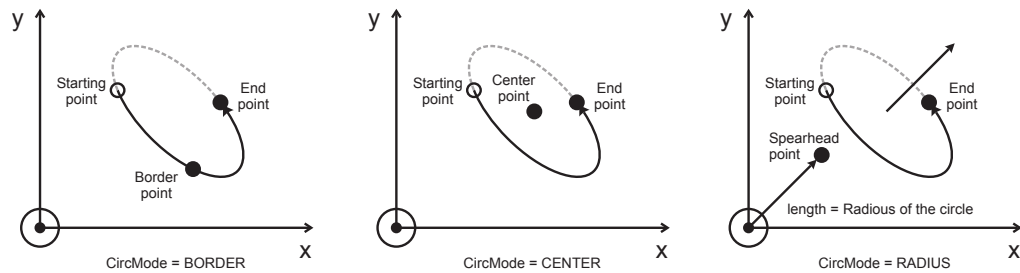
Block Symbol

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.

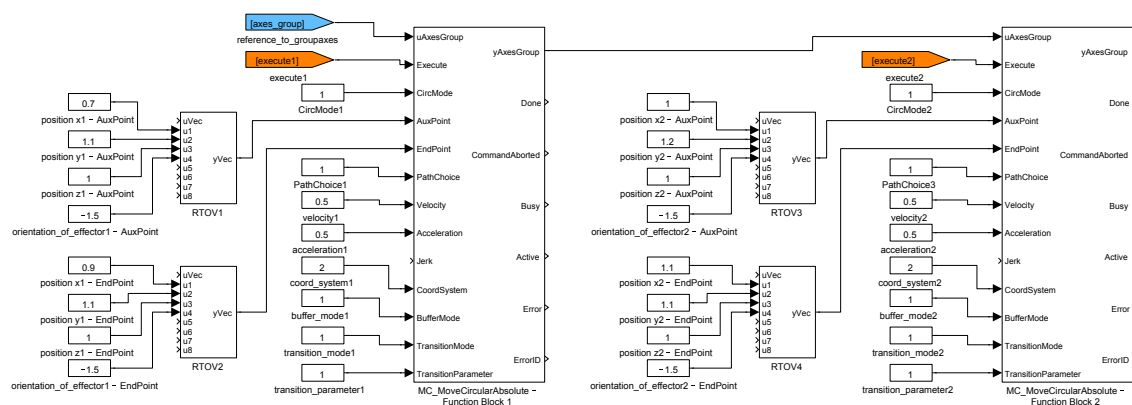


## Inputs

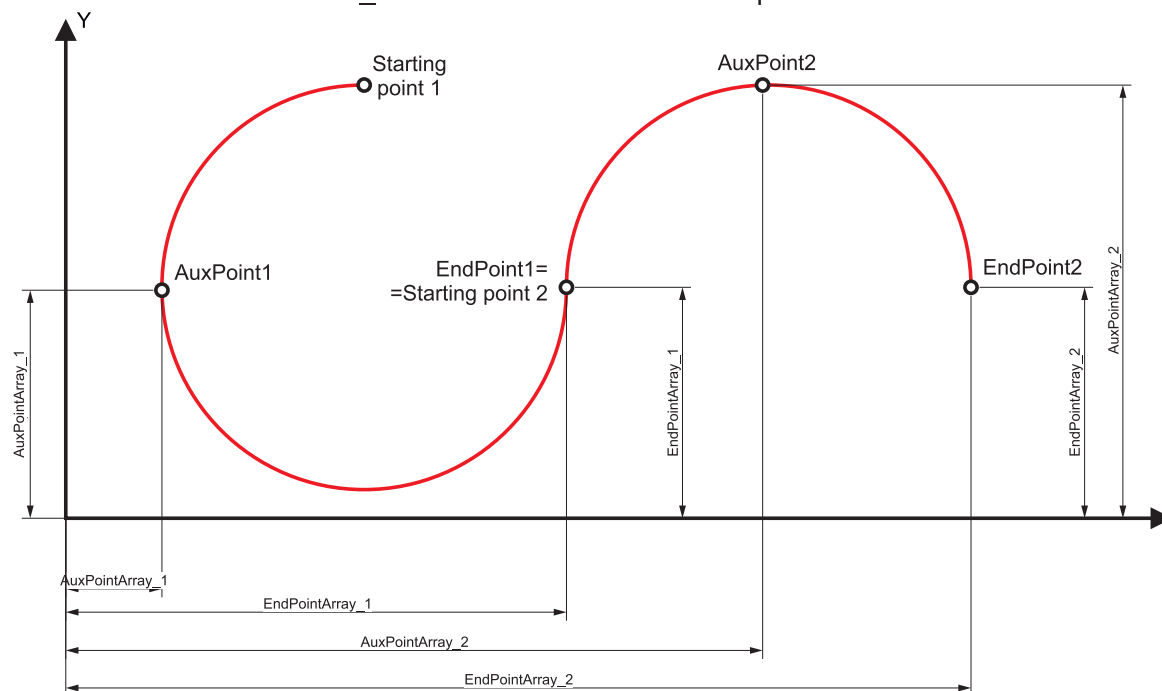
<b>uAxesGroup</b>	Axes group reference	reference
<b>Execute</b>	The block is activated on rising edge	bool
<b>CircMode</b>	Specifies the meaning of the input signals AuxPoint and CircDirection	long
	1 ..... BORDER	
	2 ..... CENTER	
	3 ..... RADIUS	
<b>AuxPoint</b>	Next coordinates to define circle (depend on CircMode)	reference
<b>EndPoint</b>	Target axes coordinates position	reference
<b>PathChoice</b>	Choice of path	long
	1 ..... Clockwise	
	2 ..... CounterClockwise	
<b>Velocity</b>	Maximal allowed velocity [unit/s]	double
<b>Acceleration</b>	Maximal allowed acceleration [unit/s <sup>2</sup> ]	double
<b>Jerk</b>	Maximal allowed jerk [unit/s <sup>3</sup> ]	double
<b>CoordSystem</b>	Reference to the coordinate system used	long
	1 ..... ACS	
	2 ..... MCS	
	3 ..... PCS	
<b>BufferMode</b>	Buffering mode	long
	1 ..... Aborting	
	2 ..... Buffered	
	3 ..... Blending low	
	4 ..... Blending high	
	5 ..... Blending previous	
	6 ..... Blending next	
<b>TransitionMode</b>	Transition mode in blending mode	long
	1 ..... TMNone	
	2 ..... TMStartVelocity	
	3 ..... TMConstantVelocity	
	4 ..... TMCornerDistance	
	5 ..... TMMaxCornerDeviation	
	11 .... Smooth	
<b>TransitionParameter</b>	Parametr for transition (depends on transition mode)	double

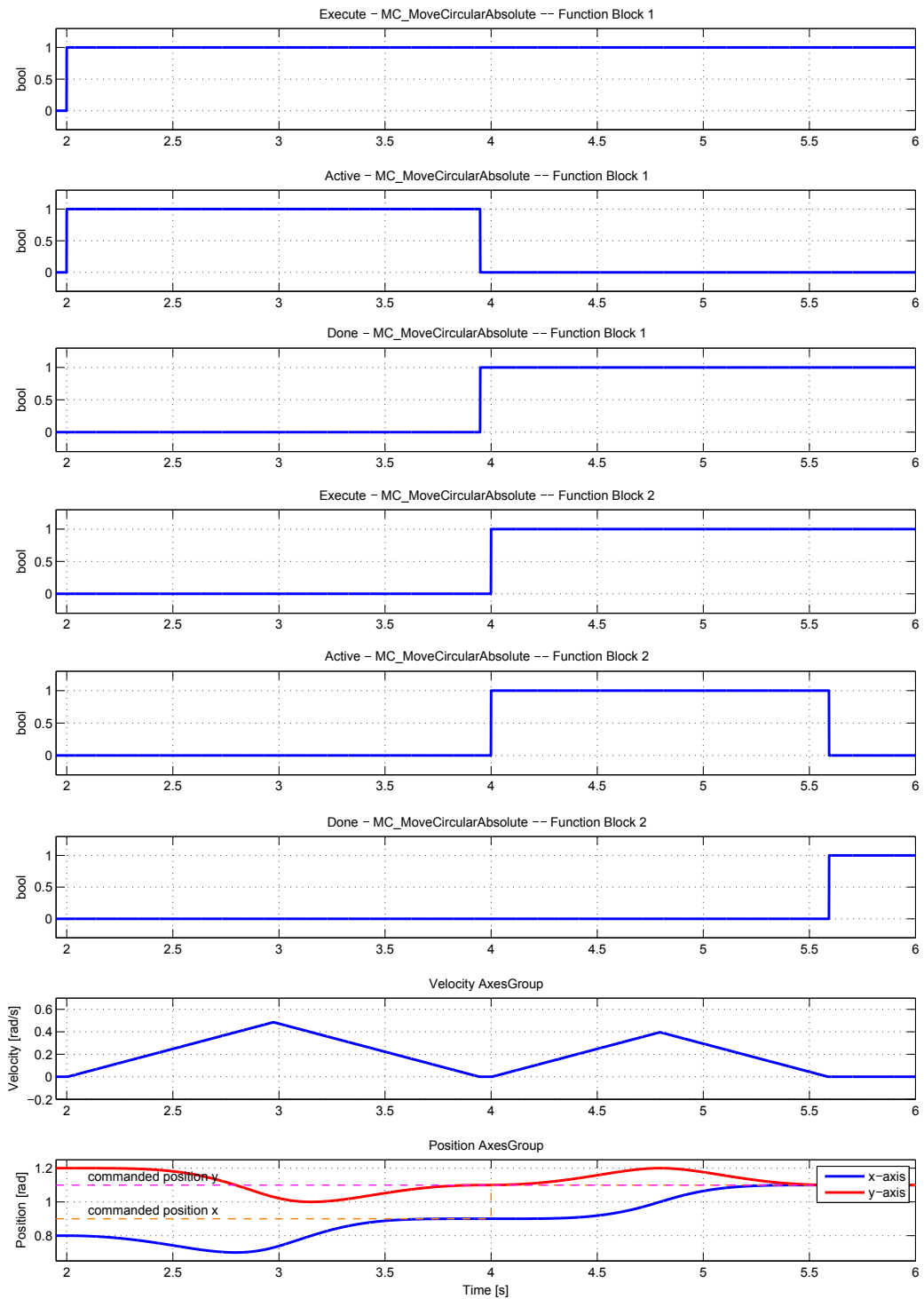
## Outputs

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

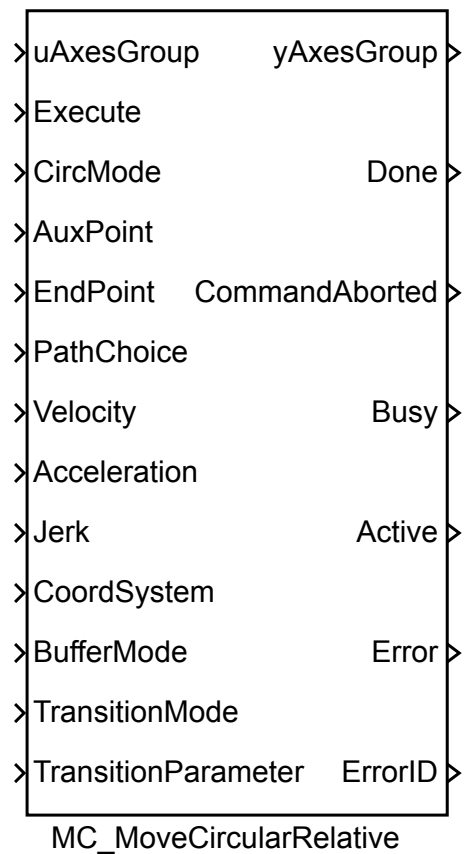




## MC\_MoveCircularRelative – Circular move to position (relative to execution point)

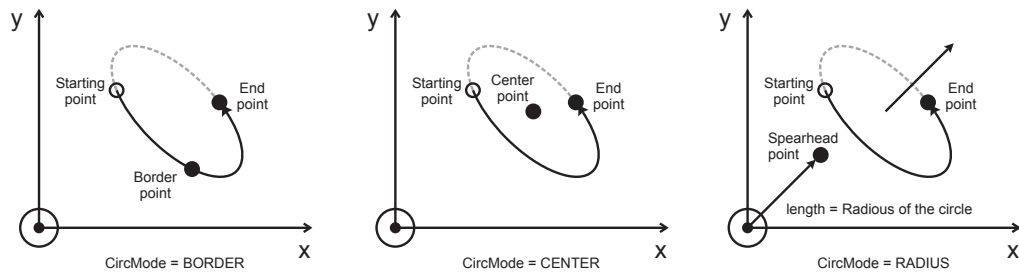
Block Symbol

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.



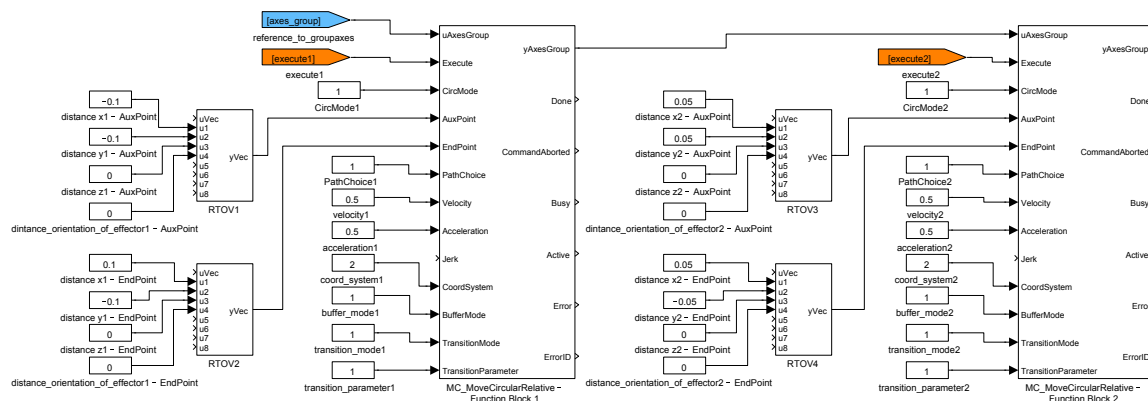
## Inputs

<b>uAxesGroup</b>	Axes group reference	reference
<b>Execute</b>	The block is activated on rising edge	bool
<b>CircMode</b>	Specifies the meaning of the input signals AuxPoint and CircDirection	long
	1 ..... BORDER	
	2 ..... CENTER	
	3 ..... RADIUS	
<b>AuxPoint</b>	Next coordinates to define circle (depend on CircMode)	reference
<b>EndPoint</b>	Target axes coordinates position	reference
<b>PathChoice</b>	Choice of path	long
	1 ..... Clockwise	
	2 ..... CounterClockwise	
<b>Velocity</b>	Maximal allowed velocity [unit/s]	double
<b>Acceleration</b>	Maximal allowed acceleration [unit/s <sup>2</sup> ]	double
<b>Jerk</b>	Maximal allowed jerk [unit/s <sup>3</sup> ]	double
<b>CoordSystem</b>	Reference to the coordinate system used	long
	1 ..... ACS	
	2 ..... MCS	
	3 ..... PCS	
<b>BufferMode</b>	Buffering mode	long
	1 ..... Aborting	
	2 ..... Buffered	
	3 ..... Blending low	
	4 ..... Blending high	
	5 ..... Blending previous	
	6 ..... Blending next	
<b>TransitionMode</b>	Transition mode in blending mode	long
	1 ..... TMNone	
	2 ..... TMStartVelocity	
	3 ..... TMConstantVelocity	
	4 ..... TMCornerDistance	
	5 ..... TMMaxCornerDeviation	
	11 .... Smooth	
<b>TransitionParameter</b>	Parametr for transition (depends on transition mode)	double

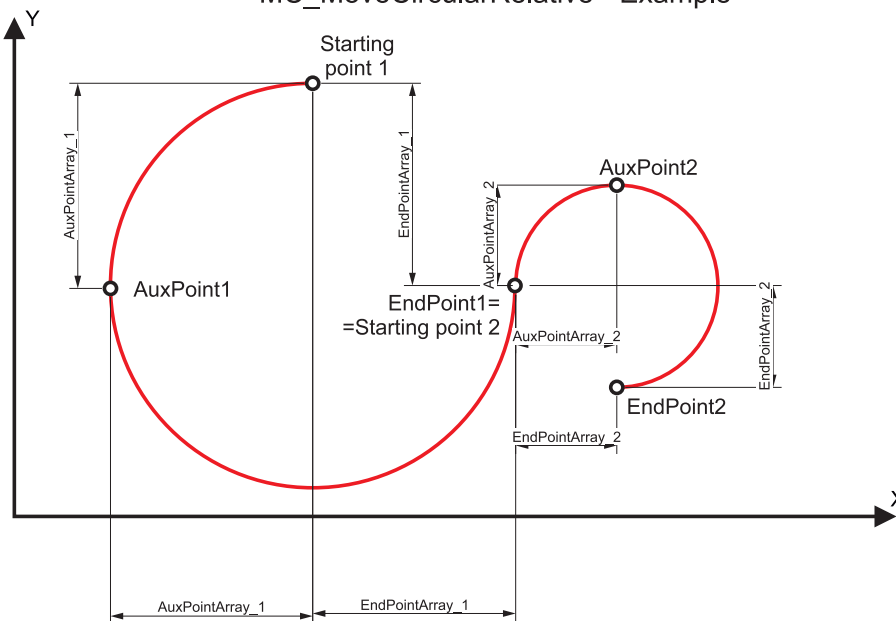


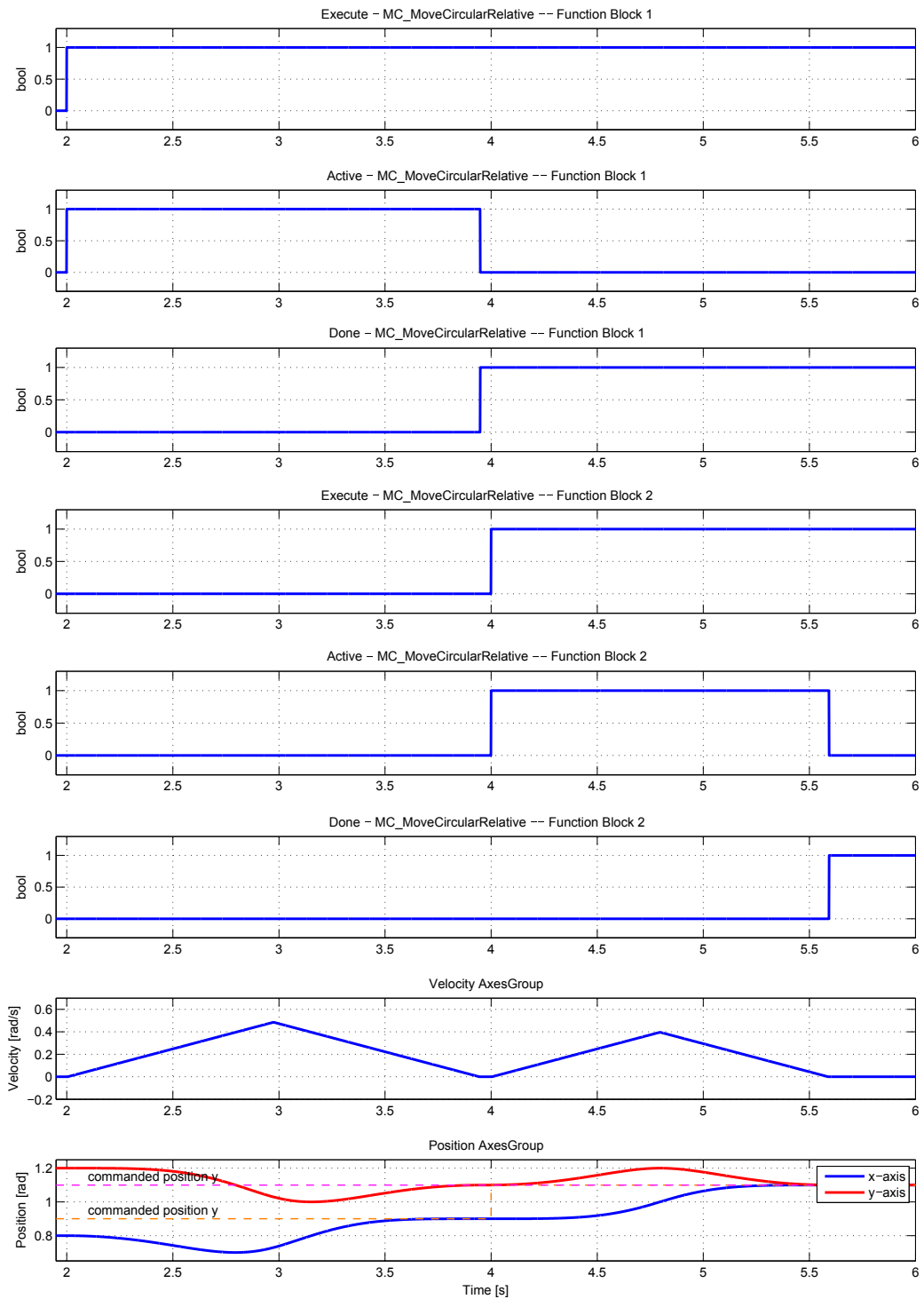
## Outputs

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\_MoveCircularRelative - Example

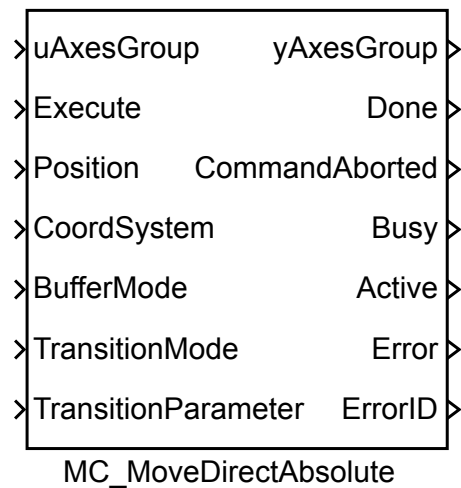




## MC\_MoveDirectAbsolute – Direct move to position (absolute coordinates)

Block Symbol

Licence: [COORDINATED MOTION](#)



### Function Description

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

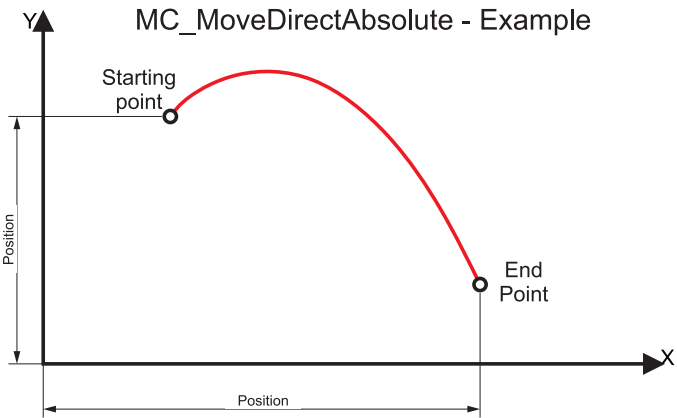
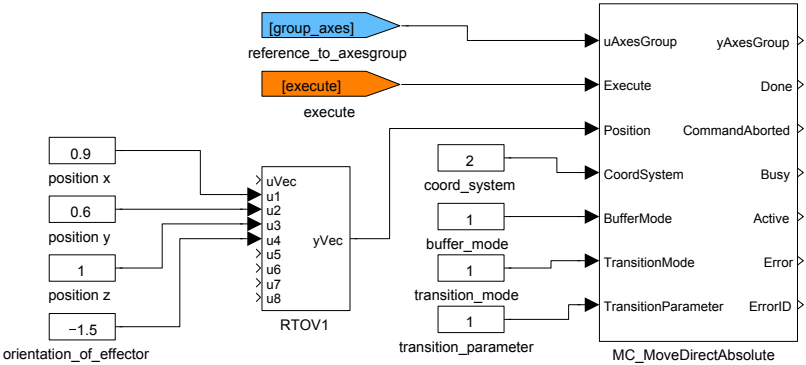
### Inputs

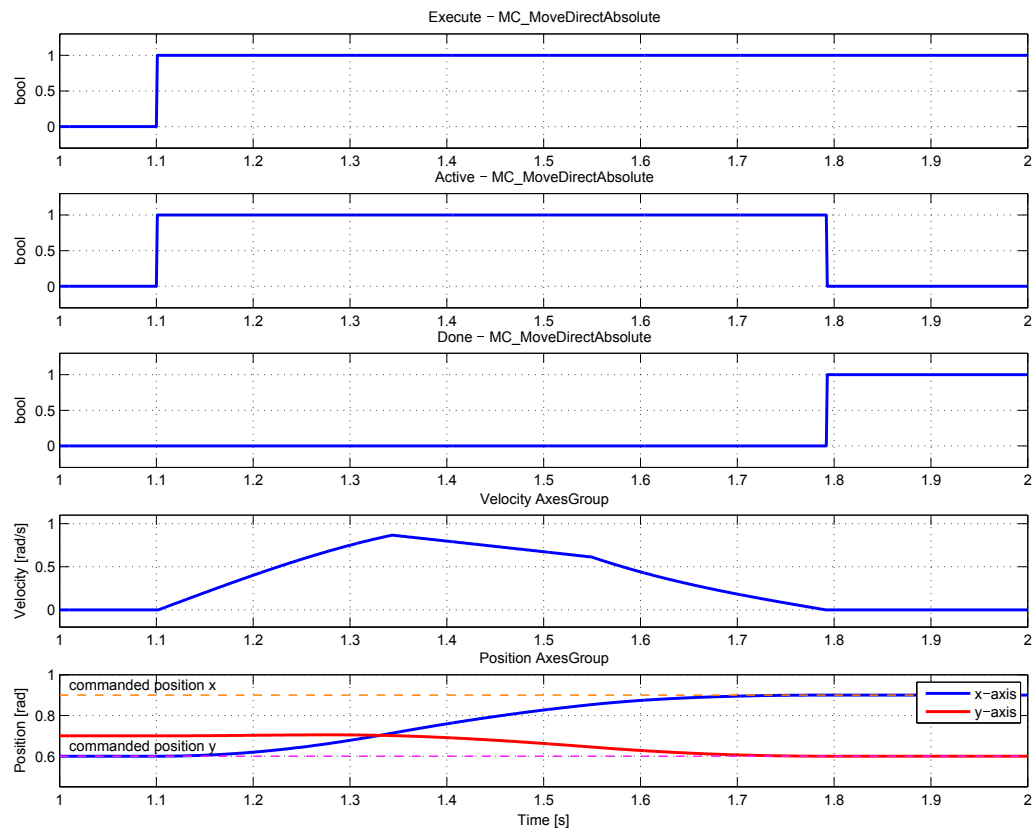
<b>uAxesGroup</b>	Axes group reference	reference
<b>Execute</b>	The block is activated on rising edge	bool
<b>Position</b>	Array of coordinates (positions and orientations)	reference
<b>CoordSystem</b>	Reference to the coordinate system used	long
	1 ..... ACS	
	2 ..... MCS	
	3 ..... PCS	
<b>BufferMode</b>	Buffering mode	long
	1 ..... Aborting	
	2 ..... Buffered	
	3 ..... Blending low	
	4 ..... Blending high	
	5 ..... Blending previous	
	6 ..... Blending next	

TransitionMode	Transition mode in blending mode	long
1	..... TMNone	
2	..... TMStartVelocity	
3	..... TMConstantVelocity	
4	..... TMCornerDistance	
5	..... TMMaxCornerDeviation	
11	..... Smooth	
TransitionParameter	Parametr for transition (depends on transition mode)	double

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	

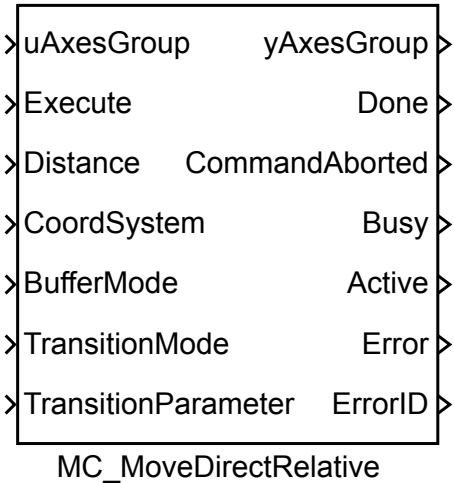




MC\_MoveDirectRelative – **Direct move to position (relative to execution point)**

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.

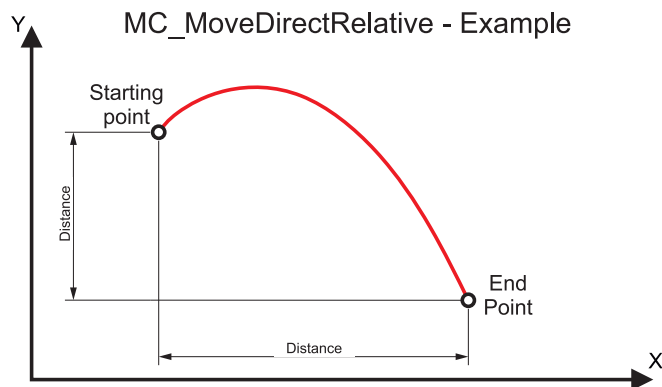
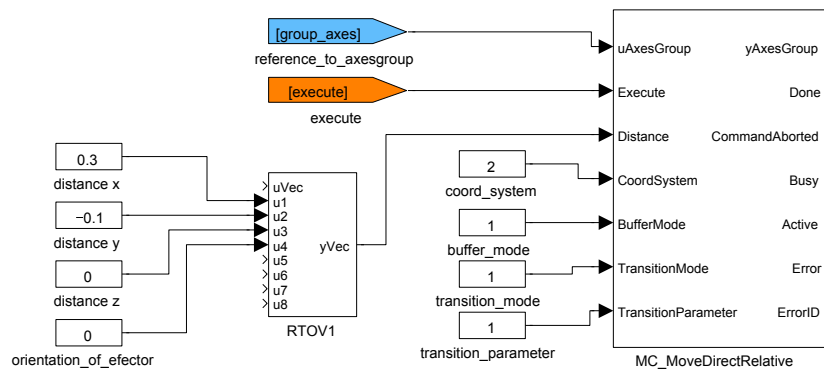
Inputs

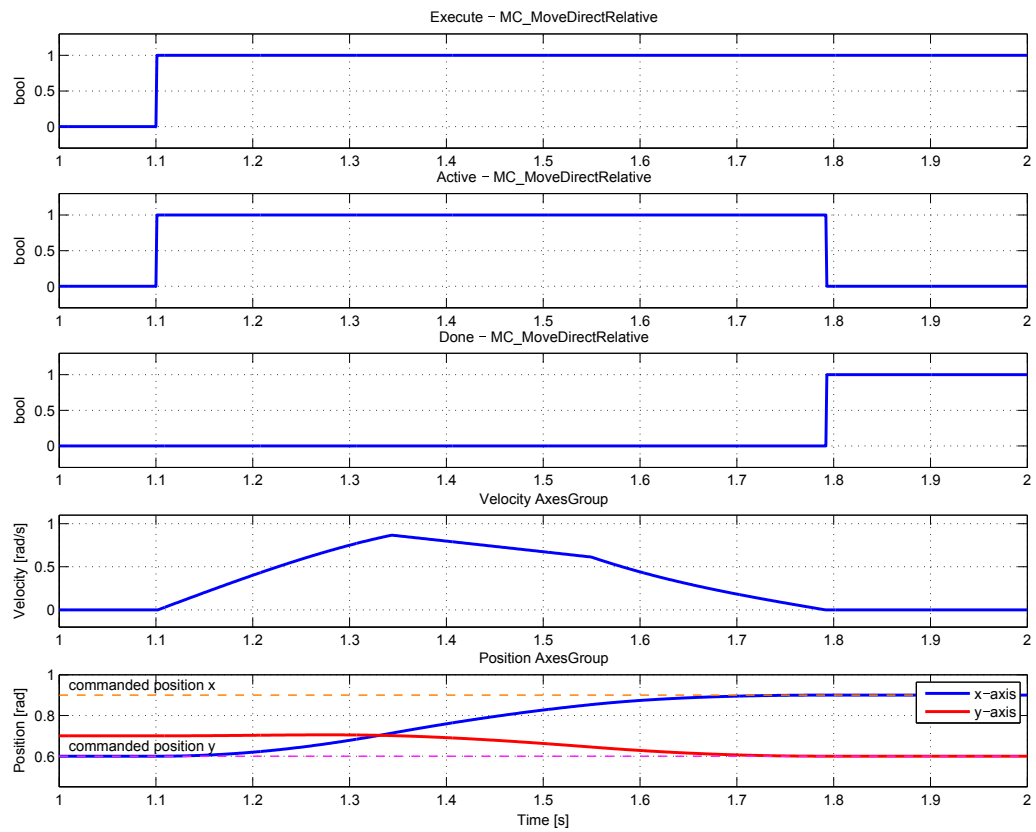
uAxesGroup	Axes group reference	reference
Execute	The block is activated on rising edge	bool
Distance	Array of coordinates (relative distances and orientations)	reference
CoordSystem	Reference to the coordinate system used	long
	1 ..... ACS	
	2 ..... MCS	
	3 ..... PCS	
BufferMode	Buffering mode	long
	1 ..... Aborting	
	2 ..... Buffered	
	3 ..... Blending low	
	4 ..... Blending high	
	5 ..... Blending previous	
	6 ..... Blending next	

<b>TransitionMode</b>	Transition mode in blending mode	long
1	..... TMNone	
2	..... TMStartVelocity	
3	..... TMConstantVelocity	
4	..... TMCornerDistance	
5	..... TMMaxCornerDeviation	
11	..... Smooth	
<b>TransitionParameter</b>	Parametr for transition (depends on transition mode)	double

## Outputs

<b>yAxesGroup</b>	Axes group reference	reference
<b>Done</b>	Algorithm finished	bool
<b>CommandAborted</b>	Algorithm was aborted	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Active</b>	The block is controlling the axis	bool
<b>Error</b>	Error occurred	bool
<b>ErrorID</b>	Result of the last operation	error
i	..... REXYGEN general error	



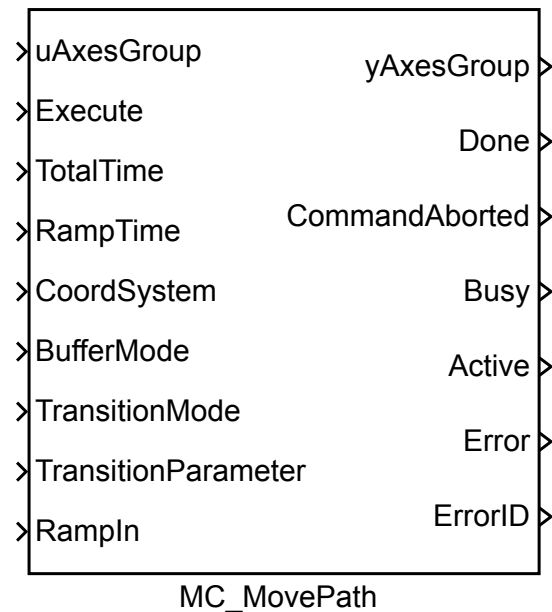




## MC\_MovePath – General spatial trajectory generation

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

<b>uAxesGroup</b>	Axes group reference	<b>reference</b>
<b>Execute</b>	The block is activated on rising edge	<b>bool</b>
<b>TotalTime</b>	Time [s] for whole move	<b>double</b>
<b>RampTime</b>	Time [s] for acceleration/deceleration	<b>double</b>
<b>CoordSystem</b>	Reference to the coordinate system used	<b>long</b>
	1 ..... ACS	
	2 ..... MCS	
	3 ..... PCS	

<b>BufferMode</b>	Buffering mode	long
1	..... Aborting	
2	..... Buffered	
3	..... Blending low	
4	..... Blending high	
5	..... Blending previous	
6	..... Blending next	
<b>TransitionMode</b>	Transition mode in blending mode	long
1	..... TMNone	
2	..... TMStartVelocity	
3	..... TMConstantVelocity	
4	..... TMCornerDistance	
5	..... TMMaxCornerDeviation	
11	.... Smooth	
<b>TransitionParameter</b>	Parametr for transition (depends on transition mode)	double
<b>RampIn</b>	RampIn factor (0 = RampIn mode not used)	double

## Parameters

<b>pc</b>	Control-points matrix	double
	$\odot[0.0 \ 1.0 \ 2.0; 0.0 \ 1.0 \ 1.0; 0.0 \ 1.0 \ 0.0]$	
<b>pk</b>	Knot-points vector	double
	$\odot[0.0 \ 0.0 \ 0.0 \ 0.0 \ 0.5 \ 1.0 \ 1.0]$	
<b>pw</b>	Weighting vector	double
	$\odot[1.0 \ 1.0 \ 1.0]$	
<b>pv</b>	Polynoms for feedrate definition	double
	$\odot[0.0 \ 0.05 \ 0.95; 0.0 \ 0.1 \ 0.1; 0.0 \ 0.0 \ 0.0; 0.1 \ 0.0 \ -0.1; -0.05 \ 0.0 \ 0.05; 0.0 \ 0.0 \ 0.0]$	
<b>pt</b>	Knot-points (time [s]) for feedrate	double
	$\odot[0.0 \ 1.0 \ 10.0 \ 11.0]$	
<b>user</b>	Only for special edit	double
	$\odot[0.0 \ 1.0 \ 2.0 \ 3.0]$	

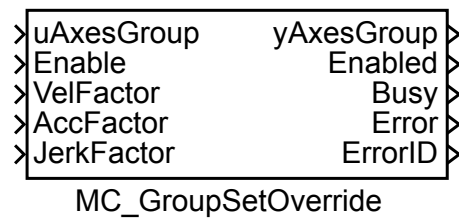
## Outputs

<b>yAxesGroup</b>	Axes group reference	reference
<b>Done</b>	Algorithm finished	bool
<b>CommandAborted</b>	Algorithm was aborted	bool
<b>Busy</b>	Algorithm not finished yet	bool
<b>Active</b>	The block is controlling the axis	bool
<b>Error</b>	Error occurred	bool
<b>ErrorID</b>	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

<b>uAxesGroup</b>	Axes group reference	<b>reference</b>
<b>Enable</b>	Block function is enabled	<b>bool</b>
<b>VelFactor</b>	Velocity multiplication factor	<b>double</b>
<b>AccFactor</b>	Acceleration/deceleration multiplication factor	<b>double</b>
<b>JerkFactor</b>	Jerk multiplication factor	<b>double</b>

### Parameter

<b>diff</b>	Deadband (difference for recalculation)	<b>⊙0.05</b>	<b>double</b>
-------------	---	--------------	---------------

### Outputs

<b>yAxesGroup</b>	Axes group reference	<b>reference</b>
<b>Enabled</b>	Signal that the override faktor are set successfully	<b>bool</b>
<b>Busy</b>	Algorithm not finished yet	<b>bool</b>
<b>Error</b>	Error occurred	<b>bool</b>
<b>ErrorID</b>	Result of the last operation	<b>error</b>
	i . . . . . REXYGEN general error	



# 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	•	
AND_	•	
ANDHEXD	•	
ANDOCT	•	
ANDQUAD	•	
ANLS	•	
ARC	•	
ARLY	•	
ASW		ADVANCED

*The list continues on the next page...*

Function block name	Licensing group	
	STANDARD	Other
ATMT	•	
AVG	•	
AVS		ADVANCED
BDHEXD	•	
BDOCT	•	
BINS	•	
BIS	•	
BITOP	•	
BMHEXD	•	
BMOCT	•	
BPF	•	
CDELSSM		ADVANCED
CMP	•	
CNA	•	
CNB	•	
CNDR	•	
CNE	•	
CNI	•	
CNR	•	
CNS	•	
CONCAT	•	
COUNT	•	
CSSM		ADVANCED
DATE_	•	
DATETIME	•	
DDELSSM		ADVANCED
DEL	•	
DELM	•	
DER	•	
DIF_	•	
Display	•	
DIV	•	
DSSM		ADVANCED
EAS	•	
EATMT		ADVANCED
EDGE_	•	
EMD	•	
EPC		ADVANCED
EVAR	•	

*The list continues on the next page...*



Function block name	Licensing group	
	STANDARD	Other
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	•	
INSTD	•	
INTE	•	
INTSM	•	
IODRV	•	
IOTASK	•	
ISSW	•	
ISUB	•	
ITOI	•	
ITOS	•	

*The list continues on the next page...*

Function block name	Licensing group	
	STANDARD	Other
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
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

*The list continues on the next page...*

Function block name	Licensing group	
	STANDARD	Other
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
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

*The list continues on the next page...*

Function block name	Licensing group	
	STANDARD	Other
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_SetOverride		MOTION CONTROL
MCP_Stop		MOTION CONTROL
MCP_TorqueControl		MOTION CONTROL
MCP_VelocityProfile		MOTION CONTROL
MCU	•	
MDL	•	
MDLI	•	
MID	•	
MINMAX	•	
MODULE	•	
MP	•	
MUL	•	
MVD	•	
NOT_	•	
NSCL	•	
OR_	•	
ORHEXD	•	

*The list continues on the next page...*

Function block name	Licensing group	
	STANDARD	Other
OROCT	•	
ORQUAD	•	
OSCALL	•	
OUTHEXD	•	
OUTOCT	•	
Outport	•	
OUTQUAD	•	
OUTRHEXD		ADVANCED
OUTROCT		ADVANCED
OUTRQUAD		ADVANCED
OUTRSTD		ADVANCED
OUTSTD	•	
PARA	•	
PARB	•	
PARI	•	
PARR	•	
PARS	•	
PIDAT		AUTOTUNING
PIDE		ADVANCED
PIDGS		ADVANCED
PIDMA		AUTOTUNING
PIDU	•	
PIDUI		ADVANCED
PJROCT	•	
PJSOCT	•	
POL	•	
POUT	•	
PRBS	•	
PRGM	•	
PROJECT	•	
PSMPC		ADVANCED
PWM	•	
QFC		ADVANCED
QFD		ADVANCED
QTASK	•	
RDC		ADVANCED
REC	•	
REGEXP		ADVANCED
REL	•	

*The list continues on the next page...*

Function block name	Licensing group	
	STANDARD	Other
REPLACE	•	
REXLANG		REXLANG
RLIM	•	
RLY	•	
RM_AxesGroup		COORDINATED MOTION
RM_Axis		MOTION CONTROL
RM_AxisOut		MOTION CONTROL
RM_AxisSpline		MOTION CONTROL
RM_Feed		COORDINATED MOTION
RM_Gcode		COORDINATED MOTION
RM_GroupTrack		COORDINATED MOTION
RM_Track		MOTION CONTROL
RS	•	
RTOI	•	
RTOS	•	
RTOV	•	
S10F2		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	•	

*The list continues on the next page...*

Function block name	Licensing group	
	STANDARD	Other
SILOS	•	
SINT	•	
SLEEP	•	
SMHCC		ADVANCED
SMHCCA		AUTOTUNING
SMTF		ADVANCED
SOPDT	•	
SPIKE		ADVANCED
SQR	•	
SQRT_	•	
SR	•	
SRTF		ADVANCED
SSW	•	
STOR	•	
SUB	•	
SubSystem	•	
SWR	•	
SWU	•	
SWVMR	•	
TASK	•	
TIME	•	
TIMER_	•	
TIODRV	•	
TRND	•	
TRNDV	•	
TSE	•	
VDEL	•	
VIN		ADVANCED
VOUT		ADVANCED
VTOR	•	
WSCH	•	
WWW	•	
ZV4IS		ADVANCED





## 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

### 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

#### Class registration, symbol and validation error codes

- 200 .... Class not registered
- 201 .... Class already registered
- 202 .... Not enough space for registry
- 203 .... Registry index out of range
- 204 .... Invalid context
- 205 .... Invalid identifier
- 206 .... Invalid input flag
- 207 .... Invalid input mask
- 208 .... Invalid object type
- 209 .... Invalid variable type
- 210 .... Invalid object workspace
- 211 .... Symbol not found
- 212 .... Symbol is ambiguous
- 213 .... Range check error
- 214 .... Not enough search space
- 215 .... Write to read-only variable denied
- 216 .... Data not ready
- 217 .... Value out of range
- 218 .... Input connection error
- 219 .... Loop of type UNKNOWN detected
- 220 .... REXLANG compilation error

## Stream and file system codes

- 300 .... Stream overflow
- 301 .... Stream underflow
- 302 .... Stream send error
- 303 .... Stream receive error
- 304 .... Stream download error
- 305 .... Stream upload error
- 306 .... File creation error
- 307 .... File open error
- 308 .... File close error
- 309 .... File read error
- 310 .... File write error
- 311 .... Invalid format
- 312 .... Unable to compress files
- 313 .... Unable to extract files

## Communication errors

- 400 .... Network communication failure
- 401 .... Communication already initialized
- 402 .... Communication finished successfully
- 403 .... Communication closed unexpectedly
- 404 .... Unknown command
- 405 .... Unexpected command
- 406 .... Communication closed unexpectedly, probably 'Too many clients'
- 407 .... Communication timeout
- 408 .... Target device not found
- 409 .... Link failed
- 410 .... 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 block
- 445 .... Invalid operation
- 446 .... Communication closed
- 447 .... Connection cancelled

#### Numerical error codes

- 500 .... General numeric error
- 501 .... Division by zero
- 502 .... Numeric stack overflow
- 503 .... Invalid numeric instruction
- 504 .... Invalid numeric address
- 505 .... Invalid numeric type
- 506 .... Not initialized numeric value
- 507 .... Numeric argument overflow/underflow
- 508 .... Numeric range check error
- 509 .... Invalid subvector/submatrix range
- 510 .... Numeric value too close to zero

#### Archive system codes

- 600 .... Archive seek underflow
- 601 .... Archive semaphore fatal error
- 602 .... Archive cleared
- 603 .... Archive reconstructed from saved vars
- 604 .... Archive reconstructed from normal vars
- 605 .... Archive check sum error
- 606 .... Archive integrity error

- 607 .... Archive sizes changed
- 608 .... Maximum size of disk archive file exceeded

### Motion control codes

- 700 .... MC - Invalid parameter
- 701 .... MC - Out of range
- 702 .... MC - Position not reachable
- 703 .... MC - Invalid axis state
- 704 .... MC - Torque limit exceeded
- 705 .... MC - Time limit exceeded
- 706 .... MC - Distance limit exceeded
- 707 .... MC - Step change in position or velocity
- 708 .... MC - Base axis error or invalid state
- 709 .... MC - Stopped by HALT input
- 710 .... MC - Stopped by POSITION limit
- 711 .... MC - Stopped by VELOCITY limit
- 712 .... MC - Stopped by ACCELERATION limit
- 713 .... MC - Stopped by LIMITSWITCH
- 714 .... MC - Stopped by position LAG
- 715 .... MC - Axis disabled during motion
- 716 .... MC - Transition failed
- 717 .... MC - Not used
- 718 .... MC - Not used
- 719 .... MC - Not used
- 720 .... MC - General failure
- 721 .... MC - Not implemented
- 722 .... MC - Command is aborted
- 723 .... MC - Conflict in block and axis periods
- 724 .... MC - Busy, waiting for activation

### Licensing codes

- 800 .... Unable to identify Ethernet interface
- 801 .... Unable to identify CPU
- 802 .... Unable to identify HDD
- 803 .... Invalid device code
- 804 .... Invalid licensing key
- 805 .... Not licensed

### Webserver-related errors

- 900 .... Web request too large
- 901 .... Web reply too large
- 902 .... Invalid format
- 903 .... Invalid parameter

### RexVision-related errors

- 1000 ... Result is not evaluated
- 1001 ... The searched object/pattern can not be found
- 1002 ... The search criterion returned more corresponding objects

### FMI standard related errors

- 1100 ... FMI Context allocation failure
- 1101 ... Invalid FMU version
- 1102 ... FMI XML parsing error
- 1103 ... FMI Model Exchange kind required
- 1104 ... FMI Co-Simulation kind required
- 1105 ... Could not create FMU loading mechanism
- 1106 ... Instantiation of FMU failed
- 1107 ... Termination of FMU failed
- 1108 ... FMU reset failed
- 1109 ... FMU Experiment setup failed
- 1110 ... Entering FMU initialization mode failed
- 1111 ... Exiting FMU initialization mode failed
- 1112 ... Error getting FMU variable list
- 1113 ... Error getting FMU real variable
- 1114 ... Error setting FMU real variable
- 1115 ... Error getting FMU integer variable
- 1116 ... Error setting FMU integer variable
- 1117 ... Error getting FMU boolean variable
- 1118 ... Error setting FMU boolean variable
- 1119 ... Doing a FMU simulation step failed
- 1120 ... FMU has too many inputs
- 1121 ... FMU has too many outputs
- 1122 ... FMU has too many parameters

# Bibliography

- [1] OPC Foundation. *Data Access Custom Interface Specification Version 3.00*. OPC Foundation, P.O. Box 140524, Austin, Texas, USA, 2003.
- [2] *Simulink reference, version 6*. The Mathworks, 3 Apple Hill Drive, Natick, MA, USA, 2006.
- [3] Schlegel Miloš. Fuzzy controller: a tutorial. *www.rexcontrols.com*.
- [4] Miloš Schlegel, Pavel Balda, and Milan Štětina. Robust PID autotuner: method of moments. *Automatizace*, 46(4):242–246, 2003.
- [5] M. Schlegel and P. Balda. Diskretizace spojitého lineárního systému (in Czech). *Automatizace*, 11, 1987.
- [6] BLAS 3.6.0. – netlib.org. Basic Linear Algebra Subprograms Version 3.6.0, 2016.
- [7] LAPACK 3.6.0 – netlib.org. Linear Algebra PACKage Version 3.6.0, 2016.
- [8] 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.





# Index

- REXYGEN Compiler
  - compiler, [31](#)
- REXYGEN Compiler compiler, [22](#)
- TODO
  - SRTF DLOG, [37](#)
- ABS\_, [69](#), [651](#)
- absolute
  - position sensor, [107](#)
- absolute value, [69](#)
- ABSR0T, [107](#), [651](#)
- ACD, [287](#), [651](#)
- ADD, [70](#), [71](#), [651](#)
- ADDHEXD, [71](#), [651](#)
- addition, [70](#)
  - extended, [79](#)
  - integer, [88](#)
- ADDOCT, [70](#), [71](#), [104](#), [651](#)
- ADDQUAD, [71](#), [651](#)
- AFLUSH, [297](#), [651](#)
- alarm
  - Boolean value, [283](#)
  - numerical value, [285](#)
- ALB, [283](#), [651](#)
- ALBI, [283](#), [651](#)
- algebraic loop, [31](#)
- ALN, [285](#), [651](#)
- ALNI, [285](#), [651](#)
- analog input
  - safety, [135](#)
- AND\_, [244](#), [245](#), [651](#)
- ANDHEXD, [245](#), [651](#)
- ANDOCT, [244](#), [245](#), [651](#)
- ANDQUAD, [245](#), [651](#)
- ANLS, [156](#), [651](#)
- application
  - of the REXYGEN System, [22](#)
- ARC, [20](#), [23](#), [284](#), [286](#), [288](#), [291](#), [293](#), [297](#), [651](#)
- architecture
  - open, [32](#)
- archiv, [282](#)
- archive, [20](#)
  - backed-up memory, [282](#)
  - configuration, [20](#)
  - disk, [282](#)
  - RAM memory, [282](#)
- archiving
  - delta criterion, [287](#)
- ARLY, [169](#), [651](#)
- ASW, [109](#), [651](#)
- ATMT, [14](#), [247](#), [255](#), [320](#), [329](#), [333](#), [652](#)
- automaton
  - finite-state, [247](#), [255](#)
- average
  - moving, [111](#)
- AVG, [111](#), [652](#)
- AVS, [14](#), [112](#), [652](#)
- band
  - frequency transmission, [113](#)
- band-pass filter, [113](#)
- bandwidth, [126](#)
- BDHEXD, [250](#), [255](#), [652](#)
- BDOCT, [250](#), [255](#), [652](#)
- Bessel filter, [126](#)
- binary number
  - transformation, [262](#)
- binary sequence
  - generator, [158](#), [160](#)
- BINS, [158](#), [652](#)
- BIS, [158](#), [160](#), [161](#), [652](#)

BITOP, 251, 652

block

- description, 15
- description format, 15
- execution, 36
- inputs, 15
- mathematic, 14
- matrix, 14
- modeling, 14
- outputs, 15
- parameters, 15
- symbol, 15
- vector, 14

blocks

- analog signal processing, 14
- data archiving, 14
- generators, 14
- input-output, 13
- logic control, 14
- parameter-related, 14
- regulation, 14
- special, 15

BMHEXD, 252, 255, 652

BMOCT, 252, 255, 652

Boolean complementation, 264

BPF, 113, 652

Butterworth filter, 126

CDELSSM, 342, 652

circuit

- flip-flop Reset-Set, 268
- flip-flop Set-Reset, 269

CMP, 114, 652

CNA, 365, 652

CNB, 73, 652

CNDR, 115, 652

CNE, 74, 652

CNI, 75, 652

CNR, 76, 652

CNS, 300, 652

coefficient

- relative damping, 113, 126

comparator, 114

compatibility

REXYGEN and Simulink, 31

compensator

- lead, 178
- lead-lag, 179
- nonlinearity, 115
- simple nonlinearity, 128

compiler

REXYGEN Compiler, 22, 31

compression, 287

CONCAT, 301, 652

conditioner

- nonlinear, 115

configuration

- REXYGEN System, 22
- archives, 22
- computation task, 22
- input-output drivers, 22
- modules, 22

constant

- Boolean, 73
- integer, 75
- logic, 73
- real, 76

control

- motion, 15
- sequential, 247, 255

control unit

- manual, 180

controller

- fuzzy logic, 170
- PID, 196
- PID with gain scheduling, 188
- PID with input-defined parameters, 199
- PID with relay autotuner, 182
- PID with static error, 185
- step, 220, 223
- with frequency autotuner, 213

conversion

- real to integer, 101

COUNT, 17, 253, 652

counter

- controlled, 253

CSSM, 345, 652

- data
  - remote connection, 455
- data storing, 289, 292
- data types, 16
- DATE\_, 274, 275, 652
- DATETIME, 274, 275, 278, 652
- DDELSSM, 348, 652
- dead time, 356, 360
- DEL, 117, 652
- delay
  - transport, 118
  - variable, 150
  - with initialization, 117
- DELM, 118, 652
- delta criterion, 287
- demultiplexer
  - bitwise, 250
- denominator, 80
- DER, 119, 652
- derivation, 119, 124
- detection
  - edge, 258
- deviation
  - standard, 121
- DIF\_, 77, 652
- difference, 77
- Display, 46, 652
- DIV, 78, 652
- division
  - extended, 80
  - integer, 94, 95
  - remainder, 95
  - two signals, 78
- DLL library, 32
- driver
  - REXYGEN System, 28
  - .rio file extension, 28
  - configuration data, 28
  - input-output, 13
  - input/output, 28
  - input/output with tasks, 41
  - user manual, 30
- drivers
  - REXYGEN system, 13
- DSSM, 350, 652
- EAS, 79, 652
- EATMT, 255, 652
- edge detection, 258
- EDGE\_, 142, 258, 652
- element
  - three state, 241
- EMD, 80, 652
- EPC, 38, 446, 652
- error
  - fatal, 34
- EVAR, 121, 652
- EXEC, 20, 22, 28–30, 32, 34, 35, 39–42, 653
- executive
  - configuration, 13, 19
  - real-time, 22
  - RexCore program, 13
- external program, 446
- feedback loop, 31
- filter
  - band-pass, 113
  - Bessel, 126
  - Butterworth, 126
  - low-pass, 126
  - moving average, 111
  - nonlinear, 146
  - spike, 146
- filtering, 119, 124
  - digital, 34
- FIND, 302, 653
- finite-state machine, 247, 255
- first order system, 360
- FLCU, 14, 170, 653
- flip-flop circuit
  - Reset-Set, 268
  - Set-Reset, 269
- FMUCS, 352
- FMUINFO, 355
- FNX, 81, 653
- FNXY, 83, 653
- FOPDT, 179, 360
- FOPDT, 356, 653

frequency transmission band, 113  
 FRID, 173, 653  
 From, 47, 50, 51, 653  
 function  
     operating system, 38  
     single variable, 81  
     two variables, 83  
 fuzzy logic, 170  
 GAIN, 85, 653  
 gain, 85  
 generator  
     binary sequence, 158, 160  
     piecewise linear function, 156  
     signal, 164  
     time function, 202  
 GETPA, 318, 653  
 GETPB, 320, 653  
 GETPI, 320, 467, 653  
 GETPR, 320, 333, 653  
 GETPS, 322, 653  
 Goto, 47–49, 51, 653  
 GotoTagVisibility, 50, 51, 653  
 GRADS, 86, 653  
 HMI, 25, 653  
 HTTP, 449, 653  
 HTTP2, 451, 653  
 hysteresis, 114  
 I3PM, 176, 653  
 IADD, 88, 653  
 identification  
     three parameter model, 176  
 IDIV, 94, 653  
 IMOD, 95, 653  
 IMUL, 92, 653  
 INFO, 27, 653  
 INHEXD, 56, 653  
 INOCT, 56, 653  
 Inport, 52, 54, 55, 653  
 INQUAD, 56, 653  
 INSTD, 47, 56, 58, 653  
 INTE, 122, 145, 653  
 integer

    division, 94  
     integer number  
         transformation, 262  
     integer signal  
         switching, 260  
     integrator  
         controlled, 122  
         simple, 145  
     interpolation  
         linear, 96  
 INTSM, 259, 261, 653  
 IODRV, 23, 28, 47, 49, 653  
 IOTASK, 30, 37, 41, 318, 320, 327, 329, 343,  
     346, 653  
 ISSW, 260, 653  
 ISUB, 90, 653  
 ITOI, 262, 653  
 ITOS, 303, 653  
 KDER, 124, 654  
 LC, 178, 654  
 least squares method, 119  
 LEN, 304, 654  
 LIN, 96, 654  
 linear  
     interpolation, 96  
 linear function  
     generator, 156  
 LLC, 179, 360, 654  
 logical sum, 265  
 loop  
     algebraic, 31  
     feedback, 31  
 low-pass filter, 126  
 LPBRK, 13, 31, 109, 654  
 LPF, 126, 654  
 LSM, 119  
 maximum, 127  
 MB\_DASUM, 366  
 MB\_DAXPY, 368  
 MB\_DCOPY, 370  
 MB\_DDOT, 372  
 MB\_DGEMM, 374

- MB\_DGEMV, 376
- MB\_DGER, 378
- MB\_DNRM2, 380
- MB\_DROT, 382
- MB\_DSCAL, 384
- MB\_DSWAP, 386
- MB\_DTRMM, 388
- MB\_DTRMV, 390
- MB\_DTRSV, 392
- MC\_AccelerationProfile, 486, 521, 522, 542, 543, 654
- MC\_AddAxisToGroup, 595, 654
- MC\_CamIn, 556, 563, 569, 579, 582, 654
- MC\_CamOut, 556, 560, 654
- MC\_CombineAxes, 564, 654
- MC\_GearIn, 568, 569, 572, 577, 579, 582, 654
- MC\_GearInPos, 571, 654
- MC\_GearOut, 577, 654
- MC\_GroupContinue, 616, 618, 654
- MC\_GroupDisable, 598, 654
- MC\_GroupEnable, 597, 654
- MC\_GroupHalt, 611, 654
- MC\_GroupInterrupt, 616, 618, 654
- MC\_GroupReadActualAcceleration, 607, 654
- MC\_GroupReadActualPosition, 605, 654
- MC\_GroupReadActualVelocity, 606, 654
- MC\_GroupReadError, 621, 654
- MC\_GroupReadStatus, 619, 654
- MC\_GroupReset, 622, 654
- MC\_GroupSetOverride, 647, 654
- MC\_GroupSetPosition, 603, 654
- MC\_GroupStop, 608, 654
- MC\_Halt, 490, 654
- MC\_HaltSuperimposed, 492, 654
- MC\_Home, 494, 527, 654
- MC\_MoveAbsolute, 496, 510, 535, 553, 572, 654
- MC\_MoveAdditive, 497, 500, 654
- MC\_MoveCircularAbsolute, 631, 654
- MC\_MoveCircularRelative, 635, 654
- MC\_MoveContinuousAbsolute, 509, 654
- MC\_MoveContinuousRelative, 513, 654
- MC\_MoveDirectAbsolute, 639, 654
- MC\_MoveDirectRelative, 642, 655
- MC\_MoveLinearAbsolute, 623, 655
- MC\_MoveLinearRelative, 627, 655
- MC\_MovePath, 645, 655
- MC\_MovePath\_PH, 655
- MC\_MoveRelative, 497, 503, 506, 514, 655
- MC\_MoveSuperimposed, 497, 506, 579, 582, 655
- MC\_MoveVelocity, 479, 517, 553, 655
- MC\_PhasingAbsolute, 579, 655
- MC\_PhasingRelative, 582, 655
- MC\_PositionProfile, 486, 487, 521, 542, 543, 553, 562, 655
- MC\_Power, 525, 655
- MC\_ReadActualPosition, 527, 655
- MC\_ReadAxisError, 528, 655
- MC\_ReadBoolParameter, 529, 655
- MC\_ReadCartesianTransform, 601, 655
- MC\_ReadParameter, 530, 655
- MC\_ReadStatus, 532, 655
- MC\_Reset, 534, 622, 655
- MC\_SetCartesianTransform, 599, 655
- MC\_SetOverride, 535, 655
- MC\_SetPosition, 494
- MC\_Stop, 490, 537, 655
- MC\_TorqueControl, 539, 655
- MC\_UngroupAllAxes, 596, 655
- MC\_VelocityProfile, 486, 487, 521, 522, 542, 655
- MC\_WriteBoolParameter, 546, 655
- MC\_WriteParameter, 547, 655
- MCP\_AccelerationProfile, 486, 655
- MCP\_CamIn, 556, 655
- MCP\_CamTableSelect, 556, 557, 562, 655
- MCP\_CombineAxes, 564, 655
- MCP\_GearIn, 568, 655
- MCP\_GearInPos, 571, 655
- MCP\_GroupHalt, 655
- MCP\_GroupInterrupt, 616, 655
- MCP\_GroupSetOverride, 655
- MCP\_GroupSetPosition, 603, 655
- MCP\_GroupStop, 655
- MCP\_Halt, 490, 655
- MCP\_HaltSuperimposed, 492, 656
- MCP\_Home, 494, 656

- MCP\_MoveAbsolute, 496, 656
- MCP\_MoveAdditive, 500, 656
- MCP\_MoveCircularAbsolute, 656
- MCP\_MoveCircularRelative, 656
- MCP\_MoveContinuousAbsolute, 509, 656
- MCP\_MoveContinuousRelative, 513, 656
- MCP\_MoveDirectAbsolute, 656
- MCP\_MoveDirectRelative, 656
- MCP\_MoveLinearAbsolute, 656
- MCP\_MoveLinearRelative, 656
- MCP\_MovePath, 656
- MCP\_MovePath\_PH, 656
- MCP\_MoveRelative, 503, 656
- MCP\_MoveSuperimposed, 506, 656
- MCP\_MoveVelocity, 517, 656
- MCP\_PhasingAbsolute, 579, 656
- MCP\_PhasingRelative, 582, 656
- MCP\_PositionProfile, 521, 656
- MCP\_SetCartesianTransform, 656
- MCP\_SetKinTransform\_Arm, 656
- MCP\_SetOverride, 535, 656
- MCP\_Stop, 537, 656
- MCP\_TorqueControl, 539, 656
- MCP\_VelocityProfile, 542, 656
- MCU, 180, 240, 656
- MDL, 357, 358, 656
- MDLI, 358, 656
- mean value, 121
- MID, 305, 656
- minimum, 127
- MINMAX, 127, 656
- ML\_DGEBAK, 394
- ML\_DGEBAL, 396
- ML\_DGEBRD, 399
- ML\_DGECON, 401
- ML\_DGEES, 403
- ML\_DGEEV, 405
- ML\_DGEHRD, 407
- ML\_DGELQF, 409
- ML\_DGELSD, 411
- ML\_DGEQRF, 413
- ML\_DGESDD, 415
- ML\_DTRSYL, 417
- model
  - first order, 356
  - FOPDT, 179, 360
  - process, 357, 358
  - second order, 360
  - state space
    - continuous, 345
    - continuous with time delay, 342
    - discrete, 350
    - discrete with time delay, 348
- modulation
  - pulse width, 208
- MODULE, 23, 28, 32, 656
- module, 32
  - extending the REXYGEN System, 32
  - extension, 28
- motion control, 15, 112
- moving average, 111
- MP, 161, 656
- MPC, 204
- MUL, 97, 656
- multiplexer
  - bitwise, 252
- multiplication
  - by a constant, 85
  - extended, 80
  - two signals, 97
- MVD, 359, 656
- MX\_CTODPA, 419
- MX\_DIM, 421
- MX\_DSAGET, 422
- MX\_DSAREF, 424
- MX\_DSASET, 426
- MX\_DTRNSP, 428
- MX\_DTRNSQ, 430
- MX\_FILL, 432
- MX\_MAT, 433
- MX\_RAND, 434
- MX\_REFCOPY, 436
- MX\_VEC, 437
- MX\_WRITE, 438
- negation, 264
- nonlinear transformation
  - simple, 128

- NOT\_, 264, 656
- NSCL, 128, 656
- OPC server, 458
- operating system, 38
- operation
  - binary, 100
  - bitwise, 251
- operator
  - relational, 100
- optimization
  - gradient based, 86
- OR\_, 265, 266, 656
- order
  - driver execution, 28
  - driver initialization, 28
  - module execution, 32
  - module initialization, 32
  - of task execution, 39
  - of task initialization, 39
- ORHEXD, 266, 656
- OROCT, 265, 266, 657
- ORQUAD, 266, 657
- OSCALL, 38, 448, 657
- OUTHEXD, 58, 60, 657
- OUTOCT, 58, 60, 657
- Outport, 52, 54, 55, 657
- output
  - pulse, 201
  - three state, 241
- output saturation, 211
- OUTQUAD, 58, 60, 657
- OUTRHEXD, 60, 657
- OUTROCT, 60, 657
- OUTRQUAD, 60, 657
- OUTRSTD, 62, 657
- OUTSTD, 49, 56, 58, 62, 657
- overhead
  - control system core, 23
- PARA, 323, 657
- parameter
  - tick, 23
  - input-defined, 324
  - remote, 318, 320, 327, 329
  - remote acquirement, 318, 320
- PARB, 324, 657
- PARI, 324, 657
- PARR, 324, 657
- PARS, 326, 657
- path
  - full, 36
- period
  - of quick task execution, 34
  - of task execution, 39
- PID
  - autotuning, 190
  - controller, 196
  - with gain scheduling, 188
  - with input-defined parameters, 199
  - with moment autotuner, 190
  - with relay autotuner, 182
  - with static error, 185
- PIDAT, 14, 182, 657
- PIDE, 185, 657
- PIDGS, 14, 188, 657
- PIDMA, 14, 190, 458, 657
- PIDU, 182, 185, 188, 190, 196, 199, 240, 458, 480, 657
- PIDUI, 199, 657
- PJROCT, 306, 657
- PJSOCT, 307, 657
- POL, 98, 657
- polynomial
  - evaluation, 98
- position sensor
  - absolute, 107
- POUT, 201, 657
- PRBS, 162, 657
- prediction, 119
- predictive control, 204
- PRGM, 202, 657
- priority
  - dependancy on the operating system, 23
  - logic, 23
  - logical, 28, 34
  - of tasks, 39

- process
  - model, 357
  - model with variable parameters, 358
- product
  - Boolean, 244, 245
  - logical, 244, 245
- program
  - REXYGEN Studio, 28
  - REXYGEN Diagnostics, 20, 28
  - REXYGEN Diagnostics, Enable check-box, 36
  - REXYGEN Diagnostics, Halt/Run button, 36
  - REXYGEN Diagnostics, Reset button, 36
  - external, 446
- programmable block, 460
- programme
  - weekly, 279
- programmer, 202
- PROJECT, 33, 657
- project
  - main file, 22, 28
- protocol
  - UDP/IP, 455
- PSMPC, 14, 204, 657
- puls, 201
- pulse
  - manually generated, 161
- pulse counting
  - bidirectional, 253
- pulse output, 201
- pulse width modulation, 208
- PWM, 189, 194, 198, 200, 208, 229, 233, 234, 657
- QFC, 63, 64, 466, 657
- QFD, 60, 62–64, 466, 657
- QTASK, 22, 23, 30, 34, 37, 39, 343, 346, 657
- quotient, 78
  - integer, 94
- rate limiter, 131
- rate monotonic scheduling, 23
- RDC, 15, 455, 657
- RDFT, 129
- real-time
  - executive, 19, 22
- REC, 99, 657
- reciprocal value, 99
- REGEXP, 308, 657
- REL, 100, 657
- relative damping coefficient, 113, 126
- relay
  - advance, 169
  - with hysteresis, 210
- remote
  - data connection, 455
  - parameter, 318, 320
- REPLACE, 311, 658
- REXLANG, 15, 460, 658
- RLIM, 131, 658
- RLY, 169, 210, 658
- RM\_AxesGroup, 588, 595, 621, 658
- RM\_Axis, 442, 479, 480, 494, 525, 527–530, 537, 549, 551, 595, 658
- RM\_AxisOut, 549, 551, 658
- RM\_AxisSpline, 551, 658
- RM\_Feed, 591, 658
- RM\_Gcode, 593, 658
- RM\_GroupTrack, 658
- RM\_Track, 553, 658
- root
  - square, 103
- RS, 268, 658
- RTOI, 101, 658
- RTOS, 312, 658
- RTOV, 440, 447, 658
- S10F2, 132, 658
- safety analog input, 135
- safety selector, 132
- SAI, 132–135, 658
- sample and hold, 144
- SAT, 211, 658
- SC2FA, 213, 658
- schedule
  - weekly, 279



- SCU, 189, 194, 198, 200, 220, 658
- SCUV, 189, 194, 196, 198, 200, 223, 658
- SEL, 138, 658
- selector
  - active controller, 227
  - analog signal, 132, 138
  - safety, 132
  - with ramp, 149
- SELHEXD, 140, 658
- SELOCT, 140, 658
- SELQUAD, 138, 140, 658
- SELSOCT, 313, 658
- SELU, 227, 442, 658
- sensor
  - absolute position, 107
- sequence
  - pseudo-random binary, 162
- sequential control, 247, 255
- SETPA, 327, 658
- SETPB, 329, 658
- SETPI, 329, 467, 658
- setpoint, 202
- SETPR, 329, 333, 658
- SETPS, 331, 658
- SG, 164, 458, 658
- SGI, 164, 658
- SGSLP, 332, 336, 658
- SHIFTOCT, 142, 658
- SHLD, 144, 658
- signal generator, 164
- SILO, 334, 336, 658
- SILOS, 338, 659
- simulation
  - parameters, 35
  - real-time, 35
- Simulink, 35
- SINT, 122, 145, 659
- SLEEP, 13, 35, 659
- SMHCC, 229, 659
- SMHCCA, 233, 659
- SMTP, 453, 659
- SOPDT, 360, 659
- SPIKE, 135–137, 146, 659
- SQR, 102, 659
- SQRT\_, 103, 659
- square root, 103
- square value, 102
- SR, 269, 659
- SRTF, 36, 659
- SSW, 148, 442, 458, 659
- stack
  - size, 28
- standard deviation, 121
- starting unit, 112
- state machine, 247, 255
- state space model, 345, 350
  - with time delay, 342, 348
- step controller
  - with position feedback, 220
  - with velocity output, 223
- STOR, 315, 659
- SUB, 71, 104, 659
- SubSystem, 48, 50, 54, 57, 59, 659
- subsystem
  - archiving, 281
  - execution, 36
- subtraction
  - extended, 79
  - two signals, 104
- sum, 70
  - Boolean, 266
  - logical, 265, 266
- switch
  - integer signals, 260
  - simple, 148
  - unit, 240
  - with automatic selection of input, 109
- SWR, 149, 442, 659
- SWU, 227, 240, 659
- SWVMR, 442, 659
- system
  - first order, 179, 356
  - second order, 360
- TASK, 23, 30, 34, 37, 39, 343, 346, 659
- task
  - driver-triggered, 30
  - execution, 36

- execution period, 39
  - priority, 39
  - quick, 34
  - quick, execution period, 34
  - standard, 39
- TIME, 275, 278, 659
- time delay, 118, 356, 360
  - variable, 150
- timer
  - system, 30
  - weekly, 279
- TIMER\_, 270, 659
- TIODRV, 23, 30, 41, 659
- trajectory
  - time-optimal, 112
- transformation
  - binary number, 262
  - integer number, 262
- transport delay, 118
- trend
  - recording, 289, 292
- TRND, 17, 289, 292, 458, 659
- TRNDLF, 294
- TRNDV, 292, 659
- TRNDVLF, 296
- TSE, 220, 223, 241, 659
- type
  - input, 16
  - output, 16
  - parameter, 16
- types
  - of variables, 16
- user programmable block, 460
- value
  - default, 16
  - maximal, 16
  - mean, 121
  - minimal, 16
  - reciprocal, 99
  - substitute, 78, 80, 81, 83, 94, 95, 99, 103
- valve
  - motor driven, 359
  - servo, 359
- variance, 121
- VDEL, 150, 659
- VIN, 60, 62, 64, 65, 466, 659
- VOUT, 63, 66, 466, 659
- VTOR, 129, 443, 447, 659
- weekly
  - schedule, 279
- WSCH, 279, 659
- WWW, 43, 659
- ZV4IS, 151, 659

