Function Blocks of the REX Control System
Reference manual

REX Controls s.r.o.

Version 2.50.1
2016-11-07
Plzeň (Pilsen), Czech Republic
Contents

1 Introduction ........................................... 11
  1.1 How to use this manual ............................... 11
  1.2 The function block description format ................. 13
  1.3 Conventions for variables, blocks and subsystems naming 14
  1.4 The signal quality corresponding with OPC .................. 15

2 EXEC – Real-time executive configuration ................. 17
  ARC – The REX system archive ........................... 18
  EXEC – Real-time executive .............................. 20
  HMI – Human-Machine Interface Configuration ................ 22
  INFO – Description of Algorithm ........................ 24
  IODRV – The REX control system input/output driver ........ 25
  IOTASK – Driver-triggered task of the REX control system ... 27
  LPBK – Loop break ...................................... 28
  MODULE – Extension module of the REX control system ....... 29
  PROJECT – Additional Project Settings .................... 30
  QTASK – Quick task of the REX control system ............. 31
  SLEEP – Timing in Simulink ............................. 32
  SRTF – Set run-time flags ................................ 33
  OSCALL – Operating system calls ......................... 35
  TASK – Standard task of the REX control system ............ 36
  TIDRV – The REX control system input/output driver with tasks .... 38
  WWW – Internal Web Server Content ...................... 40

3 INOUT – Input and output blocks ......................... 41
  Display – Numeric display of input values .................. 42
  From, INSTD – Signal connection or input .................. 43
  Goto, OUTSTD – Signal source or output ................. 45
  GotoTagVisibility – Visibility of the signal source ........ 47
  Inport, Outport – Input and output port ................... 48
  SubSystem – Subsystem block ............................ 50
  INQUAD, INOCT, INHEXD – Multi-input blocks ............. 51
  OUTQUAD, OUTOCT, OUTHEXD – Multi-output blocks ......... 52
CONTENTS

OUTRQUAD, OUTROCT, OUTRHEXD – Multi-output blocks with verification . . . 53
OUTRSTD – Output block with verification ....................................... 55
QFC – Quality flags coding .............................................................. 56
QFD – Quality flags decoding ............................................................ 57
VIN – Validation of the input signal .................................................. 58
VOUT – Validation of the output signal .............................................. 59

4 MATH – Math blocks
ABS_ – Absolute value ................................................................. 63
ADD – Addition of two signals .......................................................... 64
ADDOCT – Addition of eight signals .................................................. 65
CNB – Boolean (logic) constant ......................................................... 66
CNE – Enumeration constant ............................................................ 67
CNI – Integer constant ................................................................. 68
CNR – Real constant ................................................................. 69
DIF_ – Difference ................................................................. 70
DIV – Division of two signals ........................................................ 71
EAS – Extended addition and subtraction ........................................... 72
EMD – Extended multiplication and division ...................................... 73
FNX – Evaluation of single-variable function ..................................... 74
FNXY – Evaluation of two-variables function ..................................... 76
GAIN – Multiplication by a constant ................................................ 78
GRADS – Gradient search optimization ............................................. 79
IADD – Integer addition ............................................................... 81
ISUB – Integer subtraction ........................................................... 83
IMUL – Integer multiplication ........................................................ 85
IDIV – Integer division ............................................................... 87
IMOD – Remainder after integer division ........................................ 88
LIN – Linear interpolation ............................................................. 89
MUL – Multiplication of two signals ................................................. 90
POL – Polynomial evaluation .......................................................... 91
REC – Reciprocal value ............................................................... 92
REL – Relational operator ........................................................... 93
RTOI – Real to integer number conversion ..................................... 94
SQR – Square value ................................................................. 95
SQRT_ – Square root ............................................................... 96
SUB – Subtraction of two signals ....................................................... 97

5 ANALOG – Analog signal processing
ABSROT – Processing data from absolute position sensor .................... 101
ASW – Switch with automatic selection of input .................................. 103
AVG – Moving average filter ......................................................... 105
AVS – Motion control unit ............................................................. 106
BPF – Band-pass filter ............................................................... 107
## CONTENTS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP</td>
<td>Comparator with hysteresis</td>
<td>108</td>
</tr>
<tr>
<td>CNDR</td>
<td>Nonlinear conditioner</td>
<td>109</td>
</tr>
<tr>
<td>DEL</td>
<td>Delay with initialization</td>
<td>111</td>
</tr>
<tr>
<td>DELM</td>
<td>Time delay</td>
<td>112</td>
</tr>
<tr>
<td>DER</td>
<td>Derivation, filtering and prediction from the last n+1 samples</td>
<td>113</td>
</tr>
<tr>
<td>EVAR</td>
<td>Moving mean value and standard deviation</td>
<td>115</td>
</tr>
<tr>
<td>INTE</td>
<td>Controlled integrator</td>
<td>116</td>
</tr>
<tr>
<td>KDER</td>
<td>Derivation and filtering of the input signal</td>
<td>118</td>
</tr>
<tr>
<td>LPF</td>
<td>Low-pass filter</td>
<td>120</td>
</tr>
<tr>
<td>MINMAX</td>
<td>Running minimum and maximum</td>
<td>121</td>
</tr>
<tr>
<td>NSCL</td>
<td>Nonlinear scaling factor</td>
<td>122</td>
</tr>
<tr>
<td>RDFT</td>
<td>Running discrete Fourier transform</td>
<td>123</td>
</tr>
<tr>
<td>RLIM</td>
<td>Rate limiter</td>
<td>125</td>
</tr>
<tr>
<td>S1OF2</td>
<td>One of two analog signals selector</td>
<td>126</td>
</tr>
<tr>
<td>SAI</td>
<td>Safety analog input</td>
<td>129</td>
</tr>
<tr>
<td>SEL</td>
<td>Analog signal selector</td>
<td>132</td>
</tr>
<tr>
<td>SELQUAD, SELCT, SELHEXD</td>
<td>Analog signal selectors</td>
<td>134</td>
</tr>
<tr>
<td>SHIFTOCT</td>
<td>Data shift register</td>
<td>136</td>
</tr>
<tr>
<td>SHLD</td>
<td>Sample and hold</td>
<td>138</td>
</tr>
<tr>
<td>SINT</td>
<td>Simple integrator</td>
<td>139</td>
</tr>
<tr>
<td>SPIKE</td>
<td>Spike filter</td>
<td>140</td>
</tr>
<tr>
<td>SSW</td>
<td>Simple switch</td>
<td>142</td>
</tr>
<tr>
<td>SWR</td>
<td>Selector with ramp</td>
<td>143</td>
</tr>
<tr>
<td>VDEL</td>
<td>Variable time delay</td>
<td>144</td>
</tr>
<tr>
<td>ZV4IS</td>
<td>Zero vibration input shaper</td>
<td>145</td>
</tr>
</tbody>
</table>

### 6 GEN – Signal generators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANLS</td>
<td>Controlled generator of piecewise linear function</td>
<td>150</td>
</tr>
<tr>
<td>BINS</td>
<td>Controlled binary sequence generator</td>
<td>152</td>
</tr>
<tr>
<td>BIS</td>
<td>Binary sequence generator</td>
<td>154</td>
</tr>
<tr>
<td>MP</td>
<td>Manual pulse generator</td>
<td>155</td>
</tr>
<tr>
<td>PRBS</td>
<td>Pseudo-random binary sequence generator</td>
<td>156</td>
</tr>
<tr>
<td>SG, SGI</td>
<td>Signal generators</td>
<td>158</td>
</tr>
</tbody>
</table>

### 7 REG – Function blocks for control

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARLY</td>
<td>Advance relay</td>
<td>163</td>
</tr>
<tr>
<td>FLCU</td>
<td>Fuzzy logic controller unit</td>
<td>164</td>
</tr>
<tr>
<td>FRID</td>
<td>* Frequency response identification</td>
<td>167</td>
</tr>
<tr>
<td>I3PM</td>
<td>Identification of a three parameter model</td>
<td>169</td>
</tr>
<tr>
<td>LC</td>
<td>Lead compensator</td>
<td>171</td>
</tr>
<tr>
<td>LLC</td>
<td>Lead-lag compensator</td>
<td>172</td>
</tr>
<tr>
<td>MCU</td>
<td>Manual control unit</td>
<td>173</td>
</tr>
<tr>
<td>PIDAT</td>
<td>PID controller with relay autotuner</td>
<td>175</td>
</tr>
<tr>
<td>PIDE</td>
<td>PID controller with defined static error</td>
<td>178</td>
</tr>
<tr>
<td>Block</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>PIDGS</td>
<td>PID controller with gain scheduling</td>
<td>180</td>
</tr>
<tr>
<td>PIDMA</td>
<td>PID controller with moment autotuner</td>
<td>182</td>
</tr>
<tr>
<td>PIDU</td>
<td>PID controller unit</td>
<td>188</td>
</tr>
<tr>
<td>PIDUI</td>
<td>PID controller unit with variable parameters</td>
<td>191</td>
</tr>
<tr>
<td>POUT</td>
<td>Pulse output</td>
<td>193</td>
</tr>
<tr>
<td>PRGM</td>
<td>Setpoint programmer</td>
<td>194</td>
</tr>
<tr>
<td>PSMPC</td>
<td>Pulse-step model predictive controller</td>
<td>196</td>
</tr>
<tr>
<td>PWM</td>
<td>Pulse width modulation</td>
<td>200</td>
</tr>
<tr>
<td>RLY</td>
<td>Relay with hysteresis</td>
<td>202</td>
</tr>
<tr>
<td>SAT</td>
<td>Saturation with variable limits</td>
<td>203</td>
</tr>
<tr>
<td>SC2FA</td>
<td>State controller for 2nd order system with frequency autotuner</td>
<td>205</td>
</tr>
<tr>
<td>SCU</td>
<td>Step controller with position feedback</td>
<td>211</td>
</tr>
<tr>
<td>SCUV</td>
<td>Step controller unit with velocity input</td>
<td>214</td>
</tr>
<tr>
<td>SELU</td>
<td>Controller selector unit</td>
<td>218</td>
</tr>
<tr>
<td>SMHCC</td>
<td>Sliding mode heating/cooling controller</td>
<td>220</td>
</tr>
<tr>
<td>SMHCCA</td>
<td>Sliding mode heating/cooling controller with autotuner</td>
<td>224</td>
</tr>
<tr>
<td>SWU</td>
<td>Switch unit</td>
<td>231</td>
</tr>
<tr>
<td>TSE</td>
<td>Three-state element</td>
<td>232</td>
</tr>
</tbody>
</table>

8 LOGIC – Logic control

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Logical product of two signals</td>
<td>234</td>
</tr>
<tr>
<td>ANDOCT</td>
<td>Logical product of eight signals</td>
<td>235</td>
</tr>
<tr>
<td>ATMT</td>
<td>Finite-state automaton</td>
<td>236</td>
</tr>
<tr>
<td>BDOCT, BDHEXD</td>
<td>Bitwise demultiplexers</td>
<td>239</td>
</tr>
<tr>
<td>BITOP</td>
<td>Bitwise operation</td>
<td>240</td>
</tr>
<tr>
<td>BMOCT, BMHEXD</td>
<td>Bitwise multiplexers</td>
<td>241</td>
</tr>
<tr>
<td>COUNT</td>
<td>Controlled counter</td>
<td>242</td>
</tr>
<tr>
<td>EATMT</td>
<td>Extended finite-state automaton</td>
<td>244</td>
</tr>
<tr>
<td>EDGE</td>
<td>Falling/rising edge detection in a binary signal</td>
<td>247</td>
</tr>
<tr>
<td>INTSM</td>
<td>Integer number bit shift and mask</td>
<td>248</td>
</tr>
<tr>
<td>ISSW</td>
<td>Simple switch for integer signals</td>
<td>249</td>
</tr>
<tr>
<td>INTSM</td>
<td>Integer number bit shift and mask</td>
<td>250</td>
</tr>
<tr>
<td>ITOI</td>
<td>Transformation of integer and binary numbers</td>
<td>251</td>
</tr>
<tr>
<td>NOT_</td>
<td>Boolean complementation</td>
<td>253</td>
</tr>
<tr>
<td>OR_</td>
<td>Logical sum of two signals</td>
<td>254</td>
</tr>
<tr>
<td>OROCT</td>
<td>Logical sum of eight signals</td>
<td>255</td>
</tr>
<tr>
<td>RS</td>
<td>Reset-set flip-flop circuit</td>
<td>256</td>
</tr>
<tr>
<td>SR</td>
<td>Set-reset flip-flop circuit</td>
<td>257</td>
</tr>
<tr>
<td>TIMER_</td>
<td>Multipurpose timer</td>
<td>258</td>
</tr>
</tbody>
</table>

9 TIME – Blocks for handling time

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE_</td>
<td>Current date</td>
<td>262</td>
</tr>
<tr>
<td>DATETIME</td>
<td>Get, set and convert time</td>
<td>263</td>
</tr>
<tr>
<td>TIME</td>
<td>Current time</td>
<td>266</td>
</tr>
</tbody>
</table>
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSCH</td>
<td>Weekly schedule</td>
<td>267</td>
</tr>
<tr>
<td>10 ARC</td>
<td>Data archiving</td>
<td>269</td>
</tr>
<tr>
<td>10.1</td>
<td>Functionality of the archiving subsystem</td>
<td>270</td>
</tr>
<tr>
<td>10.2</td>
<td>Generating alarms and events</td>
<td>271</td>
</tr>
<tr>
<td></td>
<td>ALB, ALBI – Alarms for Boolean value</td>
<td>271</td>
</tr>
<tr>
<td></td>
<td>ALN, ALNI – Alarms for numerical value</td>
<td>273</td>
</tr>
<tr>
<td>10.3</td>
<td>Trends recording</td>
<td>275</td>
</tr>
<tr>
<td></td>
<td>ACD – Archive compression using Delta criterion</td>
<td>275</td>
</tr>
<tr>
<td></td>
<td>TRND – Real-time trend recording</td>
<td>277</td>
</tr>
<tr>
<td></td>
<td>TRNDV – Real-time trend recording with vector input</td>
<td>280</td>
</tr>
<tr>
<td></td>
<td>TRNDLF – * Real-time trend recording (lock-free)</td>
<td>282</td>
</tr>
<tr>
<td></td>
<td>TRNDVLFLF – * Real-time trend recording (for vector signals, lock-free)</td>
<td>284</td>
</tr>
<tr>
<td>10.4</td>
<td>Archive management</td>
<td>285</td>
</tr>
<tr>
<td></td>
<td>AFLUSH – Forced archive flushing</td>
<td>285</td>
</tr>
<tr>
<td>11 STRING</td>
<td>Blocks for string operations</td>
<td>287</td>
</tr>
<tr>
<td></td>
<td>CNS – String constant</td>
<td>288</td>
</tr>
<tr>
<td></td>
<td>CONCAT – * Concat string by pattern</td>
<td>289</td>
</tr>
<tr>
<td></td>
<td>FIND – Find a Substring</td>
<td>290</td>
</tr>
<tr>
<td></td>
<td>LEN – String length</td>
<td>291</td>
</tr>
<tr>
<td></td>
<td>MID – Substring Extraction</td>
<td>292</td>
</tr>
<tr>
<td></td>
<td>PJROCT – * Parse JSON string (real output)</td>
<td>293</td>
</tr>
<tr>
<td></td>
<td>PJSOCT – * Parse JSON string (string output)</td>
<td>294</td>
</tr>
<tr>
<td></td>
<td>REGEXP – Regular expresion parser</td>
<td>295</td>
</tr>
<tr>
<td></td>
<td>REPLACE – Replace substring</td>
<td>296</td>
</tr>
<tr>
<td></td>
<td>RTOS – Real Number to String Conversion</td>
<td>297</td>
</tr>
<tr>
<td></td>
<td>SELSOCT – * String selector</td>
<td>298</td>
</tr>
<tr>
<td></td>
<td>STOR – String to real number conversion</td>
<td>300</td>
</tr>
<tr>
<td>12 PARAM</td>
<td>Blocks for parameter handling</td>
<td>301</td>
</tr>
<tr>
<td></td>
<td>GETPA – Block for remote array parameter acquirement</td>
<td>302</td>
</tr>
<tr>
<td></td>
<td>GETPR, GETPI, GETPB – Blocks for remote parameter acquirement</td>
<td>304</td>
</tr>
<tr>
<td></td>
<td>GETPS – * Block for remote string parameter acquirement</td>
<td>306</td>
</tr>
<tr>
<td></td>
<td>PARA – Block with input-defined array parameter</td>
<td>307</td>
</tr>
<tr>
<td></td>
<td>PARR, PARI, PARB – Blocks with input-defined parameter</td>
<td>308</td>
</tr>
<tr>
<td></td>
<td>PARS – * Block with input-defined string parameter</td>
<td>310</td>
</tr>
<tr>
<td></td>
<td>SETPA – Block for remote array parameter setting</td>
<td>311</td>
</tr>
<tr>
<td></td>
<td>SETPR, SETPI, SETPB – Blocks for remote parameter setting</td>
<td>313</td>
</tr>
<tr>
<td></td>
<td>SETPS – * Block for remote string parameter setting</td>
<td>315</td>
</tr>
<tr>
<td></td>
<td>SGSLP – Set, get, save and load parameters</td>
<td>316</td>
</tr>
<tr>
<td></td>
<td>SILO – Save input value, load output value</td>
<td>320</td>
</tr>
</tbody>
</table>
13 MODEL – Dynamic systems simulation
   CDELSSM – Continuous state space model of a linear system with time delay .......... 321
   CSSM – Continuous state space model of a linear system ............................... 322
   DDELSSM – Discrete state space model of a linear system with time delay .......... 325
   DSSM – Discrete state space model of a linear system .................................. 327
   FOPDT – First order plus dead-time model .................................................. 330
   MDL – Process model .................................................................................. 331
   MDLI – Process model with input-defined parameters ...................................... 332
   MVD – Motorized valve drive ....................................................................... 333
   SOPDT – Second order plus dead-time model ............................................... 335

14 MATRIX – Blocks for matrix and vector operations
   CNA – * Array (vector/matrix) constant ......................................................... 338
   RTOV – Vector multiplexer .......................................................................... 339
   SWMR – Vector/matrix/reference signal switch .............................................. 340
   VTOR – Vector demultiplexer ...................................................................... 341

15 SPEC – Special blocks
   EPC – External program call .......................................................................... 344
   HTTP – HTTP GET or POST request .............................................................. 346
   SMTP – Send email message via SMTP ......................................................... 347
   RDC – Remote data connection .................................................................... 351
   REXLANG – User programmable block ......................................................... 356

16 MC_SINGLE – Motion control - single axis blocks
   RM_Axis – Motion control axis ...................................................................... 372
   MC_AccelerationProfile, MCP_AccelerationProfile – Acceleration profile .......... 380
   MC_Halt, MCP_Halt – Stopping a movement (interruptible) ............................ 383
   MC_HaltSuperimposed, MCP_HaltSuperimposed – Stopping a movement (superimposed and interruptible) .......................................................... 385
   MC_Home, MCP_Home – Homing ................................................................... 386
   MC_MoveAbsolute, MCP_MoveAbsolute – Move to position (absolute coordinate) ............................................................................................................. 388
   MC_MoveAdditive, MCP_MoveAdditive – Move to position (relative to previous motion) .................................................................................................................. 391
   MC_MoveRelative, MCP_MoveRelative – Move to position (relative to execution point) ........................................................................................................... 395
   MC_MoveSuperimposed, MCP_MoveSuperimposed – Superimposed move ...... 397
   MC_MoveContinuousAbsolute, MCP_MoveContinuousAbsolute – Move to position (absolute coordinate) ............................................................................ 401
   MC_MoveContinuousRelative, MCP_MoveContinuousRelative – Move to position (relative to previous motion) ................................................................. 403
   MC_MoveVelocity, MCP_MoveVelocity – Move with constant velocity ......... 407
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC_PositionProfile, MCP_PositionProfile</td>
<td>Position profile</td>
<td>412</td>
</tr>
<tr>
<td>MC_Power</td>
<td>Axis activation (power on/off)</td>
<td>416</td>
</tr>
<tr>
<td>MC_ReadActualPosition</td>
<td>Read actual position</td>
<td>417</td>
</tr>
<tr>
<td>MC_ReadAxisError</td>
<td>Read axis error</td>
<td>418</td>
</tr>
<tr>
<td>MC_ReadBoolParameter</td>
<td>Read axis parameter (bool)</td>
<td>419</td>
</tr>
<tr>
<td>MC_ReadParameter</td>
<td>Read axis parameter</td>
<td>420</td>
</tr>
<tr>
<td>MC_ReadStatus</td>
<td>Read axis status</td>
<td>422</td>
</tr>
<tr>
<td>MC_Reset</td>
<td>Reset axis errors</td>
<td>424</td>
</tr>
<tr>
<td>MC_SetOverride, MCP_SetOverride</td>
<td>Set override factors</td>
<td>425</td>
</tr>
<tr>
<td>MC_Stop, MCP_Stop</td>
<td>Stopping a movement</td>
<td>427</td>
</tr>
<tr>
<td>MC_TorqueControl, MCP_TorqueControl</td>
<td>Torque/force control</td>
<td>429</td>
</tr>
<tr>
<td>MC_VelocityProfile, MCP_VelocityProfile</td>
<td>Velocity profile</td>
<td>432</td>
</tr>
<tr>
<td>MC_WriteBoolParameter</td>
<td>Write axis parameter (bool)</td>
<td>436</td>
</tr>
<tr>
<td>MC_WriteParameter</td>
<td>Write axis parameter</td>
<td>437</td>
</tr>
<tr>
<td>RM_AxisOut</td>
<td>Axis output</td>
<td>439</td>
</tr>
<tr>
<td>RM_AxisSpline</td>
<td>Commanded values interpolation</td>
<td>441</td>
</tr>
<tr>
<td>RM_Track</td>
<td>Tracking and inching</td>
<td>443</td>
</tr>
<tr>
<td>RM_AxesGroup</td>
<td>Axes group for coordinated motion control</td>
<td>445</td>
</tr>
<tr>
<td>RM_Feed</td>
<td>MC Feeder</td>
<td>446</td>
</tr>
<tr>
<td>RM_Gcode</td>
<td>CNC motion control</td>
<td>447</td>
</tr>
<tr>
<td>MC_AddAxisToGroup</td>
<td>Adds one axis to a group</td>
<td>450</td>
</tr>
<tr>
<td>MC_GearIn, MCP_GearIn</td>
<td>Engage the master/slave velocity ratio</td>
<td>452</td>
</tr>
<tr>
<td>MC_GearInPos, MCP_GearInPos</td>
<td>Engage the master/slave velocity ratio in defined position</td>
<td>454</td>
</tr>
<tr>
<td>MC_GearOut</td>
<td>Disengage the master/slave velocity ratio</td>
<td>457</td>
</tr>
<tr>
<td>MC_PhasingAbsolute, MCP_PhasingAbsolute</td>
<td>Phase shift in synchronized motion (absolute coordinates)</td>
<td>460</td>
</tr>
<tr>
<td>MC_PhasingRelative, MCP_PhasingRelative</td>
<td>Phase shift in synchronized motion (relative coordinates)</td>
<td>465</td>
</tr>
<tr>
<td>MC_Reset</td>
<td>Reset axis errors</td>
<td>467</td>
</tr>
<tr>
<td>RM_AxesGroup</td>
<td>Axes group for coordinated motion control</td>
<td>470</td>
</tr>
<tr>
<td>RM_Feed</td>
<td>* MC Feeder</td>
<td>476</td>
</tr>
<tr>
<td>RM_Gcode</td>
<td>* CNC motion control</td>
<td>479</td>
</tr>
<tr>
<td>MC_AddAxisToGroup</td>
<td>Adds one axis to a group</td>
<td>480</td>
</tr>
<tr>
<td>MC_UngroupAllAxes</td>
<td>Removes all axes from the group</td>
<td>482</td>
</tr>
<tr>
<td>MC_GroupEnable</td>
<td>Changes the state of a group to GroupEnable</td>
<td>483</td>
</tr>
<tr>
<td>MC_GroupDisable</td>
<td>Changes the state of a group to GroupDisabled</td>
<td>484</td>
</tr>
<tr>
<td>MC_SetCartesianTransform</td>
<td>Sets Cartesian transformation</td>
<td>485</td>
</tr>
<tr>
<td>MC_ReadCartesianTransform</td>
<td>Reads the parameter of the cartesian transformation</td>
<td>486</td>
</tr>
</tbody>
</table>
MC_GroupSetPosition, MCP_GroupSetPosition – Sets the position of all axes in a group

MC_GroupReadActualPosition – Read actual position in the selected coordinate system

MC_GroupReadActualVelocity – Read actual velocity in the selected coordinate system

MC_GroupReadActualAcceleration – Read actual acceleration in the selected coordinate system

MC_GroupStop – Stopping a group movement

MC_GroupHalt – Stopping a group movement (interruptible)

MC_GroupInterrupt, MCP_GroupInterrupt – Read a group interrupt

MC_GroupContinue – Continuation of interrupted movement

MC_GroupReadStatus – Read a group status

MC_GroupReadError – Read a group error

MC_GroupReset – Reset axes errors

MC_MoveLinearAbsolute – Linear move to position (absolute coordinates)

MC_MoveLinearRelative – Linear move to position (relative to execution point)

MC_MoveCircularAbsolute – Circular move to position (absolute coordinates)

MC_MoveCircularRelative – Circular move to position (relative to execution point)

MC_MoveDirectAbsolute – Direct move to position (absolute coordinates)

MC_MoveDirectRelative – Direct move to position (relative to execution point)

MC_MovePath – General spatial trajectory generation

MC_GroupSetOverride – Set group override factors

A Licensing of individual function blocks

B Error codes of the REX Control System

Bibliography

Index

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

Introduction

The manual “REX system function blocks” is a reference manual for the REX control 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 REX control 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 REX control 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 REX control system. These blocks provide the connection between the control tasks and input/output drivers.
CHAPTER 1. INTRODUCTION

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 REX 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 REX Control 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 the REX Control System are included in this sublibrary.
1.2. THE FUNCTION BLOCK DESCRIPTION FORMAT

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 REX targets or between the Simulink and the REX 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

Example – 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.
CHAPTER 1. INTRODUCTION

If the block function is obvious, the section Example 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:

\[
\begin{array}{|c|c|}
\hline
\text{Type} & \text{Meaning} \\hline
\text{bool} & \text{Boolean value 0 or 1} \\hline
\text{byte} & \text{8 bit integer number without the sign} \\hline
\text{short} & \text{16 bit integer number with the sign} \\hline
\text{long} & \text{32 bit integer number with the sign} \\hline
\text{word} & \text{16 bit integer number without the sign} \\hline
\text{dword} & \text{32 bit integer number without the sign} \\hline
\text{float} & \text{32 bit real number in floating point arithmetics} \\hline
\text{double} & \text{64 bit real number in floating point arithmetics} \\hline
\text{string} & \text{character string} \\hline
\end{array}
\]

The meaning of the three columns is quite obvious. The third column contains the item <type>. The REX 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 REX control 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 \(\odot<\text{def}>\) is not listed in the second column, it is equal to zero. If the values of \(\downarrow<\text{min}>\) and/or \(\uparrow<\text{max}>\) are missing, the limits are given by the the minimum and/or maximum of the corresponding type (see table 1.1).

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>Boolean value 0 or 1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>byte</td>
<td>8 bit integer number without the sign</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>short</td>
<td>16 bit integer number with the sign</td>
<td>-32768</td>
<td>32767</td>
</tr>
<tr>
<td>long</td>
<td>32 bit integer number with the sign</td>
<td>-2147483648</td>
<td>2147483647</td>
</tr>
<tr>
<td>word</td>
<td>16 bit integer number without the sign</td>
<td>0</td>
<td>65535</td>
</tr>
<tr>
<td>dword</td>
<td>32 bit integer number without the sign</td>
<td>0</td>
<td>4294967295</td>
</tr>
<tr>
<td>float</td>
<td>32 bit real number in floating point arithmetics</td>
<td>(&lt;-3.4E+38)</td>
<td>(&gt;3.4E+38)</td>
</tr>
<tr>
<td>double</td>
<td>64 bit real number in floating point arithmetics</td>
<td>(&lt;-1.7E+308)</td>
<td>(&gt;1.7E+308)</td>
</tr>
<tr>
<td>string</td>
<td>character string</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1.1: Types of variables in the REX control system.

1.3 Conventions for variables, blocks and subsystems naming

Several conventions are used to simplify the use of the REX 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, ptype1, 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 REX control 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 RexComp compiler which generates warnings if inappropriate characters are found.

## 1.4 The signal quality corresponding with OPC

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

<table>
<thead>
<tr>
<th>Bit number</th>
<th>Bit weight</th>
<th>Bit field</th>
<th>Quality</th>
<th>Substatus</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6</td>
<td>128 64</td>
<td>BAD</td>
<td>0 0</td>
<td>S S S S</td>
<td>L L</td>
</tr>
<tr>
<td>5 4 3 2</td>
<td>32 16 8 4</td>
<td>UNCERTAIN</td>
<td>0 1</td>
<td>S S S S</td>
<td>L L</td>
</tr>
<tr>
<td>1 0</td>
<td>2 1</td>
<td>not used in OPC</td>
<td>1 0</td>
<td>S S S S</td>
<td>L L</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GOOD</td>
<td>1 1</td>
<td>S S S S</td>
<td>L L</td>
</tr>
</tbody>
</table>

Table 1.2: The quality flags structure
The basic quality type is determined by the QQ flags in the two most important bits. Based on these the quality is distinguished between GOOD, UNCERTAIN and BAD. The four SSSS bits provide more detailed information about the signal. They have different meaning for each basic quality. The two least significant bits LL inform whether the value exceeded its limits or if it is constant. Additional details and the meaning of all bits can be found in [1], chapter 6.8.
Chapter 2

EXEC – Real-time executive configuration

Contents

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARC</td>
<td>The REX system archive</td>
<td>18</td>
</tr>
<tr>
<td>EXEC</td>
<td>Real-time executive</td>
<td>20</td>
</tr>
<tr>
<td>HMI</td>
<td>Human-Machine Interface Configuration</td>
<td>22</td>
</tr>
<tr>
<td>INFO</td>
<td>Description of Algorithm</td>
<td>24</td>
</tr>
<tr>
<td>IODRV</td>
<td>The REX control system input/output driver</td>
<td>25</td>
</tr>
<tr>
<td>IOTASK</td>
<td>Driver-triggered task of the REX control system</td>
<td>27</td>
</tr>
<tr>
<td>LPBRK</td>
<td>Loop break</td>
<td>28</td>
</tr>
<tr>
<td>MODULE</td>
<td>Extension module of the REX control system</td>
<td>29</td>
</tr>
<tr>
<td>PROJECT</td>
<td>Additional Project Settings</td>
<td>30</td>
</tr>
<tr>
<td>QTASK</td>
<td>Quick task of the REX control system</td>
<td>31</td>
</tr>
<tr>
<td>SLEEP</td>
<td>Timing in Simulink</td>
<td>32</td>
</tr>
<tr>
<td>SRTF</td>
<td>Set run-time flags</td>
<td>33</td>
</tr>
<tr>
<td>OSCALL</td>
<td>Operating system calls</td>
<td>35</td>
</tr>
<tr>
<td>TASK</td>
<td>Standard task of the REX control system</td>
<td>36</td>
</tr>
<tr>
<td>TIODRV</td>
<td>The REX control system input/output driver with tasks</td>
<td>38</td>
</tr>
<tr>
<td>WWW</td>
<td>Internal Web Server Content</td>
<td>40</td>
</tr>
</tbody>
</table>
**ARC – The REX system archive**

**Block Symbol**

```
 prev next
ARC
```

**Licence:** STANDARD

**Function Description**

The ARC block is intended for archives configuration in the REX 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 REX control 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 RexView program after successful connecting to the target device.

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

**Input**

- **prev**
  Input for connecting with the next output of the preceding archive or with the Archives output of the EXEC block in the case of the first archive

**Output**

- **next**
  Output for creating sequences of archives by connecting to the prev input of the following archive
### Parameters

<table>
<thead>
<tr>
<th><strong>atype</strong></th>
<th>Archive type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>archive is allocated in the RAM memory (data is irreversibly lost after restarting the target device)</td>
</tr>
<tr>
<td>2</td>
<td>archive is allocated in backed-up memory, e.g. CMOS (data remains available after restarting the target device)</td>
</tr>
<tr>
<td>3</td>
<td>archive is allocated on a drive (data remains available in the file after restarting)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>asize</strong></th>
<th>Size of the archive in bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\downarrow 256 \circ 102400) long</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>nmarks</strong></th>
<th>Number of time stamps for speeding-up sequential seeking in the archive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\downarrow 2 \circ 720) long</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>ldaymax</strong></th>
<th>Maximum size of archive per day [bytes]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\downarrow 1000 \uparrow 2147480000 \circ 1048576) large</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>period</strong></th>
<th>Period of writing data to disk [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\circ 60.0) double</td>
</tr>
</tbody>
</table>
EXEC – Real-time executive

Block Symbol Licence: STANDARD

Function Description

The EXEC block is a cornerstone of the so-called project main file in the .mdl format, which configures individual subsystems of the REX control 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 RexComp compiler during building of the overall REX control system application.

The REX control 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 RexComp compiler as its limits vary by selected target platform. Generally speaking, the lower period is used, the higher computational requirements of the REX Control 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 REX control.
The REX control system uses 32 logical priorities, which are internally mapped to the target platform operating system dependent priorities. The highest logical priority of the REX control 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

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

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>Target device</td>
</tr>
<tr>
<td>tick</td>
<td>The base period (tick) of the REX control system core and also the quick task (QTASK) period (in seconds)</td>
</tr>
<tr>
<td>ntick0</td>
<td>The multiplication tick*ntick0 determines the period of tasks connected to Level0</td>
</tr>
<tr>
<td>ntick1</td>
<td>The multiplication tick*ntick1 determines the period of tasks connected to Level1</td>
</tr>
<tr>
<td>ntick2</td>
<td>The multiplication tick*ntick2 determines the period of tasks connected to Level2</td>
</tr>
<tr>
<td>ntick3</td>
<td>The multiplication tick*ntick3 determines the period of tasks connected to Level3</td>
</tr>
<tr>
<td>pri0</td>
<td>Priority of all Level0 tasks</td>
</tr>
<tr>
<td>pri1</td>
<td>Priority of all Level1 tasks</td>
</tr>
<tr>
<td>pri2</td>
<td>Priority of all Level2 tasks</td>
</tr>
<tr>
<td>pri3</td>
<td>Priority of all Level3 tasks</td>
</tr>
</tbody>
</table>
HMI – Human-Machine Interface Configuration

Block Symbol

Licence: STANDARD

Function Description

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

The REX Control System currently provides three straightforward methods of how to create Human-Machine Interface:

- **WebWatch** is an auto-generated HMI from the RexDraw development tool during project compilation. It has similar look, attributes and functions as the online mode of the RexDraw 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 RexHMI Designer with the RexHMI extensions. The RexHMI 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 RexDraw development tool and the RexHMI Designer must be installed.

Parameters

- **IncludeHMI** Include HMI files in the project  
  ○ on  bool

- **HmiDir** Output folder for HMI files  
  ○ hmi  string

- **SourceDir** Source directory  
  ○ hmisrc  string

- **GenerateWebWatch** Generate WebWatch HMI files  
  ○ on  bool

- **GenerateRexHMI** Generate HMI from SVG and JS files  
  ○ on  bool

- **RedirectToHMI** Web server will automatically redirect to HMI webpage if enabled otherwise it will serve a standard home page as a starting page.  
  ○ on  bool

- **Compression** Enables data compression in gzip format.  
  ○ on  bool
INFO – Description of Algorithm

Block Symbol

 Licence: STANDARD

Function Description

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

Parameters

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

Block Symbol

License: STANDARD

Function Description

The input/output drivers of the REX control system are implemented as extension modules (see the MODULE block). A module can contain several drivers, which are added to the REX control 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 REX control 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 REX control system driver.

The majority of drivers stores its own configuration data in files with .rio extension (REX 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 RexDraw program of the REX 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 RexView diagnostic program.

Input

\texttt{prev} \hspace{1cm} Input for connecting the driver with the Drivers output of the \texttt{EXEC} block or with the \texttt{next} output of the preceding driver \hspace{1cm} \texttt{long}

Output

\texttt{next} \hspace{1cm} Output for connecting to the \texttt{prev} input of the succeeding driver \hspace{1cm} \texttt{long}

Parameters

\texttt{module} \hspace{1cm} Name of the module, which includes the input/output driver (mandatory only if module name differs from \texttt{classname}) \hspace{1cm} \texttt{string}

\texttt{classname} \hspace{1cm} I/O driver class name; case sensitive! \hspace{1cm} \texttt{DrvClass} \hspace{1cm} \texttt{string}

\texttt{cfgname} \hspace{1cm} Name of the driver configuration file \hspace{1cm} \texttt{iodrv.rio} \hspace{1cm} \texttt{string}

\texttt{factor} \hspace{1cm} Multiple of the \texttt{EXEC} block’s \texttt{tick} parameter defining the driver’s task execution period \hspace{1cm} \texttt{1} to \texttt{10} \hspace{1cm} \texttt{long}

\texttt{stack} \hspace{1cm} Stack size of the driver’s task in bytes \hspace{1cm} \texttt{1024} to \texttt{10240} \hspace{1cm} \texttt{long}

\texttt{pri} \hspace{1cm} Logical priority of the driver’s task \hspace{1cm} \texttt{1} to \texttt{31} \hspace{1cm} \texttt{long}

\texttt{timer} \hspace{1cm} Driver is a source of time \hspace{1cm} \texttt{bool}
IOTASK – Driver-triggered task of the REX control system

Function Description
Standard tasks of the REX control 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
prev Input for connecting the first task to the Tasks output of the TIODRV block or for connecting to the previous task’s next output

Output
next Output for sequencing the tasks by connecting to the prev input of the following task

Parameters
factor Execution factor which can be used to determine the task execution period, see the user guide of the corresponding I/O driver 1
stack Stack size [bytes] 10240
filename Name of the file with the .mdl extension which contains the task algorithm; in the case filename is not specified, the filename is given by the name of the IOTASK block in the project main file (the .mdl extension is attached automatically)
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 REX control system function blocks. The block is usually placed in all feedback loops in the scheme. Its behavior differs in the REX control 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 RexComp 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 RexComp 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 REX control system and the Simulink system.

Input

\[ u \quad \text{Input signal} \quad \text{double} \]

Output

\[ y \quad \text{Output signal} \quad \text{double} \]
MODULE – Extension module of the REX control system

Block Symbol

Licence: STANDARD

Function Description

The REX control 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 REX control 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

prev  Input for connecting the module with the Modules output of the EXEC block or with the next output of the preceding module long

Output

next  Output for connecting to the prev input of the succeeding module long
CHAPTER 2. EXEC – REAL-TIME EXECUTIVE CONFIGURATION

PROJECT – Additional Project Settings

Block Symbol

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompileParams</td>
<td>Command-line options which are passed to the RexComp during project compilation.</td>
<td>string</td>
</tr>
<tr>
<td>TargetURL</td>
<td>URL address of a target on which the configuration should be run. The address is inserted into all connection dialogs automatically.</td>
<td>string</td>
</tr>
</tbody>
</table>
QTASK – Quick task of the REX control 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 REX control 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 REX 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 RexView diagnostic program.

Input

prev Input for connecting the task with the QTask output of the EXEC block long

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>factor</td>
<td>Multiple of the EXEC block’s tick parameter defining the quick task execution period</td>
<td>long</td>
</tr>
<tr>
<td>stack</td>
<td>Stack size [bytes]</td>
<td>10240 long</td>
</tr>
<tr>
<td>filename</td>
<td>Name of the file with the .mdl extension which contains the quick task algorithm; in the case filename is not specified, the filename is given by the name of the QTASK block in the project main file (the .mdl extension is attached automatically)</td>
<td>string</td>
</tr>
</tbody>
</table>
CHAPTER 2. EXEC – REAL-TIME EXECUTIVE CONFIGURATION

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 REX control system, timing and execution is provided by system resources (see the EXEC block) and the SLEEP block is ignored.

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

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

Parameter

\[ ts \]  
Simulation scheme execution period (in seconds)  
\[ \infty 0.1 \text{ double} \]
SRTF – Set run-time flags

Function Description

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

- **One-shot execution** of the object. If the object execution is disabled by the EXDIS = on input or by the RexView 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 RexView 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 RexView 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 REX control system.
CHAPTER 2. EXEC – REAL-TIME EXECUTIVE CONFIGURATION

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

**Inputs**

- **EXDIS** Disable execution 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 bname parameter was not found
  - 2 ...... REX control system internal error (invalid pointers)
  - 3 ...... Flag could not be set (timeout)

**Parameter**

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

Block Symbol

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

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

Input
TRG Trigger of the selected action bool

Outputs
E Error flag bool
iE Error code long
i ..... REX general error

Parameter
action System function to perform ⊙1 long
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
CHAPTER 2. EXEC – REAL-TIME EXECUTIVE CONFIGURATION

TASK – Standard task of the REX control system

Block Symbol Licence: STANDARD

Function Description

The overall control algorithm of the REX control 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 REX control 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 REX control 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 $0 \leq start < stop \leq ntick<i>$ must hold for the start and stop parameters. The RexComp 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 RexView program diagnoses whether the execution delay is occasional or permanent (the Level and Task tabs).

Input

prev Input for connecting the task with the corresponding Level$<i>$ output of the EXEC block or with the next output of the preceding task of the given level
Output

next  Output for connecting to the prev input of the succeeding task in the given level

Parameters

factor  Execution factor; multiple of the execution period of the i-th level of the EXEC block defining the execution period of the task: $\text{factor} \times n\text{tick}_i$  long

start  Number of tick of the given computational level which should trigger the task execution  long

stop  Number of tick of the given computational level by which the task execution should finish  long

stack  Stack size [bytes]  long

filename  Name of the file with the .mdl extension which contains the task algorithm. In the case filename is not specified, the filename is given by the name of the TASK block in the project main file (the .mdl extension is attached automatically)  string
TIODRV – The REX control system input/output driver with tasks

Block Symbol

License: STANDARD

Function Description

The TIODRV block is used for configuration of special drivers of the REX control 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 RexComp 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 RexView program.

Input

prev

Input for connecting the driver with the Drivers output of the EXEC block or with the next output of the preceding driver

Outputs

next

Output for connecting to the prev input of the succeeding driver

Tasks

The IOTASK blocks executed by the driver are connected to this output using the prev input

Parameters

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

classname
Name of the driver class; case sensitive! ○DrvClass string

cfgname
Name of the driver configuration file ○iodrv.rio string

<table>
<thead>
<tr>
<th>factor</th>
<th>Multiple of the EXEC block’s tick parameter defining the driver’s task execution period</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack</td>
<td>Stack size of the driver’s task in bytes</td>
<td>(\downarrow 1024 \circ 10240)</td>
</tr>
<tr>
<td>pri</td>
<td>Logical priority of the driver’s task</td>
<td>(\downarrow 1 \circ 31 \circ 3)</td>
</tr>
<tr>
<td>timer</td>
<td>Driver is a source of time</td>
<td>bool</td>
</tr>
</tbody>
</table>
WWW – Internal Web Server Content

Block Symbol

Licence: STANDARD

Function Description

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

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

Parameters

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

string

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

string

Compression Enables data compression in gzip format. 

bool
Chapter 3

INOUT – Input and output blocks

Contents

| Display – Numeric display of input values | 42 |
| From, INSTD – Signal connection or input | 43 |
| Goto, OUTSTD – Signal source or output | 45 |
| GotoTagVisibility – Visibility of the signal source | 47 |
| Inport, Outport – Input and output port | 48 |
| SubSystem – Subsystem block | 50 |
| INQUAD, INOCT, INHEXD – Multi-input blocks | 51 |
| OUTQUAD, OUTOCT, OUTHEXD – Multi-output blocks | 52 |
| OUTRQUAD, OUTROCT, OUTRHEXD – Multi-output blocks with verification | 53 |
| OUTRSTD – Output block with verification | 55 |
| QFC – Quality flags coding | 56 |
| QFD – Quality flags decoding | 57 |
| VIN – Validation of the input signal | 58 |
| VOUT – Validation of the output signal | 59 |
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 RexDraw even without turning on Watch mode for the block, and the same in WebWatch. Actual conversion of input into its textual representation is performed on the target device in each Decimation period so the value displayed may be also read via the REST interface or used in visualization.

Input

\( u \) Input signal unknown

Parameters

<table>
<thead>
<tr>
<th>Format</th>
<th>Format of displayed value</th>
<th>① long</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best fit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>short</td>
<td></td>
<td></td>
</tr>
<tr>
<td>long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>short_e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>long_e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bank</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>det</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Decimation Value is evaluated in each Decimation period ↓1 ↑100000 ① long

Suffix A string to be appended to the value string
From, INSTD – Signal connection or input

Block Symbols

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.

The From block can be used in both the REX control system and the Matlab-Simulink environment, the INSTD block exists only in the REX control system.

The following rules define how the RexComp 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 of the parameter (substring) before the delimiter (DRV in the example above) is considered to be the name of anIODRV type block contained in the main file of the project. The RexComp 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, A 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 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 RexComp 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.

There is no INSTD block in the Matlab-Simulink system, even the blocks whose GotoTag parameter contains the __ delimiter are considered to be of the From type. This property is suitable for simulation of both the control system and the controlled system. The model can be connected via From and Goto blocks, whose GotoTag parameters include the __ delimiter. Moreover it is possible to use one .mdl file for both simulation and real time control without any modifications if the controlled system model is "hidden" in a subsystem whose name starts with Simulation. The RexComp compiler ignores (omits) such subsystems. For further details see [2].
**Output**

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

**Parameter**

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

Block Symbols

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

The Goto block can be used in both the REX control system and the Matlab-Simulink environment, the OUTSTD block exists only in the REX control system.

The following rules define how the RexComp 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 of the parameter (substring) before the delimiter (DRV in the example above) is considered to be the name of an IODRV type block contained in the main file of the project. The RexComp 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, A 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 RexComp 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.

There is no OUTSTD block in the Matlab-Simulink system, even the blocks whose GotoTag parameter contains the `__` delimiter are considered to be of the Goto type. This property is suitable for simulation of both the control system and the controlled system. The model can be connected via From and Goto blocks, whose GotoTag parameters...
include the __ delimiter. Moreover, it is possible to use one .mdl file for both simulation and real time control without any modifications if the controlled system model is "hidden" in a subsystem whose name starts with Simulation. The RexComp compiler ignores (omits) such subsystems. For further details see [2].

Input

value  Signal going to I/O driver or From block. In case of connection to an I/O driver, the type of input is determined by the I/O driver from the GotoTag parameter.

Parameters

GotoTag  Reference to a From block with the same GotoTag parameter, which should be connected with the Goto block or a reference to output signal of the REX control system driver, which should send the data from block input to the process.

TagVisibility  Visibility (availability) of the block within the .mdl file. Defines conditions under which the two corresponding Goto and From blocks are reciprocally available:

  local  the two blocks must be in the same subsystem
  global  blocks can be anywhere in the given .mdl file
  scoped  the From block must be placed in the same subsystem or in any lower hierarchical level below the GotoTagVisibility block with the same GotoTag parameter
GotoTagVisibility – Visibility of the signal source

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 configuration compilation by the RexComp compiler, it is not included in the final configuration as it does not perform any action in real-time.

Parameter

GotoTag Reference to a Goto block with the GotoTag parameter, whose string visibility is defined by the position of this block (GotoTagVisibility)
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 REX control 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 RexComp 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 RexDraw 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 REX control system. Detailed description of the blocks for Matlab-Simulink can be found in [3].

Input

value Value going to the output pin or Inport unknown

Output

value Value coming from the input pin or Outport unknown
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>Ordering of the Inport or Outport pins</td>
<td>long</td>
</tr>
</tbody>
</table>
SubSystem – Subsystem block

Block Symbol

Licence: STANDARD

Function Description

The Subsystem block is a cornerstone of hierarchical control (and 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 REX control system. Therefore it is sometimes referred to as sequence. All blocks from the surroundings of the subsystem are executed strictly before or after the whole subsystem is executed. This is called atomic subsystem in the Matlab-Simulink terminology, see [3].

There are two possible ways of creating a subsystem in both the RexDraw program and the Matlab-Simulink editor (only the RexDraw technique is described further):

- 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 Import 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 Import and Outport blocks for signals crossing the subsystem boundary. Ports for all unconnected inputs and outputs are created as well.

Inputs

The number and names of the inputs are given by the number and names of the Import blocks contained within the subsystem.

Outputs

The number and names of the outputs are given by the number and names of the Outport blocks contained within the subsystem.
INQUAD, INOCT, INHEXD – Multi-input blocks

Block Symbols

 Licence: STANDARD

Function Description

The REX control 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). 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.

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.

Outputs

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

Function Description

The REX control 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.

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.

Inputs

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

Function Description

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

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

- Through the value of the output, which can e.g. contain the real bit value in case of exceeding the limits of 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 rawi outputs are not always refreshed right at the moment of block execution, there is some delay given by the properties of the driver, communication line and/or target platform.

These blocks are not included in the RexLib function block library for Matlab-Simulink.

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

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

Block Symbol

\[
\text{OUTRSTD} \quad \begin{cases} \text{value} \\ \text{raw} \end{cases}
\]

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.

This block is not included in the RexLib function block library for Matlab-Simulink.

Input

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

Output

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

Block Symbol

![QFC 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 REX control system. Further details about quality flags can be found in chapter 1.4 of this manual. The RexLib function block library for Matlab-Simulink does not use any quality flags.

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

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iq</td>
<td>Basic quality type flags, see table 1.2, page 15</td>
</tr>
<tr>
<td>is</td>
<td>Substatus flags, see [1]</td>
</tr>
<tr>
<td>il</td>
<td>Limits flags, see [1]</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iqf</td>
<td>Bit combination of the iq, is and il input signals</td>
</tr>
</tbody>
</table>

long
QFD – Quality flags decoding

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 REX control system. Further details about quality flags can be found in chapter 1.4 of this manual. The RexLib function block library for Matlab-Simulink does not use any quality flags.

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

Input

iqf Quality flags to be decomposed to iq, is and il components long

Outputs

iq Basic quality type flags, see table 1.2, page 15 long
is Substatus flags, see [1] long
il Limits flags, see [1] long
CHAPTER 3. INOUT – INPUT AND OUTPUT BLOCKS

VIN – Validation of the input signal

Function Description

The VIN block can be used for verification of the input signal quality in the REX control 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 \( GU = \text{off} \) the output \( QG \) is set to \( \text{on} \) if the quality is \( \text{GOOD} \). It is set to \( QG = \text{off} \) in case of \( \text{BAD} \) or \( \text{UNCERTAIN} \) quality.
- For \( GU = \text{on} \) the output \( QG \) is set to \( \text{on} \) if the quality is \( \text{GOOD} \) or \( \text{UNCERTAIN} \). It is set to \( QG = \text{on} \) only in case of \( \text{BAD} \) quality.

The output \( yg \) is equal to the \( u \) input if \( QQ = \text{on} \). Otherwise it is set to \( yg = sv \) (substitution variable).

Inputs

- \( u \): Input signal whose quality is assessed. The type of the signal is \( \text{unknown} \) determined upon the connected signal.
- \( sv \): Substitute value for an error case

Outputs

- \( yg \): Validated output signal (\( yg = u \) for \( QQ = \text{on} \) or \( yg = sv \) for \( QQ = \text{off} \)) \( \text{unknown} \)
- \( QQ \): Indicator of input signal acceptability \( \text{bool} \)
- \( iqf \): Complete quality flag separated from the \( u \) input signal \( \text{long} \)

Parameter

- \( GU \): Acceptability of \( \text{UNCERTAIN} \) quality \( \text{bool} \)
  - \( \text{off} \ldots \) Uncertain quality unacceptable
  - \( \text{on} \ldots \) Uncertain quality acceptable
**VOUT – Validation of the output signal**

**Block Symbol**

![VOUT Block Symbol](image)

**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**

- **u**: Input signal whose quality flags are being replaced. The type of the signal is determined upon the connected signal.
- **iqf**: Desired quality flags

**Output**

- **yq**: Resulting signal composed from input u and quality flags given by the iqf input.
Chapter 4

MATH – Math blocks

Contents

ABS_ – Absolute value ........................................... 63
ADD – Addition of two signals ................................. 64
ADDOCT – Addition of eight signals ....................... 65
CNB – Boolean (logic) constant ............................. 66
CNE – Enumeration constant ................................ 67
CNI – Integer constant ........................................... 68
CNR – Real constant ............................................. 69
DIF_ – Difference ................................................ 70
DIV – Division of two signals ................................. 71
EAS – Extended addition and subtraction ................ 72
EMD – Extended multiplication and division ............ 73
FNX – Evaluation of single-variable function .......... 74
FNXY – Evaluation of two-variables function ........... 76
GAIN – Multiplication by a constant ....................... 78
GRADS – Gradient search optimization ................. 79
IADD – Integer addition ...................................... 81
ISUB – Integer subtraction ................................... 83
IMUL – Integer multiplication ................................ 85
IDIV – Integer division ........................................ 87
IMOD – Remainder after integer division ............... 88
LIN – Linear interpolation .................................... 89
MUL – Multiplication of two signals ..................... 90
POL – Polynomial evaluation ................................. 91
REC – Reciprocal value ....................................... 92
REL – Relational operator .................................... 93
RTOI – Real to integer number conversion ............. 94
<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQR</td>
<td>Square value</td>
<td>95</td>
</tr>
<tr>
<td>SQRT_</td>
<td>Square root</td>
<td>96</td>
</tr>
<tr>
<td>SUB</td>
<td>Subtraction of two signals</td>
<td>97</td>
</tr>
</tbody>
</table>
ABS_ – Absolute value

Block Symbol

Function Description

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

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

Input

\( u \) Analog input of the block double

Outputs

\( y \) Absolute value of the input signal double
\( \text{sgn} \) Indication of the input signal sign long
ADD – Addition of two signals

Block Symbol

![Block Symbol]

Licence: STANDARD

Function Description

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

\[ y = u_1 + u_2. \]

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

Inputs

- \( u_1 \)  
  First analog input of the block  
  double

- \( u_2 \)  
  Second analog input of the block  
  double

Output

- \( y \)  
  Sum of the input signals  
  double
**ADDOCT — Addition of eight signals**

Block Symbol

```
  +-------------+      Licence: STANDARD
  |             |        |
  |   u1        |        |
  |   u2        |        |
  |   u3        |        |
  |   u4        |        |
  |   u5        |        |
  |   u6        |        |
  |   u7        |        |
  |   u8        |        |
  |   y         |        |
  +-------------+      ADDOCT
```

**Function Description**

The **ADDOCT** block sums (or subtracts) up to 8 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 + u8`. For e.g. `n1=2,5,7`, the block implements the function `y = u1 - u2 + u3 + u4 - u5 + u6 - u7 + u8`.

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

**Inputs**

<table>
<thead>
<tr>
<th>Name (u1, u2, u3, u4, u5, u6, u7, u8)</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>First analog input of the block</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>Second analog input of the block</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>Third analog input of the block</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>Fourth analog input of the block</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>Fifth analog input of the block</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>Sixth analog input of the block</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>Seventh analog input of the block</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>Eighth analog input of the block</td>
<td>double</td>
<td></td>
</tr>
</tbody>
</table>

**Output**

<table>
<thead>
<tr>
<th>Name (y)</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum of the input signals</td>
<td>double</td>
<td></td>
</tr>
</tbody>
</table>

**Parameter**

<table>
<thead>
<tr>
<th>Name (n1)</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of signals to subtract instead of adding. The format of the list is e.g. 1,3..5,8. Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.</td>
<td>long</td>
<td></td>
</tr>
</tbody>
</table>
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 bool
   off ... Disabled
   on .... Enabled
CNE – Enumeration constant

Block Symbol

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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>yenum</code></td>
<td>Enumeration constant</td>
<td>string</td>
</tr>
<tr>
<td><code>pupstr</code></td>
<td>Popup list definition</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>1: option A</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>2: option B</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>3: option C</td>
<td>string</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iy</code></td>
<td>Integer output of the block</td>
</tr>
<tr>
<td></td>
<td>long</td>
</tr>
</tbody>
</table>
CNI — Integer constant

Block Symbol

Function Description

The CNI block stands for an integer constant.

Output

- $i_y$ Integer output of the block $\text{long}$

Parameter

- $i_c n$ Integer constant $\odot 1 \text{ long}$
CNR – **Real constant**

**Block Symbol**

![Block Symbol](image)

**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

\[ \text{DIFF} \]

Function Description

The DIF\_ block differentiates the input signal \( u \) according to the following formula

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

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

Input

\( u \)  
Analog input of the block  
double

Output

\( y \)  
Difference of the input signal  
double

Parameters

\( \text{ISSF} \)  
Zero output at start-up  
bool

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

Block Symbol

\[
\begin{bmatrix}
  u_1 \\ u_2 \\ y \\ E
\end{bmatrix}
\]

Licence: STANDARD

Function Description

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

Inputs

- \( u_1 \): First analog input of the block, double
- \( u_2 \): Second analog input of the block, double

Outputs

- \( y \): Quotient of the inputs, double
- \( E \): Error flag – division by zero, bool

Parameter

- \( y_{err} \): Substitute value for an error case, \( \odot 1.0 \), double
EAS – Extended addition and subtraction

Block Symbol

Function Description

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

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

Inputs

- $u_1$: First analog input of the block  
- $u_2$: Second analog input of the block  
- $u_3$: Third analog input of the block  
- $u_4$: Fourth analog input of the block

Output

- $y$: Analog output of the block

Parameters

- $a$: Weighting coefficient of the $u_1$ input  
- $b$: Weighting coefficient of the $u_2$ input  
- $c$: Weighting coefficient of the $u_3$ input  
- $d$: Weighting coefficient of the $u_4$ input  
- $y_0$: Additive constant (bias)
EMD – Extended multiplication and division

Block Symbol

Function Description

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

$$y = \frac{(a \cdot u_1 + a_0)(b \cdot u_2 + b_0)}{(c \cdot u_3 + c_0)(d \cdot u_4 + d_0)}.$$ (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 $y = y_{err}$.

Inputs

- $u_1$: First analog input of the block, double
- $u_2$: Second analog input of the block, double
- $u_3$: Third analog input of the block, double
- $u_4$: Fourth analog input of the block, double

Outputs

- $y$: Analog output of the block, double
- $E$: Error flag – division by zero, bool

Parameters

- $a$: Weighting coefficient of the $u_1$ input, $\odot 1.0$ double
- $a_0$: Additive constant for $u_1$ input, double
- $b$: Weighting coefficient of the $u_2$ input, $\odot 1.0$ double
- $b_0$: Additive constant for $u_2$ input, double
- $c$: Weighting coefficient of the $u_3$ input, $\odot 1.0$ double
- $c_0$: Additive constant for $u_3$ input, double
- $d$: Weighting coefficient of the $u_4$ input, $\odot 1.0$ double
- $d_0$: Additive constant for $u_4$ input, double
- $y_{err}$: Substitute value for an error case, $\odot 1.0$ double
FNX – Evaluation of single-variable function

Block Symbol

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:

<table>
<thead>
<tr>
<th>ifn: shortcut</th>
<th>function</th>
<th>constraints on u</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>acos</td>
<td>u ∈&lt; −1.0, 1.0 &gt;</td>
</tr>
<tr>
<td>2:</td>
<td>asin</td>
<td>u ∈&lt; −1.0, 1.0 &gt;</td>
</tr>
<tr>
<td>3:</td>
<td>atan</td>
<td></td>
</tr>
<tr>
<td>4:</td>
<td>ceil</td>
<td>rounding towards the nearest higher integer</td>
</tr>
<tr>
<td>5:</td>
<td>cos</td>
<td></td>
</tr>
<tr>
<td>6:</td>
<td>cosh</td>
<td></td>
</tr>
<tr>
<td>7:</td>
<td>exp</td>
<td>exponential function e^u</td>
</tr>
<tr>
<td>8:</td>
<td>exp10</td>
<td>exponential function 10^u</td>
</tr>
<tr>
<td>9:</td>
<td>fabs</td>
<td>absolute value</td>
</tr>
<tr>
<td>10:</td>
<td>floor</td>
<td>rounding towards the nearest lower integer</td>
</tr>
<tr>
<td>11:</td>
<td>log</td>
<td>logarithm</td>
</tr>
<tr>
<td>12:</td>
<td>log10</td>
<td>decimal logarithm</td>
</tr>
<tr>
<td>13:</td>
<td>random</td>
<td>arbitrary number z ∈&lt; 0, 1 &gt; (u independent)</td>
</tr>
<tr>
<td>14:</td>
<td>sin</td>
<td>sine</td>
</tr>
<tr>
<td>15:</td>
<td>sinh</td>
<td>hyperbolic sine</td>
</tr>
<tr>
<td>16:</td>
<td>sqr</td>
<td>square function</td>
</tr>
<tr>
<td>17:</td>
<td>sqrt</td>
<td>square root</td>
</tr>
<tr>
<td>18:</td>
<td>srand</td>
<td>changes the seed for the random function to u</td>
</tr>
<tr>
<td>19:</td>
<td>tan</td>
<td>tangent</td>
</tr>
<tr>
<td>20:</td>
<td>tanh</td>
<td>hyperbolic tangent</td>
</tr>
</tbody>
</table>

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  \( \text{double} \)

Outputs

\( y \)  Result of the selected function  \( \text{double} \)
\( E \)  Error flag  \( \text{bool} \)

Parameters

\( \text{ifn} \)  Function type (see table above)  \( \odot 1 \) \( \text{long} \)
\( \text{yerr} \)  Substitute value for an error case  \( \text{double} \)
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:

<table>
<thead>
<tr>
<th>ifn: shortcut</th>
<th>function</th>
<th>constraints on u1, u2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: atan2</td>
<td>arctangent u1/u2</td>
<td></td>
</tr>
<tr>
<td>2: fmod</td>
<td>remainder after division u1/u2</td>
<td>u2 ≠ 0.0</td>
</tr>
<tr>
<td>3: pow</td>
<td>exponentiation of the inputs y = u1u2</td>
<td></td>
</tr>
</tbody>
</table>

The atan2 function result belongs to the interval $(-\pi, \pi)$. 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 ($E = \text{on}$) 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 ($y = \text{yerr}$).

Inputs

<table>
<thead>
<tr>
<th>u1</th>
<th>First analog input of the block</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>u2</td>
<td>Second analog input of the block</td>
<td>double</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>y</th>
<th>Result of the selected function</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Error flag</td>
<td>bool</td>
</tr>
<tr>
<td>off ...</td>
<td>No error</td>
<td></td>
</tr>
<tr>
<td>on ....</td>
<td>An error occurred</td>
<td></td>
</tr>
</tbody>
</table>
### Parameters

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

Block Symbol

\[
\begin{array}{c}
u \\
GAIN \\
y
\end{array}
\]

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 \( \circ1.0 \) double
GRADS – Gradient search optimization

Function Description

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

$$grad = (f_i - f_{i-1}) \cdot (dx)_{i-1}$$

$$(dx)_i = -\gamma \cdot grad,$$

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

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

Inputs

- $f$ Value of the optimized $f(.)$ for given variable $x$ double
- $x_0$ Optimization starting point double
- START Starting signal (rising edge) bool
- BRK Termination signal bool

Outputs

- $x$ Current value of the optimized variable double
- $x_{opt}$ Resulting optimal value of the $x$ variable double
- $f_{opt}$ Resulting optimal value of the function $f(x, v)$ double
- BSY Indicator of running optimization bool
- iter Number of current iteration long
- $E$ Error flag bool
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmin</td>
<td>Lower limit for the x variable</td>
<td>double</td>
</tr>
<tr>
<td>xmax</td>
<td>Upper limit for the x variable</td>
<td>double</td>
</tr>
<tr>
<td>gamma</td>
<td>Coefficient for determining the step size of the gradient optimization method</td>
<td>double</td>
</tr>
<tr>
<td>d0</td>
<td>Initial step size</td>
<td>0.05</td>
</tr>
<tr>
<td>dmin</td>
<td>Minimum step size</td>
<td>0.01</td>
</tr>
<tr>
<td>dmax</td>
<td>Maximum step size</td>
<td>1.0</td>
</tr>
<tr>
<td>n</td>
<td>Iteration period (in sampling periods $T_S$)</td>
<td>100</td>
</tr>
<tr>
<td>itermax</td>
<td>Maximum number of iterations</td>
<td>20</td>
</tr>
</tbody>
</table>
**IADD – Integer addition**

**Block Symbol**

```
  i1  n  E
  i2  IADD
```

**Licence:** STANDARD

**Function Description**

The IADD block sums two integer input signals \( n = i_1 + i_2 \). The range of integer numbers in a computer is always restricted by the variable type. This block uses the \( \text{vtype} \) parameter to specify the type. If the sum fits in the range of the given type, the result is the ordinary sum. In the other cases the result depends on the \( \text{SAT} \) parameter.

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

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

**Inputs**

- **\( i_1 \)**: First integer input of the block
  - \(-9.22E+18 \leq i_1 \leq 9.22E+18\) long

- **\( i_2 \)**: Second integer input of the block
  - \(-9.22E+18 \leq i_2 \leq 9.22E+18\) long

**Outputs**

- **\( n \)**: Integer sum of the input signals
  - long

- **\( E \)**: Error flag
  - \( \text{off} \ldots \) No error
  - \( \text{on} \ldots \) An error occurred

**Parameters**

- **\( \text{vtype} \)**: Numeric type
  - \( \circ 4 \) long
    - \( 2 \ldots \) Byte (range 0 ... 255)
    - \( 3 \ldots \) Short (range -32768 ... 32767)
    - \( 4 \ldots \) Long (range -2147483648 ... 2147483647)
    - \( 5 \ldots \) Word (range 0 ... 65536)
    - \( 6 \ldots \) DWord (range 0 ... 4294967295)
    - \( 10 \ldots \) 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 \( n = i_1 - i_2 \). The range of integer numbers in a computer is always restricted by the variable type. This block uses the \( vtype \) parameter to specify the type. If the 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 \( SAT = \text{off} \), i.e. the output \( E = \text{off} \) and the output value \( n \) corresponds with the arithmetics of the processor. E.g. for the Short type, which has the range of \(-32768..+32767\), we obtain \( 30000 - -2770 = -32766 \).

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

Inputs

| \( i_1 \) | First integer input of the block | \( \downarrow-9.22\times10^{18} \uparrow9.22\times10^{18} \) long |
| \( i_2 \) | Second integer input of the block | \( \downarrow-9.22\times10^{18} \uparrow9.22\times10^{18} \) long |

Parameters

<table>
<thead>
<tr>
<th>( vtype )</th>
<th>Numeric type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>.. Byte (range 0 ... 255)</td>
</tr>
<tr>
<td>3</td>
<td>.. Short (range -32768 ... 32767)</td>
</tr>
<tr>
<td>4</td>
<td>.. Long (range -2147483648 ... 2147483647)</td>
</tr>
<tr>
<td>5</td>
<td>.. Word (range 0 ... 65536)</td>
</tr>
<tr>
<td>6</td>
<td>.. DWord (range 0 ... 4294967295)</td>
</tr>
<tr>
<td>10</td>
<td>.. Large (range (-9.223372036854775808\ldots9.223372036854775807)</td>
</tr>
</tbody>
</table>

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

Outputs

| \( n \) | Integer difference between the input signals | long |
Error flag

- off ... No error
- on .... An error occurred
**IMUL – Integer multiplication**

**Function Description**

The IMUL block multiplies two integer input signals \( n = i1 \cdot i2 \). The range of integer numbers in a computer is always restricted by the variable type. This block uses the \( v\text{type} \) 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 \( \text{SAT} \) parameter.

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

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

**Inputs**

- \( i1 \) First integer input of the block \([-9,22E+18, 9,22E+18]\) long
- \( i2 \) Second integer input of the block \([-9,22E+18, 9,22E+18]\) long

**Parameters**

- \( v\text{type} \) Numeric type
  - 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)

- \( \text{SAT} \) Saturation (overflow) checking
  - \( \text{off} \) . . . . Overflow is not checked
  - \( \text{on} \) . . . . 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**

![Block Symbol](image.png)

**License:** STANDARD

**Function Description**

The IDIV block performs an integer division of two integer input signals, \( n = \frac{i1}{i2} \), 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 \( n = nerr \).

**Inputs**

- \( i1 \): First integer input of the block
  - \([-9,22E+18, 9,22E+18]\) long
- \( i2 \): Second integer input of the block
  - \([-9,22E+18, 9,22E+18]\) long

**Outputs**

- \( n \): Integer quotient of the inputs
  - long
- \( E \): Error flag – division by zero
  - bool

**Parameters**

- \( vtype \): Numeric type
  - 2 ...... Byte
  - 3 ...... Short
  - 4 ...... Long
  - 5 ...... Word
  - 6 ...... DWord
  - 10 .... Large
  - \( \odot 4 \) long
- \( nerr \): Substitute value for an error case
  - \( \odot 1 \) long
IMOD – Remainder after integer division

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

- \( i1 \): First integer input of the block
  - Value range: \(-9,22E+18 \) to \( 9,22E+18 \)

- \( i2 \): Second integer input of the block
  - Value range: \(-9,22E+18 \) to \( 9,22E+18 \)

Outputs

- \( n \): Remainder after integer division
  - Type: long

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

Parameters

- \( vtype \): Numeric type
  - Options: Byte, Short, Long, Word, DWord, Large
  - Value range: \( 1 \) to \( 10 \)

- \( nerr \): Substitute value for an error case
  - Value range: \( 1 \)

Licence: STANDARD
LIN – Linear interpolation

Block Symbol

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

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  $\odot$1.0  double
$y_2$  y-coordinate of the 2nd interpolation node  $\odot$1.0  double
**MUL – Multiplication of two signals**

**Block Symbol**

![Block Symbol]

**Licence:** STANDARD

**Function Description**

The MUL block multiplies two analog input signals $y = u_1 \cdot u_2$.

**Inputs**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>First analog input of the block</td>
<td>double</td>
</tr>
<tr>
<td>$u_2$</td>
<td>Second analog input of the block</td>
<td>double</td>
</tr>
</tbody>
</table>

**Output**

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>Product of the input signals</td>
<td>double</td>
</tr>
</tbody>
</table>
**POL – Polynomial evaluation**

Block Symbol

| POL | u | y |

Function Description

The POL block evaluates the polynomial of the form:

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

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

**Input**

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

**Output**

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

**Parameters**

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

Block Symbol

Function Description

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

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

In case \( u = 0 \), the error indicator is set to \( E = \text{on} \) and the output is set to the substitu-
tional value \( y = y_{\text{err}} \).

Input

\( u \) Analog input of the block double

Outputs

\( y \) Analog output of the block double
\( E \) Error flag – division by zero bool

Parameter

\( y_{\text{err}} \) Substitute value for an error case ⊕1.0 double
REL – Relational operator

Block Symbol

Function Description
The REL block evaluates the binary relation \( u_1 \circ u_2 \) 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
\[
\begin{array}{ll}
u_1 & \text{First analog input of the block} \\
u_2 & \text{Second analog input of the block}
\end{array}
\]

Output
\[
\begin{array}{ll}
Y & \text{Logical output indicating whether the relation holds}
\end{array}
\]

Parameter
\[
\begin{array}{ll}
\text{irel} & \text{Relation type} \\
1 & \text{equality (==)} \\
2 & \text{inequality (!=)} \\
3 & \text{less than (<)} \\
4 & \text{greater than (>)} \\
5 & \text{less than or equal to (<=)} \\
6 & \text{greater than or equal to (>=)}
\end{array}
\]

Licence: STANDARD
RT0I – Real to integer number conversion

Block Symbol Licence: STANDARD

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

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

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

Input
\( r \quad \text{Analog input of the block} \quad \text{double} \)

Output
\( i \quad \text{Rounded and converted input signal} \quad \text{long} \)
**SQR – Square value**

Block Symbol

\[
\begin{array}{c}
u \\
\text{SQR} \\
y
\end{array}
\]

Licence: **STANDARD**

Function Description

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

\[ y = u^2. \]

Input

\( u \)  
Analog input of the block  
\text{double}

Output

\( y \)  
Square of the input signal  
\text{double}
**SQRT** – Square root

Block Symbol

![SQRT Block Diagram](image)

Licence: STANDARD

**Function Description**

The **SQRT** block computes the square root of the input \( u \). The output is then

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

In case \( u < 0 \), the error indicator is activated (\( E = \text{on} \)) and the output \( y \) is set to the substitute value \( y = y_{err} \).

**Input**

- \( u \) Analog input of the block \( \text{double} \)

**Outputs**

- \( y \) Square root of the input signal \( \text{double} \)
- \( E \) Error flag \( \text{bool} \)
  - off ... No error
  - on .... Square root of negative number

**Parameter**

- \( y_{err} \) Substitute value for an error case \( \approx 1.0 \) \( \text{double} \)
**SUB – Subtraction of two signals**

**Block Symbol**

![SUB Block Symbol]

**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

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSROT</td>
<td>Processing data from absolute position sensor</td>
<td>101</td>
</tr>
<tr>
<td>ASW</td>
<td>Switch with automatic selection of input</td>
<td>103</td>
</tr>
<tr>
<td>AVG</td>
<td>Moving average filter</td>
<td>105</td>
</tr>
<tr>
<td>AVS</td>
<td>Motion control unit</td>
<td>106</td>
</tr>
<tr>
<td>BPF</td>
<td>Band-pass filter</td>
<td>107</td>
</tr>
<tr>
<td>CMP</td>
<td>Comparator with hysteresis</td>
<td>108</td>
</tr>
<tr>
<td>CNDR</td>
<td>Nonlinear conditioner</td>
<td>109</td>
</tr>
<tr>
<td>DEL</td>
<td>Delay with initialization</td>
<td>111</td>
</tr>
<tr>
<td>DELM</td>
<td>Time delay</td>
<td>112</td>
</tr>
<tr>
<td>DER</td>
<td>Derivation, filtering and prediction from the last n+1 samples</td>
<td>113</td>
</tr>
<tr>
<td>EVAR</td>
<td>Moving mean value and standard deviation</td>
<td>115</td>
</tr>
<tr>
<td>INTE</td>
<td>Controlled integrator</td>
<td>116</td>
</tr>
<tr>
<td>KDER</td>
<td>Derivation and filtering of the input signal</td>
<td>118</td>
</tr>
<tr>
<td>LPF</td>
<td>Low-pass filter</td>
<td>120</td>
</tr>
<tr>
<td>MINMAX</td>
<td>Running minimum and maximum</td>
<td>121</td>
</tr>
<tr>
<td>NSCL</td>
<td>Nonlinear scaling factor</td>
<td>122</td>
</tr>
<tr>
<td>RDFT</td>
<td>Running discrete Fourier transform</td>
<td>123</td>
</tr>
<tr>
<td>RLIM</td>
<td>Rate limiter</td>
<td>125</td>
</tr>
<tr>
<td>S1OF2</td>
<td>One of two analog signals selector</td>
<td>126</td>
</tr>
<tr>
<td>SAI</td>
<td>Safety analog input</td>
<td>129</td>
</tr>
<tr>
<td>SEL</td>
<td>Analog signal selector</td>
<td>132</td>
</tr>
<tr>
<td>SELQUAD, SELRCT, SELHEXD</td>
<td>Analog signal selectors</td>
<td>134</td>
</tr>
<tr>
<td>SHIFTOCT</td>
<td>Data shift register</td>
<td>136</td>
</tr>
<tr>
<td>SHLD</td>
<td>Sample and hold</td>
<td>138</td>
</tr>
</tbody>
</table>
SINT – Simple integrator ............................................. 139
SPIKE – Spike filter .................................................. 140
SSW – Simple switch .................................................. 142
SWR – Selector with ramp .......................................... 143
VDEL – Variable time delay ....................................... 144
ZV4IS – Zero vibration input shaper ............................ 145
ABSROT – Processing data from absolute position sensor

Block Symbol

Function Description

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

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

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

Inputs

<table>
<thead>
<tr>
<th>u</th>
<th>Signal from the absolute position sensor</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Block reset</td>
<td>bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>y</th>
<th>Position output</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>irev</td>
<td>Number of finished revolutions</td>
<td>long</td>
</tr>
<tr>
<td>MPI</td>
<td>Mid-point indicator</td>
<td>bool</td>
</tr>
<tr>
<td>OLI</td>
<td>Off-limits indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>

Parameters

<p>| lolim     | Lower limit of the sensor reading       | -3.141592654 double |</p>
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>hilim</td>
<td>Upper limit of the sensor reading</td>
<td>3.141592654</td>
<td>double</td>
</tr>
<tr>
<td>tol</td>
<td>Tolerance for the mid-point indicator</td>
<td>0.5</td>
<td>double</td>
</tr>
<tr>
<td>hys</td>
<td>Hysteresis for the mid-point indicator</td>
<td></td>
<td>double</td>
</tr>
<tr>
<td>RESR</td>
<td>Flag for resetting the revolutions counter</td>
<td></td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... Reset only the estimated position y</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... Reset also the irev revolutions counter</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**ASW** – Switch with automatic selection of input

**Block Symbol**  
[Diagram of ASW block]

**Licence:** ADVANCED

**Function Description**

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

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

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

**Inputs**

- \( u_1 \ldots u_4 \) Analog input signals to be selected from \( \text{double} \)
- \( iSW \) Active signal or parameter selector \( \text{long} \)

**Outputs**

- \( y \) The selected analog signal or parameter \( \text{double} \)
- \( oSW \) Identifier of the selected signal or parameter \( \text{long} \)
Parameters

\begin{itemize}
    \item \textbf{delta} \hspace{1cm} Threshold for detecting a change \hspace{1cm} $\odot 0.000001$ \hspace{1cm} \textbf{double}
    \item \textbf{p1..p4} \hspace{1cm} Parameters to be selected from \hspace{1cm} \textbf{double}
\end{itemize}
**AVG – Moving average filter**

**Block Symbol**

![AVG Block Symbol](image)

**Licence:** STANDARD

**Function Description**

The AVG block computes a moving average from the last \( n \) samples according to the formula

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

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

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

**Input**

- **\( u \)**: Input signal to be filtered  
  - double

**Output**

- **\( y \)**: Filtered output signal  
  - double

**Parameter**

- **\( n \)**: Number of samples to compute the average from  
  - \( \downarrow 1 \uparrow 10000000 \circ 10 \) long

- **\( n \)**: Limit value of parameter \( n \) (used for internal memory allocation)  
  - \( \downarrow 1 \uparrow 10000000 \circ 10 \) long
AVS – Motion control unit

Block Symbol

Function Description

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

Inputs

- **START** Starting signal (rising edge) \( \text{bool} \)
- **SET** Initialize/compute the trajectory for the current inputs \( \text{bool} \)
- **am** Maximal allowed acceleration \( [\text{m/s}^2] \) \( \text{double} \)
- **dm** Maximal allowed deceleration \( [\text{m/s}^2] \) \( \text{double} \)
- **vm** Maximum allowed velocity \( [\text{m/s}] \) \( \text{double} \)
- **sm** Desired final position \( [\text{m}] \) (initial position is 0) \( \text{double} \)

Outputs

- **a** Acceleration \( [\text{m/s}^2] \) \( \text{double} \)
- **v** Velocity \( [\text{m/s}] \) \( \text{double} \)
- **s** Position \( [\text{m}] \) \( \text{double} \)
- **tt** Time \( [\text{s}] \) \( \text{double} \)
- **RDY** Flag indicating that the block is ready to generate the trajectory \( \text{bool} \)
- **BSY** Flag indicating that the trajectory is being generated \( \text{bool} \)
BPF – Band-pass filter

Block Symbol

\[
\begin{align*}
\begin{array}{c}
\text{u} \\
\text{y}
\end{array}
\end{align*}
\]

Licence: STANDARD

Function Description

The BPF implements a second order filter in the form

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

where \(a\) and \(\xi\) are the block parameters \(fm\) 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

\(u\) Input signal to be filtered double

Output

\(y\) Filtered output signal double

Parameters

\(fm\) Peak frequency, middle of the frequency transmission band [Hz] double \(\odot 1.0\)

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

ISSF Steady state at start-up flag

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

\(\text{on} \ldots\) Initial steady state bool
CMP – Comparator with hysteresis

Block Symbol Licence: STANDARD

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

\[
Y_{-1} = 0, \quad Y_k = hyst(e_k), \quad k = 0, 1, 2, \ldots
\]

where

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

and

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

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

Inputs
- \( u_1 \) First analog input of the block \quad double
- \( u_2 \) Second analog input of the block \quad double

Output
- \( Y \) Logical output of the block \quad bool

Parameter
- \( hys \) Hysteresis \quad \bigcirc 0.5 \quad double
**CNDR – Nonlinear conditioner**

**Block Symbol**

![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  
  $\oplus 2$  
  long
<table>
<thead>
<tr>
<th>SATF</th>
<th>Saturation flag</th>
<th>on</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>off</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Signal not limited</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>on</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Saturation limits active</td>
<td></td>
<td></td>
</tr>
<tr>
<td>up</td>
<td>Vector of increasing u-coordinates</td>
<td>[-1 1]</td>
<td>double</td>
</tr>
<tr>
<td>yp</td>
<td>Vector of y-coordinates</td>
<td>[-1 1]</td>
<td>double</td>
</tr>
</tbody>
</table>
DEL – Delay with initialization

Block Symbol

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 (\( R1: \text{off} \rightarrow \text{on} \rightarrow \text{off} \)) and it is indicated by the output \( \text{RDY} = \text{off} \).

Inputs

- \( u \): Analog input of the block, \( \text{double} \)
- \( R1 \): Block reset, \( \text{bool} \)
- \( y_0 \): Initial output value, \( \text{double} \)

Outputs

- \( y \): Delayed input signal, \( \text{double} \)
- \( \text{RDY} \): Ready flag indicating that the buffer is filled with the input signal samples, \( \text{bool} \)

Parameter

- \( n \): Delay [samples] (the resulting time delay is \( n \cdot T_S \), where \( T_S \) is the block execution period), \( \downarrow 0 \uparrow 10000000 \odot 10 \) \( \text{long} \)
- \( n_{\text{max}} \): Limit for parameter \( \text{del} \) (used for internal memory allocation), \( \downarrow 1 \uparrow 10000000 \odot 10 \) \( \text{long} \)
DELM – Time delay

Block Symbol

\[ \begin{array}{c}
\textbf{DELM} \\
\frac{u}{y}
\end{array} \]

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 \( \text{del} \) seconds after initialization.

Input

\( u \)  
Analog input of the block  
double

Output

\( y \)  
Delayed input signal  
double

Parameter

\( \text{del} \)  
Time delay [s]  
\( \odot 1.0 \)  
double

\( \text{nmax} \)  
Size (number of samples) of delay buffer (used for internal memory allocation)  
\( \downarrow 1 \uparrow 10000000 \odot 10 \)  
long
**DER – Derivation, filtering and prediction from the last n+1 samples**

**Block Symbol**

```
<table>
<thead>
<tr>
<th>u</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>z</td>
</tr>
<tr>
<td>N</td>
<td>RDY</td>
</tr>
<tr>
<td>DER</td>
<td></td>
</tr>
</tbody>
</table>
```

**Licence:** STANDARD

**Function Description**

The DER block interpolates the last n + 1 samples (n ≤ 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:

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

In case of 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**

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

**Outputs**

- **y** Estimate of input signal derivative
- **z** Predicted/filtered input signal
- **RDY** Ready flag (all n + 1 samples are available)
### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>Number of samples for interpolation ((n+1)) samples are used; (1 \leq n \leq n_{\text{max}})</td>
<td>long</td>
<td>(\downarrow 1 \uparrow 10000000 \times 10)</td>
</tr>
<tr>
<td>( n_{\text{max}} )</td>
<td>Limit value for parameter (n) (used for internal memory allocation)</td>
<td>long</td>
<td>(\downarrow 1 \uparrow 10000000 \times 10)</td>
</tr>
</tbody>
</table>
EVAR – Moving mean value and standard deviation

Block Symbol

Function Description

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

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

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

where \( k \) stands for the current sampling instant.

Input

\( u \) Analog input of the block double

Outputs

\( \mu \) Mean value of the input signal double

\( \sigma \) Standard deviation of the input signal double

Parameter

\( n \) Number of samples to estimate the statistical properties from long

\( n_{\text{max}} \) Limit value of parameter \( n \) (used for internal memory allocation) long

Licence: STANDARD
CHAPTER 5. ANALOG – ANALOG SIGNAL PROCESSING

INTE – Controlled integrator

Block Symbol

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 \( C = y_0 \). 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

- u: Analog input of the block  
  - Type: double
- RUN: Enable execution  
  - Values: on, off
  - Meaning: Integration running, Integration stopped
- R1: Block reset, initialization of the integrator output to y0  
  - Type: bool
- y0: Initial output value  
  - Type: double
- ti: Integral time constant  
  - Type: double

Outputs

- y: Integrator output  
  - Type: double
- Q: Running integration indicator  
  - Type: bool
- LY: Lower level indicator (y < ymin)  
  - Type: bool
- HY: Upper level indicator (y > ymax)  
  - Type: bool
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ymin</td>
<td>Lower level definition</td>
<td>$\leq -1.0$</td>
<td>double</td>
</tr>
<tr>
<td>ymax</td>
<td>Upper level definition</td>
<td>$\leq 1.0$</td>
<td>double</td>
</tr>
</tbody>
</table>
**KDER – Derivation and filtering of the input signal**

**Block Symbol**

![Block Diagram](image)

**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 \( pbeta \approx T_S \omega_0 \) 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**

- **u** Input signal to be filtered \( \text{double} \)

**Outputs**

- **y** Filtered input signal \( \text{double} \)
- **dy** Estimated 1st order derivative \( \text{double} \)
- **d2y** Estimated 2nd order derivative \( \text{double} \)
- **d3y** Estimated 3rd order derivative \( \text{double} \)
- **d4y** Estimated 4th order derivative \( \text{double} \)
- **d5y** Estimated 5th order derivative \( \text{double} \)
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>norder</td>
<td>Order of the derivative filter</td>
<td>↓2 ↑10 ⊙3</td>
<td>long</td>
</tr>
<tr>
<td>pbeta</td>
<td>Bandwidth of the derivative filter</td>
<td>↓0.0 ⊙0.1</td>
<td>double</td>
</tr>
</tbody>
</table>
LPF – Low-pass filter

Block Symbol

\[
\begin{array}{c}
\text{LPF} \\
\end{array}
\]

Function Description

The LPF block implements a second order filter in the form

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

where

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

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

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

Input

\( u \) Input signal to be filtered double

Output

\( y \) Filtered output signal double

Parameters

\( f_b \) Filter bandwidth [Hz]; the frequencies in the range \( \langle 0, f_b \rangle \) pass through the filter, the attenuation at the frequency \( f_b \) is 3 dB and approximately 40 dB at \( 10 \cdot f_b \); it must hold \( f_b \leq \frac{1}{2\pi T_S} \) for proper function of the filter, where \( T_S \) is the block execution period \( \odot 1.0 \) double

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

\( \text{ISSF} \) Steady state at start-up

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

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

Block Symbol

Function Description

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

Inputs

| \( u \) | Analog input of the block | double |
| \( R1 \) | Block reset | bool |

Outputs

| \( ymin \) | Minimal value found | double |
| \( ymax \) | Maximal value found | double |
| \( RDY \) | Ready flag (buffer filled) | bool |

Parameter

| \( n \) | Number of samples for analysis (buffer length) | \( \downarrow 1 \uparrow 10000000 \odot 100 \) | long |
**NSCL – Nonlinear scaling factor**

**Block Symbol**

\[
\begin{array}{c}
y \\
\text{NSCL} \\
u
\end{array}
\]

**Licence:** STANDARD

**Function Description**

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

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

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**

\[
u \quad \text{Analog input of the block} \quad \text{double}
\]

**Output**

\[
y \quad \text{Analog output of the block} \quad \text{double}
\]

**Parameters**

\[
\begin{array}{ll}
gain & \text{Signal gain} \quad \circ1.0 \quad \text{double} \\
ze & \text{Shaping parameter} \quad \circ1.0 \quad \text{double}
\end{array}
\]
RDFT – Running discrete Fourier transform

Function Description

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

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

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

Use the VTOR function block to disassemble the vector signals.

Inputs
- \( u \) Analog input of the block
- \( \text{HLD} \) Hold

Outputs
- \( \text{amp} \) Amplitude of the fundamental frequency
- \( \text{thd} \) Total harmonic distortion (only for \( n\text{harm} \geq 1 \))
- \( \text{vAmp} \) Vector of amplitudes at given frequencies
- \( \text{vPhi} \) Vector of phase-shifts at given frequencies
- \( \text{vRe} \) Vector of real parts at given frequencies
- \( \text{vIm} \) Vector of imaginary parts at given frequencies
- \( \text{E} \) Error flag
- \( \text{iE} \) Error code

\( i \ldots \) REX general error
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Min</th>
<th>Max</th>
<th>Default</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>freq</td>
<td>Fundamental frequency</td>
<td>(0.000000001)</td>
<td>(1000000000.0)</td>
<td>(1.0)</td>
<td>double</td>
</tr>
<tr>
<td>nper</td>
<td>Number of periods to calculate upon</td>
<td>(1)</td>
<td>(10000)</td>
<td>(10)</td>
<td>long</td>
</tr>
<tr>
<td>nharm</td>
<td>Number of monitored harmonic frequencies</td>
<td>(0)</td>
<td>(16)</td>
<td>(3)</td>
<td>long</td>
</tr>
<tr>
<td>ifrunit</td>
<td>Frequency units</td>
<td>(1)</td>
<td>(2)</td>
<td>(1)</td>
<td>long</td>
</tr>
<tr>
<td>iphunit</td>
<td>Phase shift units</td>
<td>(0)</td>
<td>(2)</td>
<td>(1)</td>
<td>long</td>
</tr>
<tr>
<td>freq2</td>
<td>Vector of user-defined monitored frequencies</td>
<td>([2.0\ 3.0\ 4.0])</td>
<td>double</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**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: \(1/tp\)
- the steepest descent per second: \(-1/tn\)

**Input**

- `u` Input signal to be filtered \(\text{double}\)

**Output**

- `y` Filtered output signal \(\text{double}\)

**Parameters**

- `tp` Time constant defining the maximum allowed rise \(\geq 2.0\) \(\text{double}\)
- `tn` Time constant defining the maximum allowed descent (note that `tn > 0`) \(\geq 2.0\) \(\text{double}\)
S1OF2 – One of two analog signals selector

Function Description

The S1OF2 block assesses the validity of two input signals \( u_1 \) and \( u_2 \) separately. The validation method is equal to the method used in the SAI block. If the signal \( u_1 \) (or \( u_2 \)) is marked invalid, the output \( E_1 \) (or \( E_2 \)) is set to on and the error code is sent to the \( iE_1 \) (or \( iE_2 \)) output. The S1OF2 block also evaluates the difference between the two input signals. The internal flag \( D \) is set to on if the differences \( |u_1 - u_2| \) in the last \( n_d \) samples exceed the given limit, which is given by the following inequation

\[
|u_1 - u_2| > pdev \frac{v_{max} - v_{min}}{100},
\]

where \( v_{min} \) and \( v_{max} \) are the minimal and maximal limits of the inputs \( u_1 \) and \( u_2 \) 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 \( E_1 \) and \( E_2 \)) and the internal difference flag \( D \) as follows:

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

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

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

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

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

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

The input \( R \) resets the inner error flags \( F_1 - F_4 \) (see the SAI block) and the \( D \) flag. For the input \( R \) set permanently to on, the invalidity indicator \( E_1 \) (\( E_2 \)) is set to on for only...
one cycle period whenever some invalidity condition is fulfilled. On the other hand, for $R = 0$, the output $E_1$ ($E_2$) is set to on and remains true until the reset ($R$: off $\rightarrow$ 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 ($D = \text{off} \rightarrow \text{on}$). On the other hand, for $R = 0$, the output $E$ is set to on and remains true until the reset (rising edge $R$: off $\rightarrow$ on). The output $W$ is set to on only in the (iii) or (iv) cases, i.e. at least one input signal is invalid.

### Inputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>First analog input of the block</td>
</tr>
<tr>
<td>$u_2$</td>
<td>Second analog input of the block</td>
</tr>
<tr>
<td>$sv$</td>
<td>Substitute value for an error case, i.e. $E = \text{on}$</td>
</tr>
<tr>
<td>$HF_1$</td>
<td>Hardware error flag for signal $u_1$</td>
</tr>
<tr>
<td>$HF_2$</td>
<td>Hardware error flag for signal $u_2$</td>
</tr>
<tr>
<td>$R$</td>
<td>Reset inner error flags of the input signals $u_1$ and $u_2$</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>Analog output of the block</td>
</tr>
<tr>
<td>$E$</td>
<td>Output signal invalidity indicator</td>
</tr>
<tr>
<td>$E_1$</td>
<td>Invalidity indicator for input $u_1$</td>
</tr>
<tr>
<td>$E_2$</td>
<td>Invalidity indicator for input $u_2$</td>
</tr>
<tr>
<td>$iE_1$</td>
<td>Reason of input $u_1$ invalidity</td>
</tr>
<tr>
<td>$iE_2$</td>
<td>Reason of input $u_2$ invalidity, see the $iE_1$ output</td>
</tr>
<tr>
<td>$W$</td>
<td>Warning flag (invalid input signal)</td>
</tr>
</tbody>
</table>
CHAPTER 5. ANALOG – ANALOG SIGNAL PROCESSING

Parameters

- **nb**: Number of samples which are not included in the validity assessment of the signals $u_1$ and $u_2$ after initialization of the block. **Type**: long **Length**: 10

- **nc**: Number of samples for invariability testing (see the SAI block, condition $F_2$). **Type**: long **Length**: 10

- **nbits**: Number of A/D converter bits (source of the signals $u_1$ and $u_2$). **Type**: long **Length**: 12

- **nr**: Number of samples for variability testing (see the SAI block, condition $F_3$). **Type**: long **Length**: 10

- **prate**: Maximum allowed percentage change of the input $u_1$ ($u_2$) within the last $nr$ samples (with respect to the overall range of the input signals $v_{max}$ – $v_{min}$, see the SAI block). **Type**: double **Length**: 10.0

- **nv**: Number of samples for out-of-range testing (see the SAI block, condition $F_4$). **Type**: long **Length**: 1

- **vmin**: Lower limit for the input signals $u_1$ and $u_2$. **Type**: double **Length**: -1.0

- **vmax**: Upper limit for the input signals $u_1$ and $u_2$. **Type**: double **Length**: 1.0

- **nd**: Number of samples for deviation testing (inner flag $D$; $D$ is always off for $nd = 0$). **Type**: long **Length**: 5

- **pdev**: Maximum allowed percentage deviation of the inputs $u_1$ and $u_2$ with respect to the overall range of the input signals $v_{max}$ – $v_{min}$. **Type**: double **Length**: 10.0

- **mode**: Defines how to compute the output signal $y$ when both input signals are valid ($E_1 = \text{off}$, $E_2 = \text{off}$ and $D = \text{off}$). **Type**: long **Length**: 1
  1. Average, $y = \frac{u_1 + u_2}{2}$
  2. Minimum, $y = \min(u_1, u_2)$
  3. Maximum, $y = \max(u_1, u_2)$
SAI – Safety analog input

Block Symbol

Function Description

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

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

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

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

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

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

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

\[
\text{rate} = \text{prate} \frac{v_{\text{max}} - v_{\text{min}}}{100},
\]
where \textit{prate} defines the allowed percentage change in the input signal \( u \) within the last \( \text{nr} \) samples (with respect to the overall range of the input signal \( u \in \langle \text{vmin}, \text{vmax} \rangle \)). The block includes a \text{SPIKE} filter with fixed parameters \( \text{mingap} = \frac{\text{vmax} - \text{vmin}}{100} \) and \( q = 2 \) suppressing peaks in the input signal to avoid undesirable fulfilling of this condition. See the \text{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 \( \text{nr} \) is set to \( \text{nr} = 0 \), the condition \( \text{F3} \) is never fulfilled.

F4: \textbf{The input signal \( u \) is out of range.} The last \( \text{nv} \) samples of the input signal \( u \) lie out of the allowed range \( \langle \text{vmin}, \text{vmax} \rangle \).

If the parameter \( \text{nv} \) is set to \( \text{nv} = 0 \), the condition \( \text{F4} \) 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 \( \text{sv} \) input. In such a case the output \( \text{E} \) is set to \textit{on} and the output \( \text{iE} \) provides the error code. The input \( \text{R} \) resets the inner error flags \( \text{F1–F4} \). For the input \( \text{R} \) set permanently to \textit{on}, the invalidity indicator \( \text{E} \) is set to \textit{on} for only one cycle period whenever some invalidity condition is fulfilled. On the other hand, for \( \text{R} = \text{off} \), the output \( \text{E} \) is set to \textit{on} and remains true until the reset (rising edge \( \text{R}: \text{off} \rightarrow \text{on} \)).

The table of error codes \( \text{iE} \) resulting from the inner error flags \( \text{F1–F4} \):

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>iE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>8</td>
</tr>
</tbody>
</table>
The `nb` parameter defines the number of samples which are not included in the validity assessment after initialization of the block (restart). Recommended setting is $nb \geq 5$ to allow the SPIKE filter initial conditions to fade away.

### Inputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u$</td>
<td>Analog input of the block</td>
<td>double</td>
</tr>
<tr>
<td>$sv$</td>
<td>Substitute value to be used when the signal $u$ is marked as invalid</td>
<td>double</td>
</tr>
<tr>
<td>HWF</td>
<td>Hardware error indicator</td>
<td>bool</td>
</tr>
<tr>
<td>off</td>
<td>The input module of the signal works normally</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>Hardware error of the input module occurred</td>
<td></td>
</tr>
<tr>
<td>$R$</td>
<td>Reset inner error flags F1–F4</td>
<td>bool</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>Analog output of the block</td>
<td>double</td>
</tr>
<tr>
<td>$yf$</td>
<td>Filtered analog output signal $y$, output of the SPIKE filter</td>
<td>double</td>
</tr>
<tr>
<td>$E$</td>
<td>Output signal invalidity indicator</td>
<td>bool</td>
</tr>
<tr>
<td>off</td>
<td>Signal is valid $sv$</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>Signal is invalid, $y = yf =$</td>
<td></td>
</tr>
<tr>
<td>$iE$</td>
<td>Reason of invalidity</td>
<td>long</td>
</tr>
<tr>
<td>0</td>
<td>Signal valid</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Signal out of range</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Signal varies too little</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Signal varies too little and signal out of range</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Signal varies too much</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Signal varies too much and signal out of range</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Signal varies too much and too little</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Signal varies too much and too little and signal out of range</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Hardware error</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$nb$</td>
<td>Number of samples which are not included in the validity assessment of the signal $u$ after initialization of the block</td>
<td>long</td>
</tr>
<tr>
<td>$nc$</td>
<td>Number of samples for invariability testing (the F2 condition)</td>
<td>long</td>
</tr>
<tr>
<td>$nbits$</td>
<td>Number of A/D converter bits</td>
<td>long</td>
</tr>
<tr>
<td>$nr$</td>
<td>Number of samples for variability testing (the F3 condition)</td>
<td>long</td>
</tr>
<tr>
<td>$prate$</td>
<td>Maximum allowed percentage change of the input $u$ within the last $nr$ samples (with respect to the overall range of the input signal $v_{max} - v_{min}$)</td>
<td>double</td>
</tr>
<tr>
<td>$nv$</td>
<td>Number of samples for out-of-range testing (the F4 condition)</td>
<td>long</td>
</tr>
<tr>
<td>$v_{min}$</td>
<td>Lower limit for the input signal $u$</td>
<td>double</td>
</tr>
<tr>
<td>$v_{max}$</td>
<td>Upper limit for the input signal $u$</td>
<td>double</td>
</tr>
</tbody>
</table>
SEL – Analog signal selector

Function Description

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

The SEL block selects one of the four input signals $u_1$, $u_2$, $u_3$ and $u_4$ and copies it to the output signal $y$. The selection is based on the $iSW$ input or the binary inputs $SW_1$ and $SW_2$. These two modes are distinguished by the BINF binary flag. The signal is selected according to the following table:

<table>
<thead>
<tr>
<th>$iSW$</th>
<th>$SW_1$</th>
<th>$SW_2$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>off</td>
<td>off</td>
<td>$u_1$</td>
</tr>
<tr>
<td>1</td>
<td>off</td>
<td>on</td>
<td>$u_2$</td>
</tr>
<tr>
<td>2</td>
<td>on</td>
<td>off</td>
<td>$u_3$</td>
</tr>
<tr>
<td>3</td>
<td>on</td>
<td>on</td>
<td>$u_4$</td>
</tr>
</tbody>
</table>

Inputs

- $u_1$ First analog input of the block, double
- $u_2$ Second analog input of the block, double
- $u_3$ Third analog input of the block, double
- $u_4$ Fourth analog input of the block, double
- $iSW$ Active signal selector, active when $BINF = \text{off}$, long
- $SW_1$ Binary signal selector, active when $BINF = \text{on}$, bool
- $SW_2$ Binary signal selector, active when $BINF = \text{on}$, bool

Output

- $y$ The selected signal, double
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINF</td>
<td>Enable the binary selectors</td>
<td>bool</td>
</tr>
<tr>
<td>off</td>
<td>Disabled (analog selector)</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>Enabled (binary selectors)</td>
<td></td>
</tr>
</tbody>
</table>
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 \( u_0 \ldots u_{15} \) is based on the \( iSW \) input or the binary inputs \( SW_0 \ldots SW_3 \). These two modes are distinguished by the BINF binary flag. The signal is selected according to the following table:

<table>
<thead>
<tr>
<th>( iSW )</th>
<th>( SW_0 )</th>
<th>( SW_1 )</th>
<th>( SW_2 )</th>
<th>( SW_3 )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>( u_0 )</td>
</tr>
<tr>
<td>1</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>( u_1 )</td>
</tr>
<tr>
<td>2</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>( u_2 )</td>
</tr>
<tr>
<td>3</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>( u_3 )</td>
</tr>
<tr>
<td>4</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>( u_4 )</td>
</tr>
<tr>
<td>5</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>( u_5 )</td>
</tr>
<tr>
<td>6</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>( u_6 )</td>
</tr>
<tr>
<td>7</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>( u_7 )</td>
</tr>
<tr>
<td>8</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>( u_8 )</td>
</tr>
<tr>
<td>9</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>( u_9 )</td>
</tr>
<tr>
<td>10</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>( u_{10} )</td>
</tr>
<tr>
<td>11</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>( u_{11} )</td>
</tr>
<tr>
<td>12</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>( u_{12} )</td>
</tr>
<tr>
<td>13</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>( u_{13} )</td>
</tr>
<tr>
<td>14</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>( u_{14} )</td>
</tr>
<tr>
<td>15</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>( u_{15} )</td>
</tr>
</tbody>
</table>
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**

```
   u   y0
   |   y1
   v y2
   | y3
   v y4
   | y5
   v y6
   | y7
   v y8
   v SHIFTOCT
```

**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:

\[ y_i = y_{i-1}, \ i = 1..7 \]

\[ y_0 = u \]

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**

<table>
<thead>
<tr>
<th>i</th>
<th>Data input of the register</th>
<th>unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td></td>
<td>unknown</td>
</tr>
<tr>
<td>RUN</td>
<td>Enables outputs shift</td>
<td>bool</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>i</th>
<th>First output of the block</th>
<th>unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>y0</td>
<td></td>
<td>unknown</td>
</tr>
<tr>
<td>y1</td>
<td>Second output of the block</td>
<td>unknown</td>
</tr>
<tr>
<td>y2</td>
<td>Third output of the block</td>
<td>unknown</td>
</tr>
<tr>
<td>y3</td>
<td>Fourth output of the block</td>
<td>unknown</td>
</tr>
<tr>
<td>y4</td>
<td>Fifth output of the block</td>
<td>unknown</td>
</tr>
<tr>
<td>y5</td>
<td>Sixth output of the block</td>
<td>unknown</td>
</tr>
<tr>
<td>y6</td>
<td>Seventh output of the block</td>
<td>unknown</td>
</tr>
</tbody>
</table>
y7 Eighth output of the block

Parameters

<table>
<thead>
<tr>
<th>vtype</th>
<th>Output data type</th>
<th>08</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Short</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Word</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>DWord</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Float</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Double</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>Large</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SHLD – Sample and hold

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

- \( u \) Analog input of the block double
- SETH Trigger for the set and hold operation bool
- R1 Block reset, R1 = on \( \rightarrow y = y0 \) bool

Output

- \( y \) Analog output of the block double

Parameter

- \( y0 \) Initial output value double
- mode Sampling mode ⊗3 long
  1 .... Triggered sampling
  2 .... Hold last value
  3 .... Hold current value
**SINT – Simple integrator**

**Block Symbol**

<table>
<thead>
<tr>
<th>u</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>SINT</td>
<td></td>
</tr>
</tbody>
</table>

**Licence: STANDARD**

**Function Description**

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

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

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

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

**Input**

- \( u \) 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 \( (u, y) \rightarrow y \):

\[
\begin{align*}
\delta &:= y - u; \\
\text{if } \left| \delta \right| &< \text{gap} \\
\text{then} & \\
\text{begin} & \\
y &:= u; \\
gap &:= \text{gap}/q; \\
\text{ifgap} < \text{mingap} &\text{ then gap} := \text{mingap}; \\
\text{end} & \\
\text{else} & \\
\text{begin} & \\
\text{if } \delta < 0 &\text{ then } y := y + \text{gap} \\
\text{else } y &:= y - \text{gap}; \\
gap &:= \text{gap} \times q; \\
\text{end} & \\
\end{align*}
\]

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

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

Input
\( u \) Input signal to be filtered double
### Output

| $y$     | Filtered output signal | double |

### Parameters

<table>
<thead>
<tr>
<th>mingap</th>
<th>Minimum size of the tolerance window</th>
<th>$\geq 0.01$</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>Tolerance window adaptation speed</td>
<td>$\leq 1.0 \leq 2.0$</td>
<td>double</td>
</tr>
</tbody>
</table>
SSW – Simple switch

Block Symbol

```
<table>
<thead>
<tr>
<th>u1</th>
<th>u2</th>
<th>SW</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>SSW</td>
</tr>
</tbody>
</table>
```

licence: STANDARD

Function Description

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

Inputs

- $u_1$: First analog input of the block, double
- $u_2$: Second analog input of the block, double
- $SW$: Signal selector, bool
  - off ... The $u_1$ signal is selected, $y = u_1$
  - on .... The $u_2$ signal is selected, $y = u_2$

Output

- $y$: Analog output of the block, double
**SWR – Selector with ramp**

**Block Symbol**

![Block Symbol](image)

**Function Description**

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

**Inputs**

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

**Parameters**

- \( t_1 \) Rate limiter time constant for switching from \( u_2 \) to \( u_1 \) \( \oplus 1.0 \) \( \text{double} \)
- \( t_2 \) Rate limiter time constant for switching from \( u_1 \) to \( u_2 \) \( \oplus 1.0 \) \( \text{double} \)
- \( y_0 \) Initial output value to start the tracking from (before the first switching of signals occurs) \( \text{double} \)

**Output**

- \( y \) Analog output of the block \( \text{double} \)
**VDEL – Variable time delay**

**Block Symbol**

![block_symbol]

**Function Description**

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

**Inputs**

- $u$: Analog input of the block
- $d$: Time delay [s]

**Output**

- $y$: Delayed input signal

**Parameter**

- $y_0$: Initial/substitute output value
- $n_{\text{max}}$: Size (number of samples) of delay buffer (used for internal memory allocation)
**ZV4IS – Zero vibration input shaper**

**Block Symbol**

![Block Symbol](image)

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

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

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

The additional parameter $p_{a3}$ needs to be chosen for symmetrical filters ($p_{alpha} = 0$). 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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y )</td>
<td>Filtered output signal</td>
<td>double</td>
</tr>
<tr>
<td>( E )</td>
<td>Error flag</td>
<td>bool</td>
</tr>
</tbody>
</table>

- **off** ... No error
- **on** ... An error occurred

## Parameters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \omega )</td>
<td>Natural frequency</td>
<td>( \odot 1.0 ) double</td>
</tr>
<tr>
<td>( x_i )</td>
<td>Relative damping coefficient</td>
<td>double</td>
</tr>
<tr>
<td>( ipar )</td>
<td>Specification</td>
<td>( \odot 1 ) long</td>
</tr>
<tr>
<td>( istype )</td>
<td>Type</td>
<td>( \odot 2 ) long</td>
</tr>
<tr>
<td></td>
<td>1 ... Basic types of IS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ... Complete parametrization</td>
<td></td>
</tr>
<tr>
<td>( p_{\alpha} )</td>
<td>Shaper duration/assymetry parameter</td>
<td>( \odot 0.2 ) double</td>
</tr>
<tr>
<td>( p_{a2} )</td>
<td>Insensitivity parameter</td>
<td>( \odot 0.5 ) double</td>
</tr>
<tr>
<td>( p_{a3} )</td>
<td>Additional parameter (only for ( p_{\alpha} = 0 ))</td>
<td>( \odot 0.5 ) double</td>
</tr>
<tr>
<td>( n_{max} )</td>
<td>Size (number of samples) of delay buffer (used for internal memory allocation)</td>
<td>( \downarrow 1 \uparrow 10000000 \odot 10 ) long</td>
</tr>
</tbody>
</table>
Chapter 6

GEN – Signal generators

Contents

- ANLS – Controlled generator of piecewise linear function . . . . . . . . 150
- BINS – Controlled binary sequence generator . . . . . . . . . . . . . . 152
- BIS – Binary sequence generator . . . . . . . . . . . . . . . . . . . . . . 154
- MP – Manual pulse generator . . . . . . . . . . . . . . . . . . . . . . . . . 155
- PRBS – Pseudo-random binary sequence generator . . . . . . . . . . . . 156
- SG, SGI – Signal generators . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 158
CHAPTER 6. GEN – SIGNAL GENERATORS

ANLS – Controlled generator of piecewise linear function

Block Symbol  

<table>
<thead>
<tr>
<th>Licence: STANDARD</th>
</tr>
</thead>
</table>

Function Description

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

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

within the time intervals \( (t_i, t_{i+1}) \), \( i = 0, \ldots, 3, t_0 = 0 \).

To generate a step change in the output signal, it is necessary to define two nodes in the same time instant (i.e. \( t_i = t_{i+1} \)). The generation ends when time \( t_4 \) is reached or when time \( t_i \) is reached and the following node precedes the active one (i.e. \( t_{i+1} < t_i \)). The output holds its final value afterwards. But for the RPT parameter set to on, instead of holding the final value, the block returns to its initial state \( y_0 \), the internal block timer is set to 0 and the sequence is generated repeatedly. This can be used to generate square or sawtooth functions. The generation can also be prematurely terminated by the RUN input signal set to off. In that case the block returns to its initial state \( y_0 \), the internal block timer is set to 0 and \( i_0 = 0 \) becomes the active time interval.

Input

| RUN | Enable execution, run the analog sequence generation | bool |

Outputs

<table>
<thead>
<tr>
<th>y</th>
<th>Analog output of the block</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>is</td>
<td>Index of the active time interval</td>
<td>long</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>y0</th>
<th>Initial output value</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>Node 1 time</td>
<td>⬤1.0 double</td>
</tr>
<tr>
<td>y1</td>
<td>Node 1 value</td>
<td>double</td>
</tr>
<tr>
<td>t2</td>
<td>Node 2 time</td>
<td>⬤1.0 double</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------</td>
<td>--------</td>
</tr>
<tr>
<td>y2</td>
<td>Node 2 value</td>
<td>1.0</td>
</tr>
<tr>
<td>t3</td>
<td>Node 3 time</td>
<td>2.0</td>
</tr>
<tr>
<td>y3</td>
<td>Node 3 value</td>
<td>1.0</td>
</tr>
<tr>
<td>t4</td>
<td>Node 4 time</td>
<td>2.0</td>
</tr>
<tr>
<td>y4</td>
<td>Node 4 value</td>
<td></td>
</tr>
<tr>
<td>RPT</td>
<td>Repeating sequence</td>
<td></td>
</tr>
<tr>
<td></td>
<td>off</td>
<td>Disabled</td>
</tr>
<tr>
<td></td>
<td>on</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>double</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bool</td>
<td></td>
</tr>
</tbody>
</table>
**BINS – Controlled binary sequence generator**

Block Symbol

![Block Symbol](image)

**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 ti, where ti+1 < ti 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 (< Ts/2) or melting successive thin pulses into one thick pulse. Therefore it is strongly recommended to use integer multiples of the execution period as the switching times.

**Input**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>Starting signal (rising edge)</td>
<td>bool</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>Logical output of the block</td>
<td>bool</td>
</tr>
<tr>
<td>is</td>
<td>Index of the active time interval</td>
<td>long</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y0</td>
<td>Initial output value</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... Disabled/</td>
<td>off ...</td>
</tr>
<tr>
<td></td>
<td>on .... Enabled/true</td>
<td>on ....</td>
</tr>
<tr>
<td>t1</td>
<td>Switching time 1 [s]</td>
<td>1.0</td>
</tr>
<tr>
<td>t2</td>
<td>Switching time 2 [s]</td>
<td>2.0</td>
</tr>
<tr>
<td>t3</td>
<td>Switching time 3 [s]</td>
<td>3.0</td>
</tr>
<tr>
<td>t4</td>
<td>Switching time 4 [s]</td>
<td>0.4.0 double</td>
</tr>
<tr>
<td>----</td>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>t5</td>
<td>Switching time 5 [s]</td>
<td>0.5.0 double</td>
</tr>
<tr>
<td>t6</td>
<td>Switching time 6 [s]</td>
<td>0.6.0 double</td>
</tr>
<tr>
<td>t7</td>
<td>Switching time 7 [s]</td>
<td>0.7.0 double</td>
</tr>
<tr>
<td>t8</td>
<td>Switching time 8 [s]</td>
<td>0.8.0 double</td>
</tr>
<tr>
<td>RPT</td>
<td>Repeating sequence</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... Disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... Enabled</td>
<td></td>
</tr>
</tbody>
</table>
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 ti, where ti+1 < ti 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 (< TS/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

| Y | Logical output of the block | bool |
| is | Index of the active time interval | long |

Parameters

| Y0 | Initial output value |
| off ... | False/0 | on ... | True/1 |
| t1 | Switching time 1 [s] | 1.0 | double |
| t2 | Switching time 2 [s] | 2.0 | double |
| t3 | Switching time 3 [s] | 3.0 | double |
| 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 |
| off ... | false/0 | on ... | True/1 |
**MP – Manual pulse generator**

**Block Symbol**

![Block Symbol](image)

**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 REX Control System executive. Use the BIS block for such a purpose.

**Output**

<table>
<thead>
<tr>
<th>Y</th>
<th>Logical output of the block</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td></td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pwidth</code></td>
<td>Pulse width [s]</td>
<td>$\geq 1.0$ double</td>
</tr>
<tr>
<td>BSTATE</td>
<td>Output pulse activation</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... No action</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... Generate output pulse</td>
<td></td>
</tr>
<tr>
<td>RPTF</td>
<td>Allow pulse extension</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... Disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... Enabled</td>
<td></td>
</tr>
</tbody>
</table>
PRBS – Pseudo-random binary sequence generator

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

<table>
<thead>
<tr>
<th>START</th>
<th>Starting signal (rising edge)</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRK</td>
<td>Termination signal</td>
<td>bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>y</th>
<th>Generated pseudo-random binary sequence</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSY</td>
<td>Busy flag</td>
<td>bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>val0</th>
<th>Initial and final value</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>valhi</td>
<td>Upper level of the y output</td>
<td>1.0</td>
</tr>
<tr>
<td>vallo</td>
<td>Lower level of the y output</td>
<td>-1.0</td>
</tr>
<tr>
<td>swper</td>
<td>Period of random output switching [s]</td>
<td>1.0</td>
</tr>
<tr>
<td>swprob</td>
<td>Probability of switching</td>
<td>0.2</td>
</tr>
<tr>
<td>seqt</td>
<td>Length of the sequence [s]</td>
<td>10.0</td>
</tr>
<tr>
<td>waitt</td>
<td>Settling period [s]</td>
<td>2.0</td>
</tr>
</tbody>
</table>
SG, SGI – Signal generators

Block Symbols

Function Description

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>Enable execution, run the binary sequence generation</td>
<td>bool</td>
</tr>
<tr>
<td>SYN</td>
<td>Synchronization signal</td>
<td>bool</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>Analog output of the block</td>
<td>double</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>isig</td>
<td>Generated signal type</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>1 .... Sine wave</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 .... Symmetrical rectangular signal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 .... Sawtooth signal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 .... White noise with uniform distribution</td>
<td></td>
</tr>
<tr>
<td>amp</td>
<td>Amplitude of the generated signal</td>
<td>double</td>
</tr>
<tr>
<td>freq</td>
<td>Frequency of the generated signal</td>
<td>double</td>
</tr>
<tr>
<td>phase</td>
<td>Phase shift of the generated signal</td>
<td>double</td>
</tr>
<tr>
<td>offset</td>
<td>Value added to the generated signal</td>
<td>double</td>
</tr>
<tr>
<td>ifrunit</td>
<td>Frequency units</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>1 .... Hz</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 .... rad/s</td>
<td></td>
</tr>
<tr>
<td>Phaseshift unit</td>
<td>1 long</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>degrees</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>radians</td>
<td></td>
</tr>
</tbody>
</table>
# Chapter 7

**REG – Function blocks for control**

## Contents

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARLY</td>
<td>Advance relay</td>
<td>163</td>
</tr>
<tr>
<td>FLCU</td>
<td>Fuzzy logic controller unit</td>
<td>164</td>
</tr>
<tr>
<td>FRID</td>
<td>Frequency response identification</td>
<td>167</td>
</tr>
<tr>
<td>I3PM</td>
<td>Identification of a three parameter model</td>
<td>169</td>
</tr>
<tr>
<td>LC</td>
<td>Lead compensator</td>
<td>171</td>
</tr>
<tr>
<td>LLC</td>
<td>Lead-lag compensator</td>
<td>172</td>
</tr>
<tr>
<td>MCU</td>
<td>Manual control unit</td>
<td>173</td>
</tr>
<tr>
<td>PIDAT</td>
<td>PID controller with relay autotuner</td>
<td>175</td>
</tr>
<tr>
<td>PIDE</td>
<td>PID controller with defined static error</td>
<td>178</td>
</tr>
<tr>
<td>PIDGS</td>
<td>PID controller with gain scheduling</td>
<td>180</td>
</tr>
<tr>
<td>PIDMA</td>
<td>PID controller with moment autotuner</td>
<td>182</td>
</tr>
<tr>
<td>PIDU</td>
<td>PID controller unit</td>
<td>188</td>
</tr>
<tr>
<td>PIDUI</td>
<td>PID controller unit with variable parameters</td>
<td>191</td>
</tr>
<tr>
<td>POUT</td>
<td>Pulse output</td>
<td>193</td>
</tr>
<tr>
<td>PRGM</td>
<td>Setpoint programmer</td>
<td>194</td>
</tr>
<tr>
<td>PSMPC</td>
<td>Pulse-step model predictive controller</td>
<td>196</td>
</tr>
<tr>
<td>PWM</td>
<td>Pulse width modulation</td>
<td>200</td>
</tr>
<tr>
<td>RLY</td>
<td>Relay with hysteresis</td>
<td>202</td>
</tr>
<tr>
<td>SAT</td>
<td>Saturation with variable limits</td>
<td>203</td>
</tr>
<tr>
<td>SC2FA</td>
<td>State controller for 2nd order system with frequency autotuner</td>
<td>205</td>
</tr>
<tr>
<td>SCU</td>
<td>Step controller with position feedback</td>
<td>211</td>
</tr>
<tr>
<td>SCUV</td>
<td>Step controller unit with velocity input</td>
<td>214</td>
</tr>
<tr>
<td>SELU</td>
<td>Controller selector unit</td>
<td>218</td>
</tr>
<tr>
<td>SMHCC</td>
<td>Sliding mode heating/cooling controller</td>
<td>220</td>
</tr>
<tr>
<td>SMHCCA</td>
<td>Sliding mode heating/cooling controller with autotuner</td>
<td>224</td>
</tr>
<tr>
<td>SWU</td>
<td>Switch unit</td>
<td>231</td>
</tr>
</tbody>
</table>
**ARLY – Advance relay**

**Block Symbol**

<table>
<thead>
<tr>
<th>u</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARLY</td>
<td></td>
</tr>
</tbody>
</table>

**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.

![Diagram of ARLY block](image)

**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 | \( \ominus1.0 \)  
| en   | Value for switching the output to the "Off" state | \( \odot1.0 \)  
| tol  | Tolerance limit for the superposed noise of the input signal \( u \) | \( \downarrow0.0 \odot0.5 \)  
| ap   | Value of the \( y \) output in the "On" state | \( \odot1.0 \)  
| an   | Value of the \( y \) output in the "Off" state | \( \ominus1.0 \)  
| y0   | Initial output value | double
FLCU – Fuzzy logic controller unit

Block Symbol

Function Description

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

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

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

where $U_i, i = 1, \ldots, nu$ are the linguistic terms of the u input; $V_j, j = 1, \ldots, nv$ are the linguistic terms of the v input and $Y_k, k = 1, \ldots, ny$ 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 $x_1, \ldots, x_4$ are mutually different in triangular functions. The matrices of membership functions of the u and v input are composed of rows $[x_1, x_2, x_3, x_4]$. The dimensions of matrices $mfu$ and $mfv$ 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 $Y_k, k = 1, \ldots, ny$ in the case of $a_k = b_k = 0$. If $a_k \neq 0$ and $b_k \neq 0$, then the term $Y_k$ is assigned the value of $y_k + a_k u + b_k v$.

The output membership function matrix $sty$ has a dimension of $(ny \times 3)$ and contains the rows $[y_k, a_k, b_k], k = 1, \ldots, ny$.

The set of inference rules is also a matrix whose rows are $[i_l, j_l, k_l, w_l], l = 1, \ldots, nr$, 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 $w_l \in \{0, 1, \ldots, 100\}$. It is possible to suppress or emphasize a particular inference rule if necessary.
CHAPTER 7. REG – FUNCTION BLOCKS FOR CONTROL

Inputs

- u: First analog input of the block, double
- v: Second analog input of the block, double

Outputs

- y: Analog output of the block, double
- ir: Dominant rule, long
- wr: Degree of truth of the dominant rule, double

Parameters

- umax: Upper limit of the u input, \(\ominus 1.0\) double
- umin: Lower limit of the u input, \(\ominus -1.0\) double
- nu: Number of membership functions of the input u, \(\downarrow 1 \uparrow 25 \ominus 3\) long
- vmax: Upper limit of the v input, \(\ominus 1.0\) double
- vmin: Lower limit of the v input, \(\ominus -1.0\) double
- nv: Number of membership functions of the input v, \(\downarrow 1 \uparrow 25 \ominus 3\) long
- ny: Number of membership functions of the output y, \(\downarrow 1 \uparrow 100 \ominus 3\) long
- nr: Number of inference rules, \(\downarrow 1 \uparrow 25 \ominus 3\) long
- mfu: Matrix of membership functions of the input u
  \(\ominus [-1 -1 -1 0; -1 0 0 1; 0 1 1 1]\) double
- mfv: Matrix of membership functions of the input v
  \(\ominus [-1 -1 -1 0; -1 0 0 1; 0 1 1 1]\) double
- sty: Matrix of membership functions of the output y
  \(\ominus [-1 0 0; 0 0 0; 1 0 0]\) double
- rls: Matrix of inference rules
  \(\ominus [1 2 3 100; 1 1 1 100; 1 0 3 100]\) byte
FRID – * Frequency response identification

Block Symbol

Function Description

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

Inputs

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

Parameters

- **ubias**: Static component of the exciting signal
- **uamp**: Amplitude of the exciting signal
- **wb**: Frequency interval lower limit [rad/s]
- **wf**: Frequency interval higher limit [rad/s]
- **isweep**: Frequency sweeping mode
  - 1 . . . . : Logarithmic
  - 2 . . . . : Linear
- **cp**: Sweeping Rate
- **iavg**: Number of values for averaging
### Chapter 7. REG – Function Blocks for Control

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>obw</td>
<td>Observer bandwidth</td>
<td>LOW, NORMAL, HIGH</td>
</tr>
<tr>
<td>stime</td>
<td>Settling period [s]</td>
<td>10.0</td>
</tr>
<tr>
<td>umax</td>
<td>Maximum generator amplitude</td>
<td>1.0</td>
</tr>
<tr>
<td>thdmin</td>
<td>Minimum demanded THD threshold</td>
<td>0.0</td>
</tr>
<tr>
<td>adapt_rc</td>
<td>Maximum rate of amplitude variation</td>
<td>0.00</td>
</tr>
<tr>
<td>pv_max</td>
<td>Maximum desired process value</td>
<td>1.0</td>
</tr>
<tr>
<td>pv_sat</td>
<td>Maximum allowed process value</td>
<td>2.0</td>
</tr>
<tr>
<td>ADAPT_EN</td>
<td>Enable automatic amplitude adaptation</td>
<td>yes</td>
</tr>
<tr>
<td>immode</td>
<td>Measurement mode</td>
<td>Manual, Linear, Logarithmic, Automatic</td>
</tr>
<tr>
<td>nwm</td>
<td>Number of frequency response points for automatic mode</td>
<td>2.0 4.0 6.0 8.0</td>
</tr>
</tbody>
</table>

#### Outputs

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

Function Description

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

Inputs

<table>
<thead>
<tr>
<th></th>
<th>Input of the identified system</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>Output of the identified system</td>
<td>double</td>
</tr>
<tr>
<td>y</td>
<td>Input steady state</td>
<td>double</td>
</tr>
<tr>
<td>u0</td>
<td>Output steady state</td>
<td>double</td>
</tr>
<tr>
<td>y0</td>
<td>Execute identification</td>
<td>bool</td>
</tr>
<tr>
<td>RUN</td>
<td>Block reset</td>
<td>bool</td>
</tr>
<tr>
<td>ips</td>
<td>Meaning of the output signals</td>
<td>long</td>
</tr>
</tbody>
</table>

0 ...... FOPDT model
   p1 ... gain
   p2 ... time delay
   p3 ... time constant

1 ...... moments of input and output
   p1 ... parameter $mu_0$
   p2 ... parameter $mu_1$
   p3 ... parameter $mu_2$
   p4 ... parameter $my_0$
   p5 ... parameter $my_1$
   p6 ... parameter $my_2$

2 ...... process moments
   p1 ... parameter $mp_0$
   p2 ... parameter $mp_1$
   p3 ... parameter $mp_2$
3. . . . characteristic numbers
   \( p_1 \) . . . parameter \( \kappa \)
   \( p_2 \) . . . parameter \( \mu \)
   \( p_3 \) . . . parameter \( \sigma^2 \)
   \( p_4 \) . . . parameter \( \sigma \)

### Outputs

<table>
<thead>
<tr>
<th>( p_i )</th>
<th>Identified parameters with respect to ( i_{ps}, i = 1, \ldots, 8 )</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSY</td>
<td>Busy flag</td>
<td>bool</td>
</tr>
<tr>
<td>RDY</td>
<td>Ready flag</td>
<td>bool</td>
</tr>
<tr>
<td>E</td>
<td>Error flag</td>
<td>bool</td>
</tr>
<tr>
<td>iE</td>
<td>Error code</td>
<td>long</td>
</tr>
</tbody>
</table>

1. . . . Premature termination (RUN = off)
2. . . . \( \mu_0 = 0 \)
3. . . . \( mp_0 = 0 \)
4. . . . \( \sigma^2 < 0 \)

### Parameters

<table>
<thead>
<tr>
<th>( t_{ident} )</th>
<th>Duration of identification [s]</th>
<th>( \odot 100.0 )</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>( irtype )</td>
<td>Controller type (control law)</td>
<td>( \odot 6 )</td>
<td>long</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>( ispeed )</th>
<th>Desired closed loop speed</th>
<th>( \odot 2 )</th>
<th>long</th>
</tr>
</thead>
</table>

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

Block Symbol

\[
\begin{array}{c}
u \\
\text{LC} \\
y
\end{array}
\]

License: STANDARD

Function Description

The LC block is a discrete simulator of derivative element

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

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

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

Input

\( u \)  
Analog input of the block  
\text{double}

Output

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

Parameters

\( td \)  
Derivative time constant  
\text{\( \circ \)1.0 double}

\( nd \)  
Derivative filtering parameter  
\text{\( \circ \)10.0 double}

ISSF  
Steady state at start-up

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

\( \text{on} \)  
Initial steady state

\text{bool}
LLC – Lead-lag compensator

Block Symbol

\[
\begin{array}{c}
u \\
\text{LLC} \\
y
\end{array}
\]

Licence: STANDARD

Function Description

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

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

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

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

\( u \) Analog input of the block double

Output

\( y \) Analog output of the block double

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau )</td>
<td>Time constant</td>
<td>1.0</td>
<td>double</td>
</tr>
<tr>
<td>( a )</td>
<td>Numerator time constant coefficient</td>
<td></td>
<td>double</td>
</tr>
<tr>
<td>( \text{ISSF} )</td>
<td>Steady state at start-up</td>
<td>off</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>Zero initial state</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>on Initial steady state</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MCU – Manual control unit

Block Symbol

Function Description
The MCU block is suitable for manual setting of the numerical output value \( y \), e.g. a setpoint. In the local mode (\( \text{LOC} = \text{on} \)) the value is set using the buttons \( \text{UP} \) and \( \text{DN} \). 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 (\( \text{SATF} = \text{on} \)) by saturation limits \( \text{lolim} \) and \( \text{hilim} \). If none of the buttons is pushed (\( \text{UP} = \text{off} \) and \( \text{DN} = \text{off} \)), the output \( y \) tracks the input value \( t_v \). The tracking speed is controlled by the integration time constant \( t_t \).

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

Inputs

- \( t_v \) Tracking variable, double
- \( \text{UP} \) The "up" signal, bool
- \( \text{DN} \) The "down" signal, bool
- \( r_v \) Remote output value in the mode \( \text{LOC} = \text{off} \), double
LOC  Local or remote mode  bool

Output
y  Analog output of the block  double

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>tt</td>
<td>Tracking time constant of the input tv</td>
<td>double</td>
</tr>
<tr>
<td>tm</td>
<td>Initial value of integration time constant</td>
<td>double</td>
</tr>
<tr>
<td>y0</td>
<td>Initial output value</td>
<td>double</td>
</tr>
<tr>
<td>q</td>
<td>Multiplication quotient</td>
<td>double</td>
</tr>
<tr>
<td>ta</td>
<td>Interval after which the rate is changed [s]</td>
<td>double</td>
</tr>
<tr>
<td>tf</td>
<td>Interval after which the rate changes no more [s]</td>
<td>double</td>
</tr>
<tr>
<td>SATF</td>
<td>Saturation flag</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... Signal not limited</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ..... Saturation limits active</td>
<td></td>
</tr>
<tr>
<td>hilim</td>
<td>Upper limit of the output signal</td>
<td>double</td>
</tr>
<tr>
<td>lolim</td>
<td>Lower limit of the output signal</td>
<td>double</td>
</tr>
</tbody>
</table>
PIDAT – PID controller with relay autotuner

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 a 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

<table>
<thead>
<tr>
<th>dv</th>
<th>Feedforward control variable</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp</td>
<td>Setpoint variable</td>
<td>double</td>
</tr>
<tr>
<td>pv</td>
<td>Process variable</td>
<td>double</td>
</tr>
<tr>
<td>tv</td>
<td>Tracking variable</td>
<td>double</td>
</tr>
<tr>
<td>hv</td>
<td>Manual value</td>
<td>double</td>
</tr>
</tbody>
</table>
CHAPTER 7. REG – FUNCTION BLOCKS FOR CONTROL

Manual or automatic mode

off ... Automatic mode
on .... Manual mode

Start the tuning experiment

Stop the tuning experiment

Manipulated variable (controller output)

Deviation error

Saturation flag

The controller implements a linear control law

The controller output is saturated

Tuner busy flag

Tuning error

Error code; expected time (in seconds) to finishing the tuning experiment while the tuning experiment is active

Signal/noise ratio too low

Hysteresis too high

Too tight termination rule

Phase out of interval

Proposed controller gain

Proposed integral time constant

Proposed derivative time constant

Proposed derivative component filtering

Proposed weighting factor – proportional component

Controller type (control law)

Reverse action flag

Higher \( mv \) → higher \( pv \)

Higher \( mv \) → lower \( pv \)

Controller gain \( K \)

Integral time constant \( T_i \)

Derivative time constant \( T_d \)

Derivative filtering parameter \( N \)

Setpoint weighting – proportional part

Setpoint weighting – derivative part

Tracking time constant. No meaning for controllers without integrator.
hilim  Upper limit of the controller output  0.1  double
lolim  Lower limit of the controller output  -1.0  double
iainf  Type of apriori information  0  long
  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
k0  Static gain of the process (must be provided in case of iainf = 3, 4, 5)  0.1  double
n1  Maximum number of half-periods for estimation of frequency response point  20  long
mm  Maximum number of half-periods for averaging  4  long
amp  Relay controller amplitude  0.1  double
uhys  Relay controller hysteresis  double
ntime  Length of noise amplitude estimation period at the beginning of the tuning experiment [s]  5.0  double
rerrap  Termination value of the oscillation amplitude relative error  0.1  double
aerrph  Termination value of the absolute error in oscillation phase  10.0  double
maxdev  Maximal admissible deviation error from the initial steady state  1.0  double

It is recommended not to change the parameters n1, mm, ntime, rerrap and aerrph.
CHAPTER 7. REG – FUNCTION BLOCKS FOR CONTROL

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 (\( \text{MAN} = \text{off} \)) or manual mode (\( \text{MAN} = \text{on} \)).

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

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

where

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

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

By connecting the output \( \text{mv} \) of the controller to the controller input \( \text{tv} \) and properly setting the tracking time constant \( \text{tt} \) 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 \( \text{mv} \) occurs (antiwindup).

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

<table>
<thead>
<tr>
<th>dv</th>
<th>Feedforward control variable</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp</td>
<td>Setpoint variable</td>
<td>double</td>
</tr>
<tr>
<td>pv</td>
<td>Process variable</td>
<td>double</td>
</tr>
<tr>
<td>tv</td>
<td>Tracking variable</td>
<td>double</td>
</tr>
<tr>
<td>hv</td>
<td>Manual value</td>
<td>double</td>
</tr>
<tr>
<td>MAN</td>
<td>Manual or automatic mode</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... Automatic mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... Manual mode</td>
<td></td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>mv</th>
<th>Manipulated variable (controller output)</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>de</td>
<td>Deviation error</td>
<td>double</td>
</tr>
<tr>
<td>SAT</td>
<td>Saturation flag</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... The controller implements a linear control law</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... The controller output is saturated</td>
<td></td>
</tr>
</tbody>
</table>

## Parameters

<table>
<thead>
<tr>
<th>irtype</th>
<th>Controller type (control law)</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ..... D</td>
<td>2 ..... I</td>
<td>3 ..... ID</td>
</tr>
<tr>
<td>RACT</td>
<td>Reverse action flag</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... Higher mv -&gt; higher pv</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... Higher mv -&gt; lower pv</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>Controller gain $K$</td>
<td>double</td>
</tr>
<tr>
<td>ti</td>
<td>Integral time constant $T_i$</td>
<td>double</td>
</tr>
<tr>
<td>td</td>
<td>Derivative time constant $T_d$</td>
<td>double</td>
</tr>
<tr>
<td>nd</td>
<td>Derivative filtering parameter $N$</td>
<td>double</td>
</tr>
<tr>
<td>b</td>
<td>Setpoint weighting – proportional part</td>
<td>double</td>
</tr>
<tr>
<td>c</td>
<td>Setpoint weighting – derivative part</td>
<td>double</td>
</tr>
<tr>
<td>tt</td>
<td>Tracking time constant. No meaning for controllers without integrator.</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>4.0</td>
</tr>
<tr>
<td>bp</td>
<td>Static error coefficient</td>
<td>double</td>
</tr>
<tr>
<td>hilim</td>
<td>Upper limit of the controller output</td>
<td>double</td>
</tr>
<tr>
<td>lolim</td>
<td>Lower limit of the controller output</td>
<td>double</td>
</tr>
</tbody>
</table>
PIDGS – PID controller with gain scheduling

Block Symbol

Function Description

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

Inputs

\begin{verbatim}
\textbf{dv} \hspace{0.5cm} \text{Feedforward control variable} \hspace{4cm} \text{double}
\textbf{sp} \hspace{0.5cm} \text{Setpoint variable} \hspace{4cm} \text{double}
\textbf{pv} \hspace{0.5cm} \text{Process variable} \hspace{4cm} \text{double}
\textbf{tv} \hspace{0.5cm} \text{Tracking variable} \hspace{4cm} \text{double}
\textbf{hv} \hspace{0.5cm} \text{Manual value} \hspace{4cm} \text{double}
\textbf{MAN} \hspace{0.5cm} \text{Manual or automatic mode} \hspace{4cm} \text{bool}
\textbf{IH} \hspace{0.5cm} \text{Integrator hold} \hspace{4cm} \text{bool}
\textbf{ip} \hspace{0.5cm} \text{Parameter set index} \hspace{4cm} \text{long}
\textbf{vp} \hspace{0.5cm} \text{Switching analog signal} \hspace{4cm} \text{double}
\end{verbatim}

Outputs

\begin{verbatim}
\textbf{mv} \hspace{0.5cm} \text{Manipulated variable (controller output)} \hspace{4cm} \text{double}
\textbf{dmv} \hspace{0.5cm} \text{Controller velocity output (difference)} \hspace{4cm} \text{double}
\textbf{de} \hspace{0.5cm} \text{Deviation error} \hspace{4cm} \text{double}
\end{verbatim}
SAT Saturation flag

- **off**: The controller implements a linear control law
- **on**: The controller output is saturated

kp Active parameter set index

### Parameters

- **hilim**: Upper limit of the controller output
  - Value: 1.0
  - Type: double

- **lolim**: Lower limit of the controller output
  - Value: -1.0
  - Type: double

- **dz**: Dead zone
  - Value: double

- **icotype**: Controller output type
  - Value: 1
  - Type: long

- **npars**: Number of controller parameter sets
  - Value: 6
  - Type: long

- **GSCF**: Switch parameters by analog signal vp
  - Value: bool

- **hys**: Hysteresis for controller parameters switching
  - Value: double

- **irtypea**: Vector of controller types (control laws)
  - Value: 6 6 6 6 6 6
  - Type: byte

- **RACTA**: Vector of reverse action flags
  - Value: 0 0 0 0 0 0
  - Type: bool

- **ka**: Vector of controller gains $K$
  - Value: 1.0 1.0 1.0 1.0 1.0
  - Type: double

- **tia**: Vector of integral time constants $T_i$
  - Value: 4.0 4.0 4.0 4.0 4.0
  - Type: double

- **tda**: Vector of derivative time constants $T_d$
  - Value: 1.0 1.0 1.0 1.0 1.0
  - Type: double

- **nda**: Vector of derivative filtering parameters $N$
  - Value: 10.0 10.0 10.0 10.0 10.0
  - Type: double

- **ba**: Setpoint weighting factors – proportional part
  - Value: 1.0 1.0 1.0 1.0 1.0
  - Type: double

- **ca**: Setpoint weighting factors – derivative part
  - Value: 0.0 0.0 0.0 0.0 0.0
  - Type: double

- **tta**: Vector of tracking time constants. No meaning for controllers without integrator.
  - Value: 1.0 1.0 1.0 1.0 1.0
  - Type: double

- **thrsha**: Vector of thresholds for switching the parameters
  - Value: 0.1 0.2 0.3 0.4 0.5
  - Type: double
PIDMA – PID controller with moment autotuner

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

\[ U(s) = \pm K \left\{ \frac{bW(s) - Y(s)}{T_i s} + \frac{1}{T_i s} [W(s) - Y(s)] + \frac{T_d s}{N s^2 + 1} [cW(s) - Y(s)] \right\} + Z(s) \]

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:

![Diagram of PIDMA block](image)

The block $PIDMA$ extends the control function of the standard PID controller by the built-in autotuning feature. Before start of the autotuner the operator have to reach the steady state of the process at a suitable working point (in manual or automatic mode) and specify the required type of the controller $ittype$ (PI or PID) and other tuning parameters ($iainf$, $DGC$, $tdg$, $tn$, $amp$, $dy$ and $ispeed$). The identification experiment is started by the 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**

- `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
- `IH` Integrator hold `bool`
  - `off` ... Integration enabled
  - `on` ..... Integration disabled
- `TUNE` Start the tuning experiment (`off→on`) or force transition to the next tuning phase (see the description of the `ite` output) `bool`
- `TBRK` Stop the tuning experiment `bool`
TAFF  Tuning affirmation; determines the way the computed parameters are handled

- off ... Parameters are only computed
- on ..... Parameters are set into the control law

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

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

>99 ... Reserved for diagnostic purposes

**Outputs**

- **mv**  Manipulated variable (controller output)  double
- **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
- **TBSY** Tuner busy flag  bool
- **TE**  Tuning error
  - off ... Autotuning successful
  - on ..... An error occurred during the experiment  bool
- **ite** Error code  long

**Tuning error codes (after the experiment):**

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

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

Remark about terminating the tuning phases

TUN... 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]
pk Proposed controller gain $K \ (\text{ips}=0)$
pti Proposed integral time constant $T_i \ (\text{ips}=0)$
ptd Proposed derivative time constant $T_d \ (\text{ips}=0)$
pnd Proposed derivative component filtering $N \ (\text{ips}=0)$
pb Proposed weighting factor – proportional component (\text{ips}=0) \ double
pc Proposed weighting factor – derivative component (\text{ips}=0) \ double

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>irtype</td>
<td>Controller type (control law)</td>
</tr>
<tr>
<td>k</td>
<td>Controller gain $K$</td>
</tr>
<tr>
<td>ti</td>
<td>Integral time constant $T_i$</td>
</tr>
<tr>
<td>td</td>
<td>Derivative time constant $T_d$</td>
</tr>
<tr>
<td>nd</td>
<td>Derivative filtering parameter $N$</td>
</tr>
<tr>
<td>b</td>
<td>Setpoint weighting – proportional part</td>
</tr>
<tr>
<td>c</td>
<td>Setpoint weighting – derivative part</td>
</tr>
<tr>
<td>tt</td>
<td>Tracking time constant. No meaning for controllers without integrator</td>
</tr>
<tr>
<td>hilim</td>
<td>Upper limit of the controller output</td>
</tr>
<tr>
<td>lolim</td>
<td>Lower limit of the controller output</td>
</tr>
<tr>
<td>dz</td>
<td>Dead zone</td>
</tr>
<tr>
<td>icotype</td>
<td>Controller output type</td>
</tr>
</tbody>
</table>

1 . . . Analog output
2 . . . Pulse width modulation (PWM)
3 . . . Step controller unit with position feedback (SCU)
4 . . . Step controller unit without position feedback (SCUV)
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ittype</td>
<td>Controller type to be designed</td>
<td>long</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>PI controller</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>PID controller</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iainf</td>
<td>Type of apriori information</td>
<td>long</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Static process</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Astatic process</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DGC</td>
<td>Drift gradient compensation</td>
<td>bool</td>
<td>on</td>
</tr>
<tr>
<td></td>
<td>off ... Disabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... Enabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tdg</td>
<td>Drift gradient estimation time [s]</td>
<td>double</td>
<td>60.0</td>
</tr>
<tr>
<td>tn</td>
<td>Length of noise estimation period [s]</td>
<td>double</td>
<td>5.0</td>
</tr>
<tr>
<td>amp</td>
<td>Tuning pulse amplitude</td>
<td>double</td>
<td>0.5</td>
</tr>
<tr>
<td>dy</td>
<td>Tuning pulse get down threshold (absolute difference from the steady pv value)</td>
<td>double</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>↓0.0 ▼0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ispeed</td>
<td>Desired closed loop speed</td>
<td>long</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>Slow closed loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Normal (middle fast) closed loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Fast closed loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ipid</td>
<td>PID controller form</td>
<td>long</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Parallel form</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Series form</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PIDU – PID controller unit

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

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

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 ($\text{MAN} = \text{on}$), 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
- $\text{MAN}$: Manual or automatic mode  
  bool
  - off ... Automatic mode
  - on .... Manual mode
- $\text{IH}$: Integrator hold  
  bool
  - off ... Integration enabled
  - on .... Integration disabled

**Outputs**
- $mv$: Manipulated variable (controller output)  
  double
- $dmv$: Controller velocity output (difference)  
  double
- $de$: Deviation error  
  double
- $\text{SAT}$: Saturation flag  
  bool
  - off ... The controller implements a linear control law
  - on .... The controller output is saturated
Parameters

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

Function Description

The functionality of the PIDUI block is completely equivalent to the PIDU block. The only difference is that the PID control algorithm parameters are defined by the input signals and therefore they can depend on the outputs of other blocks. This allows creation of special adaptive PID controllers.

Inputs

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
IH Integrator hold bool
  off ... Integration enabled
  on .... Integration disabled
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
CHAPTER 7. REG – FUNCTION BLOCKS FOR CONTROL

**Outputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>mv</td>
<td>Manipulated variable (controller output)</td>
<td>double</td>
</tr>
<tr>
<td>dmv</td>
<td>Controller velocity output (difference)</td>
<td>double</td>
</tr>
<tr>
<td>de</td>
<td>Deviation error</td>
<td>double</td>
</tr>
<tr>
<td>SAT</td>
<td>Saturation flag</td>
<td>bool</td>
</tr>
</tbody>
</table>

- **The controller implements a linear control law**
- **The controller output is saturated**

**Parameters**

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

- **Higher mv! higher pv**
- **Higher mv! lower pv**
POUT – Pulse output

Block Symbol

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 \( \text{dtime} \) seconds and the idle period between two successive output pulses is at least \( \text{btime} \) seconds. The input pulse occurring sooner than the period of \( \text{btime} \) seconds since the last falling edge of the output signal elapses has no effect on the output signal \( Y \).

Input
\( U \) Logical input of the block bool

Output
\( Y \) Logical output of the block bool

Parameters
\( \text{dtime} \) Minimum width of the output pulse [s] 1.0 double
\( \text{btime} \) Minimum delay between two successive output pulses [s] 1.0 double
CHAPTER 7. REG – FUNCTION BLOCKS FOR CONTROL

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 \((tm = [t_0, \ldots, t_n])\) and output values \((y = [y_0, \ldots, y_n])\). The generated time-course is continuous piecewise linear, see figure below. This block is most commonly used as a setpoint generator for a controller. The program generation starts when RUN = on. In the case of RUN = off the programmer is set back to the initial state. The input DEF = on sets the output sp to the value spv. It follows a ramp to the nearest future node of the time function when DEF = off. The internal time of the generator is not affected by this input. The input HLD = on freezes the output sp and the internal time, thus also the outputs tsc, tt and rt. The program follows from freezing point as planned when HLD = off unless the input CON = on at the moment when the signal HLD on→off. In that case the program follows a ramp to reach the node with index ind in time trt. The node index ind must be equal to or higher than the index of current sector isc (at the moment when HLD on→off). If RPT = on, the program is generated repeatedly.

![Diagram](image)

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>Enable execution</td>
<td>bool</td>
</tr>
<tr>
<td>DEF</td>
<td>Initialize sp to the value of spv</td>
<td>bool</td>
</tr>
</tbody>
</table>
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 ind double
RPT  Repetition flag bool

Outputs

sp  Setpoint variable (function value of the time function at given time) double
isc  Current function sector long
tsc  Time elapsed since the start of current sector double
tt  Time elapsed since the start of program generation double
rt  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

n  Number of sectors ↓1 ↑10000000 ⊙2 long
tmunits  Time units ⊙1 long
           1  . . .  seconds
           2  . . .  minutes
           3  . . .  hours
tm  (n + 1)-dimensional vector of ascending node times ⊙[0 1 2] double
y   (n + 1)-dimensional vector of node values (values of the time function) double
    ⊙[0 1 0]
PSMPC – Pulse-step model predictive controller

Block Symbol

Licence: ADVANCED

Function Description

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

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

The prediction is based on the discrete step response $g(j)$, $j = 1, \ldots, N$ is used. The figure above shows how to obtain the discrete step response $g(j)$, $j = 0, 1, \ldots, N$ and the discrete impulse response $h(j)$, $j = 0, 1, \ldots, N$ with sampling period $T_S$ from continuous step response. Note that $N$ must be chosen such that $N \cdot T_S > t_{95}$, where $t_{95}$ is the time to reach 95 % of the final steady state value.
For stable, linear and t-invariant systems with monotonous step response it is also possible to use the moment model set approach \[5\] and describe the system by only 3 characteristic numbers, 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

\[
F_{\text{FOPDT}}(s) = \frac{K}{\tau s + 1} e^{-D_s}, \quad \kappa = K, \quad \mu = \tau + D, \quad \sigma^2 = \tau^2
\] (7.1)

or second order plus dead-time system

\[
F_{\text{SOPDT}}(s) = \frac{K}{(\tau s + 1)^2} e^{-D_s}, \quad \kappa = K, \quad \mu = 2\tau + D, \quad \sigma^2 = 2\tau^2
\] (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:

![Pulse-Step Shape](attachment:image.png)

Note that each of these sequences is uniquely defined by only four numbers \(n_1, n_2 \), \(p_0\) and \(u^\infty\) in \(\langle u^-, u^+ \rangle\), where \(N_C \in \{0, 1, \ldots\}\) is the control horizon and \(u^-, u^+\) stand for the given lower and upper limit of the manipulated variable. The on-line optimization (with respect to \(p_0, n_1, n_2\) and \(u^\infty\)) minimizes the criterion

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

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 \(p_0, n_1, n_2\) 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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sp</code></td>
<td>Setpoint variable</td>
<td>double</td>
</tr>
<tr>
<td><code>pv</code></td>
<td>Process variable</td>
<td>double</td>
</tr>
<tr>
<td><code>tv</code></td>
<td>Tracking variable (applied control signal)</td>
<td>double</td>
</tr>
<tr>
<td><code>hv</code></td>
<td>Manual value</td>
<td>double</td>
</tr>
<tr>
<td><code>MAN</code></td>
<td>Manual or automatic mode</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ...  Automatic mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ....  Manual mode</td>
<td></td>
</tr>
</tbody>
</table>

### Outputs

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

### Parameters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nc</code></td>
<td>Control horizon length ($N_C$)</td>
<td>$\odot$5 long</td>
</tr>
<tr>
<td><code>np1</code></td>
<td>Start of coincidence interval ($N_1$)</td>
<td>$\odot$1 long</td>
</tr>
<tr>
<td><code>np2</code></td>
<td>End of coincidence interval ($N_2$)</td>
<td>$\odot$10 long</td>
</tr>
<tr>
<td><code>lambda</code></td>
<td>Control signal penalization coefficient ($\lambda$)</td>
<td>$\odot$0.05 double</td>
</tr>
<tr>
<td><code>umax</code></td>
<td>Upper limit of the controller output ($u^+$)</td>
<td>$\odot$1.0 double</td>
</tr>
<tr>
<td><code>umin</code></td>
<td>Lower limit of the controller output ($u^-$)</td>
<td>$\odot$-1.0 double</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>imtype</td>
<td>Controlled process model type</td>
<td>3 long</td>
</tr>
<tr>
<td></td>
<td>1 ....</td>
<td>FOPDT model (7.1)</td>
</tr>
<tr>
<td></td>
<td>2 ....</td>
<td>SOPDT model (7.2)</td>
</tr>
<tr>
<td></td>
<td>3 ....</td>
<td>Discrete step response</td>
</tr>
<tr>
<td>kappa</td>
<td>Static gain ((\kappa))</td>
<td>1.0 double</td>
</tr>
<tr>
<td>mu</td>
<td>Resident time constant ((\mu))</td>
<td>20.0 double</td>
</tr>
<tr>
<td>sigma</td>
<td>Measure of the system response length ((\sqrt{\sigma^2}))</td>
<td>10.0 double</td>
</tr>
<tr>
<td>nsr</td>
<td>Length of the discrete step response ((N)), see the warning above</td>
<td>101000000 11 long</td>
</tr>
<tr>
<td>sr</td>
<td>Discrete step response sequence ([(g(1), \ldots, g(N))])</td>
<td>0.2642 0.5940 0.8009 0.9084 0.9596 0.9826 0.9927 0.9970 0.9988 0.9995 double</td>
</tr>
</tbody>
</table>
PWM – Pulse width modulation

Block Symbol

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:

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

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

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

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

Input

$u$ Analog input of the block double

Outputs

$\text{UP}$ The "up" signal bool
$\text{DN}$ The "down" signal bool
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pertm</td>
<td>Modulation period length [s]</td>
<td>10.0</td>
<td>double</td>
</tr>
<tr>
<td>dtime</td>
<td>Minimum width of the output pulse [s]</td>
<td>0.1</td>
<td>double</td>
</tr>
<tr>
<td>btime</td>
<td>Minimum delay between output pulses [s]</td>
<td>0.1</td>
<td>double</td>
</tr>
<tr>
<td>offtime</td>
<td>Minimum delay when altering direction [s]</td>
<td>1.0</td>
<td>double</td>
</tr>
<tr>
<td>asyfac</td>
<td>Asymmetry factor</td>
<td>1.0</td>
<td>double</td>
</tr>
<tr>
<td>SYNCH</td>
<td>Synchronization flag of the period start</td>
<td></td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... Synchronization disabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>on ... Synchronization enabled</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**RLY – Relay with hysteresis**

**Block Symbol**

![Block Symbol](image)

**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")  
  - $1.0$  
  **double**

- **$en$**  
The value $u < en$ causes $y = an$ ("Off")  
  - $-1.0$  
  **double**

- **$ap$**  
Output value $y$ in the "On" state  
  - $1.0$  
  **double**

- **$an$**  
Output value $y$ in the "Off" state  
  - $-1.0$  
  **double**

- **$y0$**  
Initial output value at start-up  
  -  
  **double**
SAT – Saturation with variable limits

Block Symbol

Function Description

The SAT block copies the input \( u \) to the output \( y \) if the input signal satisfies \( \text{lolim} \leq u \) 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 (HLD = on) defined by parameters \( \text{hilim0} \) and \( \text{lolim0} \) respectively or input-driven variables (HLD = off, \( \text{hi} \) and \( \text{lo} \) inputs). The maximal rate at which the active limits may vary is given by time constants \( tp \) (positive slope) and \( tn \) (negative slope). These rates are active even if the saturation limits are changed manually (HLD = on) using the \( \text{hilim0} \) and \( \text{lolim0} \) parameters. To allow immediate changes of the saturation limits, set \( tp = 0 \) and \( tn = 0 \). 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
- \( \text{hi} \) Upper limit of the output signal (for the case HLD = off) double
- \( \text{lo} \) Lower limit of the output signal (for the case HLD = 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

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

Fixed saturation limits

on bool

off . . . Variable limits on . . . Fixed limits
SC2FA – State controller for 2nd order system with frequency autotuner

Block Symbol

License: 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 $ID = \text{on}$. Two points of frequency response with given phase delay are measured during the identification experiment. Based on these two points a model of the controlled system is built. The experiment itself is initiated by the rising edge of the $RUN$ input. A harmonic signal with amplitude $u_{\text{amp}}$, frequency $\omega$ and bias $u_{\text{bias}}$ then appears at the output $mv$. The frequency runs through the interval $[\omega_b, \omega_f]$, 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

$$cp = \frac{\omega_b}{\omega(T_b)} = \frac{\omega_b e^{-\gamma T_b}}{\omega_b e^{-\gamma T_b}} = e^{-\gamma T_b}.$$
The $cp$ parameter usually lies within the interval $cp \in (0.95; 1)$. 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 $\omega = wb$. 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 $iDE$ displays the individual phases of the identification: $iDE = -1$ means approaching the first point, $iDE = 1$ means the break at the first point, $iDE = -2$ means approaching the second point, $iDE = 2$ means the break at the second point and $iDE = -3$ 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 $ID = off$) has manual ($MAN = on$) and automatic ($MAN = off$) submodes. After a cold start of the block with the input $ID = off$ it is assumed that the block parameters $mb0$, $mb1$, $ma0$ and $ma1$ reflect formerly identified coefficients $b0$, $b1$, $a0$ and $a1$ of the controlled system transfer function and the state controller design is performed automatically. Moreover if the controller is in the automatic
mode and SETC = on, then the control law uses the parameters from the very beginning. In this way the identification phase can be skipped when starting the block repeatedly.

The diagram above is a simplified inner structure of the frequency autotuning part of the controller. The diagram below shows the state feedback, observer and integrator anti-wind-up. The diagram does not show the fact, that the controller design block automatically adjusts the observer and state feedback parameters \( f_1, \ldots, f_5 \) after identification experiment (and SETC = on).

The controlled system is assumed in the form of (7.4). Another forms of this transfer function are

\[
F(s) = \frac{(b_1 s + b_0)}{s^2 + a_1 s + a_0} \tag{7.5}
\]

and

\[
F(s) = \frac{K_0 \Omega^2 (\tau s + 1)}{s^2 + 2 \xi \Omega s + \Omega^2} \tag{7.6}
\]

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.
CHAPTER 7. REG – FUNCTION BLOCKS FOR CONTROL

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
ID  Identification or controller operating mode  bool
        off ... Controller mode  on ... Identification and design
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  bool
        off ... Parameters are only computed
        on ... Parameters are accepted as soon as computed
        off→on  One-shot confirmation of the computed parameters
ips  Switch for changing the meaning of the output signals  long
        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
MFR: Generation of the parametric model frequency response at the \( w, x_{re} \) and \( x_{im} \) outputs (\texttt{off}→\texttt{on} triggers the generator)

### Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( mv )</td>
<td>Manipulated variable (controller output)</td>
<td>double</td>
</tr>
<tr>
<td>( de )</td>
<td>Deviation error</td>
<td>double</td>
</tr>
<tr>
<td>( SAT )</td>
<td>Saturation flag</td>
<td>bool</td>
</tr>
<tr>
<td>( w )</td>
<td>Frequency response point estimate - frequency in rad/s</td>
<td>double</td>
</tr>
<tr>
<td>( x_{re} )</td>
<td>Frequency response point estimate - real part</td>
<td>double</td>
</tr>
<tr>
<td>( x_{im} )</td>
<td>Frequency response point estimate - imaginary part</td>
<td>double</td>
</tr>
<tr>
<td>( epv )</td>
<td>Reconstructed ( pv ) signal</td>
<td>double</td>
</tr>
<tr>
<td>( IDE )</td>
<td>Identification error indicator</td>
<td>bool</td>
</tr>
<tr>
<td>( iIDE )</td>
<td>Error code</td>
<td>long</td>
</tr>
<tr>
<td>( p1...p6 )</td>
<td>Results of identification and design phase</td>
<td>double</td>
</tr>
</tbody>
</table>

### Parameters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( ubias )</td>
<td>Static component of the exciting harmonic signal</td>
<td>double</td>
</tr>
<tr>
<td>( uamp )</td>
<td>Amplitude of the exciting harmonic signal</td>
<td>double</td>
</tr>
<tr>
<td>( wb )</td>
<td>Frequency interval lower limit [rad/s]</td>
<td>double</td>
</tr>
<tr>
<td>( wf )</td>
<td>Frequency interval upper limit [rad/s]</td>
<td>double</td>
</tr>
<tr>
<td>( isweep )</td>
<td>Frequency sweeping mode</td>
<td>long</td>
</tr>
<tr>
<td>( cp )</td>
<td>Sweeping rate</td>
<td>double</td>
</tr>
<tr>
<td>( iavg )</td>
<td>Number of values for averaging</td>
<td>long</td>
</tr>
<tr>
<td>( alpha )</td>
<td>Relative positioning of the observer poles (in identification phase)</td>
<td>double</td>
</tr>
<tr>
<td>( xi )</td>
<td>Observer damping coefficient (in identification phase)</td>
<td>double</td>
</tr>
<tr>
<td>( MANF )</td>
<td>Manual frequency response points selection</td>
<td>bool</td>
</tr>
<tr>
<td>( ph1 )</td>
<td>Phase delay of the 1st point in degrees</td>
<td>double</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>ph2</td>
<td>Phase delay of the 2nd point in degrees</td>
<td>-120°</td>
</tr>
<tr>
<td>stime</td>
<td>Settling period [s]</td>
<td>10.0</td>
</tr>
<tr>
<td>ralpha</td>
<td>Relative positioning of the observer poles</td>
<td>4.0</td>
</tr>
<tr>
<td>rxi</td>
<td>Observer damping coefficient</td>
<td>0.707</td>
</tr>
<tr>
<td>acl1</td>
<td>Relative positioning of the 1st closed-loop poles couple</td>
<td>1.0</td>
</tr>
<tr>
<td>xicl1</td>
<td>Damping of the 1st closed-loop poles couple</td>
<td>0.707</td>
</tr>
<tr>
<td>INTGF</td>
<td>Integrator flag</td>
<td>on</td>
</tr>
<tr>
<td>apcl</td>
<td>Relative position of the real pole</td>
<td>1.0</td>
</tr>
<tr>
<td>DISF</td>
<td>Disturbance flag</td>
<td>off</td>
</tr>
<tr>
<td>dom</td>
<td>Disturbance model natural frequency</td>
<td>1.0</td>
</tr>
<tr>
<td>dxi</td>
<td>Disturbance model damping coefficient</td>
<td></td>
</tr>
<tr>
<td>acl2</td>
<td>Relative positioning of the 2nd closed-loop poles couple</td>
<td>2.0</td>
</tr>
<tr>
<td>xicl2</td>
<td>Damping of the 2nd closed-loop poles couple</td>
<td>0.707</td>
</tr>
<tr>
<td>tt</td>
<td>Tracking time constant</td>
<td>1.0</td>
</tr>
<tr>
<td>hlim</td>
<td>Upper limit of the controller output</td>
<td>1.0</td>
</tr>
<tr>
<td>lolim</td>
<td>Lower limit of the controller output</td>
<td>-1.0</td>
</tr>
<tr>
<td>mb1p</td>
<td>Controlled system transfer function coefficient $b_1$</td>
<td></td>
</tr>
<tr>
<td>mb0p</td>
<td>Controlled system transfer function coefficient $b_0$</td>
<td>1.0</td>
</tr>
<tr>
<td>ma1p</td>
<td>Controlled system transfer function coefficient $a_1$</td>
<td>0.2</td>
</tr>
<tr>
<td>ma0p</td>
<td>Controlled system transfer function coefficient $a_0$</td>
<td>1.0</td>
</tr>
</tbody>
</table>
**SCU — Step controller with position feedback**

**Block Symbol**

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 $thoff$ (see the TSE block, use parameters $ep = thron$, $epoff = thoff$, $en = -thron$ and $enoff = -thoff$). 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)\(\text{double}\)
- **pv** Controlled variable (position of the motorized valve drive)\(\text{double}\)
- **HS** Upper end switch (detects the upper limit position of the valve)\(\text{bool}\)
- **LS** Lower end switch (detects the lower limit position of the valve)\(\text{bool}\)
- **MUP** Manual UP signal\(\text{bool}\)
- **MDN** Manual DN signal\(\text{bool}\)
- **mdv** Manual differential value (requested position increment/decrement with higher priority than direct signals MUP/MDN)\(\text{double}\)
- **DVC** Differential value change command (off→on)\(\text{bool}\)
- **MAN** Manual or automatic mode\(\text{bool}\)
  - **off** .. Automatic mode
  - **on** .. Manual mode

**Outputs**
- **UP** The "up" signal\(\text{bool}\)
- **DN** The "down" signal\(\text{bool}\)
- **de** Deviation error\(\text{double}\)

**Parameters**
- **thron** Switch-on value\(\downarrow0.0\circ0.02\)\(\text{double}\)
- **thoff** Switch-off value\(\downarrow0.0\circ0.01\)\(\text{double}\)
- **dtime** Minimum width of the output pulse [s]\(\downarrow0.0\circ0.1\)\(\text{double}\)
- **btime** Minimum delay between two subsequent output pulses [s] to do\(\downarrow0.0\circ0.1\)\(\text{double}\)
- **RACT** Reverse action flag\(\text{bool}\)
  - **off** .. Higher mv → higher pv
  - **on** .. Higher mv → lower pv
**trun**  
Motor time constant (determines the time during which the motor position changes by one unit)  
*double*  
\[0.0 \leq 10.0\]
SCUV – Step controller unit with velocity input

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 (CWOI = 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 \(ep = \text{thron}, \text{epoff} = \text{throff}, \text{en} = -\text{thron}\) and \(\text{enoff} = -\text{throff}\)). Pulse outputs of the three state element are further shaped in such a way that the minimum pulse duration time \(d\text{time}\) and minimum pulse break time \(b\text{time}\) 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 (CWOI = on), then the deviation error of the primary controller can be eliminated by the bias ub manually. In this case, the control algorithm is slightly modified, the position of the motor pos is used and the proper settings of thron, throff and the tracking time constant tt are necessary for the suppressing of up/down pulses in the steady state.

There is also a group of input signals for manual control available. The manual mode is activated by the MAN = on input signal. Then it is possible to move the motor back and forth by the MUP and MDN input signals. It is also possible to specify a position
increment/decrement request by the MDV input. In this case the request must be confirmed by a rising edge (off→on) in the DVC input signal.
CHAPTER 7. REG – FUNCTION BLOCKS FOR CONTROL

The overall control function of the SCUV block is obvious from the following diagram:

The complete structures of the three-state controllers are depicted in the following figures:

### Primary controller with integration: I, PI, PID

- **Setpoint**
- **Valve Drive** MAN/AUT
- **Optional Connections**
- **Primary controller with integration: I, PI, PID**
- **Process Value**
- **Motorized Valve Drive**
- **SCUV (CWOI=0)**

### Primary controller without integration: P, PD

- **Setpoint**
- **Manual Bias**
- **Valve Drive** MAN/AUT
- **Optional Connections**
- **Primary controller without integration: P, PD**
- **Process Value**
- **Motorized Valve Drive**
- **SCUV (CWOI=1)**

### Inputs

- **mv** Manually Variable (controller output) [double]
- **dmv** Controller velocity output (difference) [double]
- **ub** Bias (only for P or PD primary controller) [double]
SAT: Internal integrator reset (connected to the SAT output of the primary controller)

HS: Upper end switch (detects the upper limit position of the valve)

LS: Lower end switch (detects the lower limit position of the valve)

MUP: Manual UP signal

MDN: Manual DN signal

mdv: Manual differential value (requested position increment/decrement with higher priority than direct signals MUP/MDN)

DVC: Differential value change command (off→on)

MAN: Manual or automatic mode

MAN off ... Automatic mode

MAN on .... Manual mode

Outputs

UP: The "up" signal

DN: The "down" signal

pos: Position output of motor simulator

MR: Request to move the motor

MR off ... Motor idle (UP = off and DN = off)

MR on .... Request to move (UP = on or DN = on)

Parameters

thron: Switch-on value

thoff: Switch-off value

dtime: Minimum width of the output pulse [s]

btime: Minimum delay between two subsequent output pulses [s]

RACT: Reverse action flag

RACT off ... Higher mv → higher pv

RACT on .... Higher mv → lower pv

trun: Motor time constant (determines the time during which the motor position changes by one unit)

CWOI: Controller without integration flag

CWOI off ... The primary controller has an integrator (I, PI, PID)

CWOI on .... The primary controller does not have an integrator (P, PD)

tt: Tracking time constant
SELU – Controller selector unit

Function Description

The SELU block is tailored for selecting the active controller in selector control. It chooses one of the input signals \( u_1, u_2, u_3, u_4 \) and copies it to the output \( y \). For \( \text{BINF}=\text{off} \) the active signal is selected by the \( iSW \) input. In the case of \( \text{BINF}=\text{on} \) the selection is based on the binary inputs \( \text{SW1} \) and \( \text{SW2} \) according to the following table:

<table>
<thead>
<tr>
<th>( iSW )</th>
<th>( \text{SW1} )</th>
<th>( \text{SW2} )</th>
<th>( y )</th>
<th>( U_1 )</th>
<th>( U_2 )</th>
<th>( U_3 )</th>
<th>( U_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>off</td>
<td>off</td>
<td>( u_1 )</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>1</td>
<td>off</td>
<td>on</td>
<td>( u_2 )</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>2</td>
<td>on</td>
<td>off</td>
<td>( u_3 )</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>3</td>
<td>on</td>
<td>on</td>
<td>( u_4 )</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>off</td>
</tr>
</tbody>
</table>

This table also explains the meaning of the binary outputs \( U_1, U_2, U_3 \) and \( U_4 \), which are used by the inactive controllers in selector control for tracking purposes (via the \( \text{SWU} \) blocks).

Inputs

- \( u_1..u_4 \): Signals to be selected from
- \( iSW \): Active signal selector in case of \( \text{BINF}=\text{off} \)
- \( \text{SW1} \): Binary signal selector, used when \( \text{BINF}=\text{on} \)
- \( \text{SW2} \): Binary signal selector, used when \( \text{BINF}=\text{on} \)

Outputs

- \( y \): Analog output of the block
- \( U_1..U_4 \): Binary output signal for selector control
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINF</td>
<td>Enable the binary selectors</td>
<td>bool</td>
</tr>
<tr>
<td>off</td>
<td>Disabled (analog selector)</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>Enabled (binary selectors)</td>
<td></td>
</tr>
</tbody>
</table>
SMHCC – Sliding mode heating/cooling controller

Function Description

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

It is important to note that the block SMHCC works with two time periods. The first period $T_S$ is the sampling time of the process temperature, and this period is equal to the period with which the block SMHCC itself is executed. The second period $T_C = i_{pwmc}T_S$ is the control period with which the block SMHCC generates manipulated variable. This period $T_C$ is also equal to the cycle time of PWM block. At every instant when the manipulated variable $mv$ 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:

\[ T_S = 0.1, \quad i_{pwm_c} = 100, \quad T_C = 10s, \quad T_R = 0.01s. \]

The control law of the block SMHCC in automatic mode (MAN = off) is based on the discrete dynamic sliding mode control technique and special 3rd order filters for estimation of the first and second derivatives of the control error.

The first control stage, after a setpoint change or upset, is the reaching phase when the dynamic sliding variable

\[ s_k = \dot{e}_k + 2\xi \Omega \dot{e}_k + \Omega^2 e_k \]

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

\[ s = \dot{e} + 2\xi \Omega \dot{e} + \Omega^2 e = 0 \]

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

The manipulated variable \( mv \) 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 \( u_{0_p} \) (\( u_{0_m} \)) to the suitable value less than \( hilim_p \) (\( hilim_m \)). Otherwise set \( u_{0_p} = hilim_p \) and \( u_{0_m} = hilim_m \).

The current amplitudes of heating and cooling \( u_{k_p}, u_{k_m} \), 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sp$</td>
<td>setpoint variable</td>
</tr>
<tr>
<td>$pv$</td>
<td>process variable</td>
</tr>
<tr>
<td>$hv$</td>
<td>manual value</td>
</tr>
<tr>
<td>$MAN$</td>
<td>controller mode</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$mv$</td>
<td>manipulated variable (position controller output)</td>
</tr>
<tr>
<td>$mve$</td>
<td>equivalent manipulated variable</td>
</tr>
<tr>
<td>$de$</td>
<td>deviation error</td>
</tr>
<tr>
<td>$SAT$</td>
<td>saturation flag</td>
</tr>
<tr>
<td>$isv$</td>
<td>number of the positive (+) or negative (−) sliding variable steps</td>
</tr>
<tr>
<td>$t_{ukp}$</td>
<td>current amplitude of heating</td>
</tr>
<tr>
<td>$t_{ukm}$</td>
<td>current amplitude of cooling</td>
</tr>
<tr>
<td>$t_{sk}$</td>
<td>discrete dynamic sliding variable $s_k$</td>
</tr>
<tr>
<td>$t_{pv}$</td>
<td>filtered control error $-de$</td>
</tr>
<tr>
<td>$t_{dpv}$</td>
<td>filtered first derivative of the control error $t_{ek}$</td>
</tr>
<tr>
<td>$t_{d2pv}$</td>
<td>filtered second derivative of the control error $t_{ek}$</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ipwm_c$</td>
<td>PWM cycle in the sampling periods of SMHCC ($T_C/T_S$)</td>
</tr>
<tr>
<td>$xi$</td>
<td>relative damping $\xi$ of sliding zero dynamics $xi \geq 0$</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>om</td>
<td>natural frequency $\Omega$ of sliding zero dynamics</td>
</tr>
<tr>
<td>taup</td>
<td>time constant for adaptation of heating action amplitude in seconds</td>
</tr>
<tr>
<td>taum</td>
<td>time constant for adaptation of cooling action amplitude in seconds</td>
</tr>
<tr>
<td>beta</td>
<td>bandwidth parameter of the derivative filter</td>
</tr>
<tr>
<td>hilim_p</td>
<td>high limit of the heating action amplitude</td>
</tr>
<tr>
<td>hilim_m</td>
<td>high limit of the cooling action amplitude</td>
</tr>
<tr>
<td>u0_p</td>
<td>initial value of the heating action amplitude after setpoint change and</td>
</tr>
<tr>
<td></td>
<td>start of the block</td>
</tr>
<tr>
<td>u0_m</td>
<td>initial value of the cooling action amplitude after setpoint change and</td>
</tr>
<tr>
<td></td>
<td>start of the block</td>
</tr>
<tr>
<td>sp_dif</td>
<td>Setpoint difference threshold</td>
</tr>
<tr>
<td>tauf</td>
<td>Equivalent manipulated variable filter time constant</td>
</tr>
</tbody>
</table>
SMHCCA – Sliding mode heating/cooling controller with autotuner

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 \( T_C = i_{pwmc} T_S \) 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 (\( i_{pwmc} \) as maximal as possible), the ratio \( T_C/T_S \) as maximal as possible but \( T_C \) should be sufficiently small with respect to the process dynamics. An example of reasonable values for an extruder temperature control is as follows:

\[
T_S = 0.1, \quad i_{pwmc} = 50, \quad T_C = 5s, \quad T_R = 0.1s.
\]

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 \( \text{beta} \) is critical for correct function of the controller. Also, some other parameters with the clear meaning described below have to be chosen manually. All the remaining parameters (\( xi, om, taup, taum, tauf \)) 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 → 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 → 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 ($\text{MAN} = \text{off}$) 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

$$s_k \triangleq \ddot{e}_k + 2 \xi \dot{e}_k + \Omega^2 e_k$$

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.
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

$$s = \triangle \dot{e} + 2\xi \Omega \dot{e} + \Omega^2 e$$

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

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

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

The manipulated variable $mv$ 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 $u_{0_p}$ ($u_{0_m}$) to the suitable value less than $hilim_p$ ($hilim_m$). Otherwise set $u_{0_p} = hilim_p$ and $u_{0_m} = hilim_m$.

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 $s_k$ alternates 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 (\( MAN = \text{on} \)) 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 \( \tau_{af} \) applied to \( mv \).

**Inputs**

- **sp**: Setpoint variable, double
- **pv**: Process variable, double
- **hv**: Manual value, double
- **MAN**: Manual or automatic mode, bool
  
  0 — Automatic mode  
  1 — Manual mode
- **TMODE**: Tuning mode, bool
- **TUNE**: Start the tuning experiment: TUNE off→on, bool
- **TBRK**: Stop the tuning experiment: TBRK off→on, bool
- **TAFF**: Affirmation of the parameter set provided by the tuning procedure: TAEFF = on, bool
- **ips**: Meaning of the output signals \( p_1,\ldots,p_6 \), long
  
  0 — Controller parameters
  
  \( p_1 \ldots \) recommended control period \( T_C \)  
  \( p_2 \ldots \) xi  
  \( p_3 \ldots \) om  
  \( p_4 \ldots \) taup  
  \( p_5 \ldots \) taum  
  \( p_6 \ldots \) tauf  
  1 — Auxiliary parameters
  
  \( p_1 \ldots \) htp2 — time of the peak in the second derivative of \( pv \)  
  \( p_2 \ldots \) hpeak2 — peak value in the second derivative of \( pv \)  
  \( p_3 \ldots \) d2 — peak to peak amplitude of \( t_{d2pv} \)  
  \( p_4 \ldots \) tgain

**Outputs**

- **mv**: Manipulated variable (controller output), double
- **mve**: Equivalent manipulated variable, double
- **de**: Deviation error, double
- **SAT**: Saturation flag, bool
  
  0 — Signal not limited  
  1 — Saturation limits active, \( mv \geq hilim_p \) or \( mv \leq -hilim_m \)
- **isv**: Number of the positive (+) or negative (−) sliding variable steps, long
- **t_ukp**: Current amplitude of heating, double
- **t_ukm**: Current amplitude of cooling, double
- **t_sk**: Discrete dynamic sliding variable, double
- **t_pv**: Filtered process variable \( pv \) by 3rd order filter, double
CHAPTER 7. REG – FUNCTION BLOCKS FOR CONTROL

- $t_{dpv}$: Filtered first derivative of $pv$ by 3rd order filter, $\text{double}
- t_{d2pv}$: Filtered second derivative of $pv$ by 3rd order filter, $\text{double}
- \text{TBSY}$: Tuner busy flag ($\text{TBSY} = \text{on}$), $\text{bool}
- \text{TE}$: Tuning error, $\text{bool}
  - \text{off} ... Autotuning successful
  - \text{on} ... An error occurred during the experiment
- $\text{ite}$: Error code, $\text{long}
  - 0 ... No error
  - 1 ... Noise level in $pv$ too high, check the temperature input
  - 2 ... Incorrect parameter $ut_p$
  - 3 ... Setpoint $sp$ too low
  - 4 ... The two minimal process time constants are probably too small with respect to the sampling period $T_S$ OR too high level of noise in the second derivative of $pv$ (try to decrease the $\text{beta}$ parameter)
  - 5 ... Premature termination of the tuning procedure ($\text{TBRK}$)
- $\text{pi}$: Identified parameters with respect to $ips$, $i = 1, ..., 6$, $\text{double}$

Parameters

- $\text{ipwm}$: PWM cycle (in sampling periods of the block, $T_C/T_S$) $\circ 100$ $\text{long}$
- $\text{xi}$: Relative damping of sliding zero dynamics $\downarrow 0.5 \uparrow 8.0 \circ 1.0$ $\text{double}$
- $\text{om}$: Natural frequency $\omega$ of sliding zero dynamics $\downarrow 0.0 \circ 0.01$ $\text{double}$
- $\text{taup}$: Time constant for adaptation of heating action amplitude [s] $\circ 700.0$ $\text{double}$
- $\text{taum}$: Time constant for adaptation of cooling action amplitude [s] $\circ 400.0$ $\text{double}$
- $\text{beta}$: Bandwidth parameter of the derivative filter $\circ 0.01$ $\text{double}$
- $\text{hilim}_p$: Upper limit of the heating action amplitude $\downarrow 0.0 \uparrow 1.0 \circ 1.0$ $\text{double}$
- $\text{hilim}_m$: Upper limit of the cooling action amplitude $\downarrow 0.0 \uparrow 1.0 \circ 1.0$ $\text{double}$
- $\text{u0}_p$: Initial amplitude of the heating action $\circ 1.0$ $\text{double}$
- $\text{u0}_m$: Initial amplitude of the cooling action $\circ 1.0$ $\text{double}$
- $\text{sp_dif}$: Setpoint difference threshold for blocking of heating/cooling amplitudes reset $\circ 10.0$ $\text{double}$
- $\text{tauf}$: Time constant of the filter for obtaining the equivalent manipulated variable $\circ 400.0$ $\text{double}$
- $\text{itm}$: Tuning method $\circ 1$ $\text{long}$
  - 1 ... Restricted to symmetrical processes
  - 2 ... Asymmetrical processes (not implemented yet)
- $\text{ut}_p$: Amplitude of heating for tuning experiment $\downarrow 0.0 \uparrow 1.0 \circ 1.0$ $\text{double}$
- $\text{ut}_m$: Amplitude of cooling for tuning experiment $\downarrow 0.0 \uparrow 1.0 \circ 1.0$ $\text{double}$
**SWU – Switch unit**

**Block Symbol**

<table>
<thead>
<tr>
<th>uc</th>
<th>uo</th>
<th>OR1</th>
<th>OR2</th>
<th>OR3</th>
<th>OR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>uc</td>
<td>uo</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Function Description**

The **SWU** block is used to select the appropriate signal which should be tracked by the inactive **PIDU** and **MCU** units in complex control structures. The input signal **uc** is copied to the output **y** when all the binary inputs **OR1**, . . . , **OR4** are off, otherwise the output **y** takes over the **uo** input signal.

**Inputs**

- **uc**
  - This input is copied to output **y** when all the binary inputs **OR1**, **OR2**, **OR3** and **OR4** are off
  - double

- **uo**
  - This input is copied to output **y** when any of the binary inputs **OR1**, **OR2**, **OR3**, **OR4** is on
  - 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**

![Block Symbol](image)

**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  
\( \text{double} \)

**Outputs**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>UP</td>
<td>The &quot;up&quot; signal</td>
<td>( \text{bool} )</td>
</tr>
<tr>
<td>DN</td>
<td>The &quot;down&quot; signal</td>
<td>( \text{bool} )</td>
</tr>
</tbody>
</table>

**Parameters**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ep</td>
<td>The input value ( u &gt; \text{ep} ) results in ( \text{UP} = \text{on} ) and ( \text{DN} = \text{off} )</td>
<td>( \text{⊙}1.0 )</td>
<td>( \text{double} )</td>
<td></td>
</tr>
<tr>
<td>en</td>
<td>The input value ( u &lt; \text{en} ) results in ( \text{UP} = \text{off} ) and ( \text{DN} = \text{off} )</td>
<td>( \text{⊙}-1.0 )</td>
<td>( \text{double} )</td>
<td></td>
</tr>
<tr>
<td>eoff</td>
<td>UP switch off value; if ( \text{UP} = \text{on} ) and ( u &lt; \text{eoff} ) then ( \text{UP} = \text{off} )</td>
<td>( \text{⊙}0.5 )</td>
<td>( \text{double} )</td>
<td></td>
</tr>
<tr>
<td>enoff</td>
<td>DN switch off value; if ( \text{DN} = \text{on} ) and ( u &gt; \text{enoff} ) then ( \text{DN} = \text{off} )</td>
<td>( \text{⊙}-0.5 )</td>
<td>( \text{double} )</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 8

LOGIC – Logic control

Contents

AND - Logical product of two signals ............................................. 234
ANDOCT – Logical product of eight signals ................................. 235
ATMT – Finite-state automaton .................................................. 236
BDOCT, BDHEXD – Bitwise demultiplexers ............................... 239
BITOP – Bitwise operation ....................................................... 240
BMOCT, BMHEXD – Bitwise multiplexers .................................. 241
COUNT – Controlled counter ..................................................... 242
EATMT – Extended finite-state automaton .................................. 244
EDGE – Falling/rising edge detection in a binary signal ............... 247
INTSM – Integer number bit shift and mask ............................. 248
ISSW – Simple switch for integer signals ................................... 249
INTSM – Integer number bit shift and mask ............................. 250
ITOI – Transformation of integer and binary numbers ................. 251
NOT – Boolean complementation .............................................. 253
OR – Logical sum of two signals ............................................ 254
OROCT – Logical sum of eight signals .................................... 255
RS – Reset-set flip-flop circuit ................................................. 256
SR – Set-reset flip-flop circuit ................................................. 257
TIMER – Multipurpose timer .................................................... 258
**AND_ – Logical product of two signals**

**Block Symbol**

![Block Symbol](image)

**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  
  Type: bool
- **U2**: Second logical input of the block  
  Type: bool

**Outputs**

- **Y**: Output signal, logical product (U1 \( \land \) U2)  
  Type: bool
- **NY**: Boolean complementation of Y (NY = \( \neg \)Y)  
  Type: bool
ANDOCT – Logical product of eight signals

Block Symbol

Function Description

The ANDOCT block computes the logical product of eight input signals \( U_1, U_2, \ldots, U_8 \). The signals listed in the \( n_1 \) parameter are negated prior to computing the logical product.

For an empty \( n_1 \) parameter a simple logical product \( Y = U_1 \land U_2 \land U_3 \land U_4 \land U_5 \land U_6 \land U_7 \land U_8 \) is computed. For e.g. \( n_1=2, 6..8 \), the logical function is \( Y = U_1 \land \neg U_2 \land U_3 \land U_4 \land U_5 \land \neg U_6 \land \neg U_7 \land \neg U_8 \).

If you have less than 8 signals, use the \( n_1 \) parameter to handle the unconnected inputs. If you have only two input signals, consider using the AND block.

Inputs

<table>
<thead>
<tr>
<th>( U_1 )</th>
<th>First logical input of the block</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>( U_2 )</td>
<td>Second logical input of the block</td>
<td>bool</td>
</tr>
<tr>
<td>( U_3 )</td>
<td>Third logical input of the block</td>
<td>bool</td>
</tr>
<tr>
<td>( U_4 )</td>
<td>Fourth logical input of the block</td>
<td>bool</td>
</tr>
<tr>
<td>( U_5 )</td>
<td>Fifth logical input of the block</td>
<td>bool</td>
</tr>
<tr>
<td>( U_6 )</td>
<td>Sixth logical input of the block</td>
<td>bool</td>
</tr>
<tr>
<td>( U_7 )</td>
<td>Seventh logical input of the block</td>
<td>bool</td>
</tr>
<tr>
<td>( U_8 )</td>
<td>Eighth logical input of the block</td>
<td>bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>( Y )</th>
<th>Output signal, logical product</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>( NY )</td>
<td>Boolean complementation of ( Y )</td>
<td>bool</td>
</tr>
</tbody>
</table>

Parameter

<table>
<thead>
<tr>
<th>( n_1 )</th>
<th>List of signals to negate. The format of the list is e.g. 1,3..5,8.</th>
<th>long</th>
</tr>
</thead>
</table>

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**

```
R1
ns0
SET
HLD
C0
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12
C13
C14
C15
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
Q9
Q10
Q11
Q12
Q13
Q14
Q15
ksa
tstep
TOUT
```

**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 \( i, i = 0, 1, \ldots, 15 \) is indicated by the binary outputs \( Q_0, Q_1, \ldots, Q_{15} \). If the state \( i \) is active, the corresponding output is set to \( Q_i = \text{on} \). The current state is also indicated by the \( \text{ksa} \) output (\( \text{ksa} \in \{0, 1, \ldots, 15\} \)).

The transition conditions \( C_k, k = 0, 1, \ldots, 15 \) are activated by the binary inputs \( C_0, C_1, \ldots, C_{15} \). If \( C_k = \text{on} \) the \( k \)-th transition condition is fulfilled. The transition cannot happen when \( C_k = \text{off} \).

The automat function is defined by the following table of transitions:

\[
\begin{array}{ccc}
S_1 & C_1 & FS_1 \\
S_2 & C_2 & FS_2 \\
& & \\
Sn & C_n & FS_n \\
\end{array}
\]

Each row of this table represents one transition rule. For example the first row

\[
S_1 \quad C_1 \quad FS_1
\]

has the meaning

If (\( S_1 \) is the current state AND transition condition \( C_1 \) is fulfilled)

then proceed to the following state \( FS_1 \).
The above mentioned table can be easily constructed from the automat state diagram or SFC description (Sequential Function Charts, formerly Grafcet).

The \( R1 = \text{on} \) input resets the automat to the initial state \( S0 \). The \( \text{SET} \) input allows manual transition from the current state to the \( \text{ns}0 \) state when rising edge occurs. The \( R1 \) input overpowers the \( \text{SET} \) input. The \( \text{HLD} = \text{on} \) input freezes the automat activity, the automat stays in the current state regardless of the \( C_i \) input signals and the \( \text{tstep} \) timer is not incremented. The \( \text{TOUT} \) output indicates that the machine remains in the given state longer than expected. The time limits \( TO_i \) for individual states are defined by the \( \text{touts} \) array. There is no time limit for the given state when \( TO_i \) is set to zero. The \( \text{TOUT} \) output is set to \( \text{off} \) whenever the automat changes its state.

It is possible to allow more state transitions in one cycle by the \( \text{morestps} \) parameter. However, this option must be thoroughly considered and tested, namely when the \( \text{TOUT} \) output is used in transition conditions. In such a case it is strongly recommended to incorporate the \( \text{k} \) output in the transition conditions as well.

The development tools of the REX Control System include also the SFCEditor program. You can create SFC schemes graphically using this tool. Run this editor from RexDraw by clicking the \textit{Configure} button in the parameter dialog of the \textit{ATMT} block.

### Inputs

- **R1**
  - Reset signal, \( R1 = \text{on} \) brings the automat to the initial state \( S0 \); the \( R1 \) input overpowers the \( \text{SET} \) input
  - \text{bool}

- **ns0**
  - This state is reached when rising edge occurs at the \( \text{SET} \) input
  - \text{long}

- **SET**
  - The rising edge of this signal forces the transition to the \( \text{ns}0 \) state
  - \text{bool}

- **HLD**
  - The \( \text{HLD} = \text{on} \) freezes the automat, no transitions occur regardless of the input signals, \( \text{tstep} \) is not increasing
  - \text{bool}

- **C0...C15**
  - The transition conditions; \( C_i = \text{on} \) means that the \( i \)-th condition was fulfilled and the corresponding transition rule can be executed
  - \text{bool}

### Outputs

- **Q0...Q15**
  - Output signals indicating the current state of the automat; the current state \( i \) is indicated by \( Q_i = \text{on} \)
  - \text{bool}

- **ksa**
  - Integer code of the active state
  - \text{long}

- **tstep**
  - Time elapsed since the current state was reached; the timer is set to \( 0 \) whenever a state transition occurs
  - \text{double}

- **TOUT**
  - Flag indicating that the time limit for the current state was exceeded
  - \text{bool}

### Parameters

- **morestps**
  - Allow multiple transitions in one cycle of the automat
  - \text{bool}
    - \text{off} ... Disabled
    - \text{on} ... Enabled

- **ntr**
  - Number of state transition table rows
  - \text{long}
    - \( \downarrow 0 \uparrow 64 \circ 4 \)
<table>
<thead>
<tr>
<th>sfcname</th>
<th>Filename of block configurator data file (filename is generated by system if parameter is empty)</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>STT</td>
<td>State transition table (matrix) ⊙[0 0 1; 1 1 2; 2 2 3; 3 3 0]</td>
<td>byte</td>
</tr>
<tr>
<td>touts</td>
<td>Vector of timeouts $TO_0, TO_1, \ldots, TO_{15}$ for the states $S_0, S_1, \ldots, S_{15}$</td>
<td>double</td>
</tr>
</tbody>
</table>

⊙\[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16\]
BDOCT, BDHEXD – Bitwise demultiplexers

Block Symbols

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 $Y_0$ output is the least significant bit.

Input

$iu$ Input signal to be decomposed $\text{long}$

Outputs

$Y_0 \ldots Y_{15}$ Individual bits of the input signal $\text{bool}$

Parameter

$shift$ Bit shift of the input signal $0 \leftrightarrow 31 \text{ long}$
BITOP – Bitwise operation

Block Symbol

\[ \begin{array}{ccc}
i_1 & \circ & i_2 \\
\hline
n \\
\end{array} \]

Licence: STANDARD

Function Description

The BITOP block performs bitwise operation \( i_1 \circ i_2 \) on the signals \( i_1 \) and \( i_2 \), resulting in an integer output \( n \). The type of operation is selected by the \( iop \) parameter described below. In case of logical negation or 2’s complements the input \( i_2 \) is ignored (i.e. the operation is unary).

Inputs

\[
\begin{array}{ll}
i_1 & \text{First integer input of the block} \\
i_2 & \text{Second integer input of the block} \\
\end{array}
\]

Output

\[
\begin{array}{ll}
n & \text{Result of the bitwise operation } iop \\
\end{array}
\]

Parameter

\[
\begin{array}{ll}
iop & \text{Bitwise operation} \\
1 & \text{Bitwise negation (Bit NOT)} \\
2 & \text{Bitwise logical sum (Bit OR)} \\
3 & \text{Bitwise logical product (Bit AND)} \\
4 & \text{Bitwise logical exclusive sum (Bit XOR)} \\
5 & \text{Shift of the } i_1 \text{ signal by } i_2 \text{ bits to the left (Shift Left)} \\
6 & \text{Shift of the } i_1 \text{ signal by } i_2 \text{ bits to the right (Shift Right)} \\
7 & \text{2’s complement of the } i_1 \text{ signal on 8 bits (2’s Complement - Byte)} \\
8 & \text{2’s complement of the } i_1 \text{ signal on 16 bits (2’s Complement - Word)} \\
9 & \text{2’s complement of the } i_1 \text{ signal on 32 bits (2’s Complement - Long)} \\
\end{array}
\]
**BMOCT, BMHEXD – Bitwise multiplexers**

**Block Symbols**

**Function Description**

Both BMOCT and BMHEXD are bitwise multiplexers for easy composition of the output signal from individual bits. The only difference is the number of inputs, the BMOCT block has 8 Boolean inputs while the BMHEXD block offers 16-bit composition. The input signals $U_i$ correspond with the individual bits of the output signal $iy$, the $U_0$ input is the least significant bit.

**Inputs**

$U_0...U_{15}$  Individual bits of the output signal  $bool$

**Output**

$iy$  Composed output signal  $long$

**Parameter**

$shift$  Bit shift of the output signal  $\downarrow0\uparrow31$  $long$
COUNT – Controlled counter

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 \textit{cnt} output is incremented (decremented) by 1. Simultaneous occurrence of rising edges at both inputs is indicated by the error output E set to \textit{on}. The R1 input resets the counter to 0 and no addition or subtraction is performed unless the R1 input returns to \textit{off} again. It is also possible to set the output \textit{cnt} to the value \textit{n0} by the SETH input. Again, no addition or subtraction is performed unless the SETH input returns to \textit{off} again. The R1 input has higher priority than the SETH input. The input HLD = \textit{on} prevents both incrementing and decrementing. When the counter reaches the value \textit{cnt} ≥ \textit{nmax}, the Q output is set to \textit{on}.

Inputs

- \textbf{R1} Block reset (R1 = \textit{on}) \hspace{1cm} \textit{bool}
- \textbf{n0} Value to set the counter to (using the SETH input) \hspace{1cm} \textit{long}
- \textbf{SETH} Set the counter value to \textit{n0} (SETH = \textit{on}) \hspace{1cm} \textit{bool}
- \textbf{UP} Incrementing input signal \hspace{1cm} \textit{bool}
- \textbf{DN} Decrementing input signal \hspace{1cm} \textit{bool}
- \textbf{HLD} Counter freeze
  - \textit{off} \ldots Counter is running
  - \textit{on} \ldots Counter is locked
- \textbf{nmax} Counter target value \hspace{1cm} \textit{long}

Outputs

- \textbf{cnt} Total number of pulses \hspace{1cm} \textit{long}
- \textbf{SGN} Sign of the \textit{cnt} output
  - \textit{off} \ldots for \textit{cnt} ≤ 0
  - \textit{on} \ldots for \textit{cnt} > 0
Q Target value indicator  
  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 \( i, i = 0, 1, \ldots, 255 \) is indicated by individual bits of the integer outputs \( q_0, q_1, \ldots, q_{15} \). Only a single bit with index \( i \mod 16 \) of the \( q(i \div 16) \) output is set to 1. The remaining bits of that output and the other outputs are zero. The bits are numbered from zero, least significant bit first. Note that the DIV and MOD operators denote integer division and remainder after integer division respectively. The current state is also indicated by the \( ksa \in \{0, 1, \ldots, 255\} \) output.

The transition conditions \( C_k, k = 0, 1, \ldots, 255 \) are activated by individual bits of the inputs \( c_0, c_1, \ldots, c_{15} \). The \( k \)-th transition condition is fulfilled when the \( (k \mod 16) \)-th bit of the input \( c(k \div 16) \) is equal to 1. The transition cannot happen otherwise.

The BMHEXD or BMOCT bitwise multiplexers can be used for composition of the input signals \( c_0, c_1, \ldots, c_{15} \) from individual Boolean signals. Similarly the output signals \( q_0, q_1, \ldots, q_{15} \) can be decomposed using the BDHEXD or BDOCT bitwise demultiplexers.

The automat function is defined by the following table of transitions:

| \( S_1 \) | \( C_1 \) | \( FS_1 \) |
| \( S_2 \) | \( C_2 \) | \( FS_2 \) |
| \( \ldots \) |
| \( S_n \) | \( C_n \) | \( FS_n \) |

Each row of this table represents one transition rule. For example the first row

| \( S_1 \) | \( C_1 \) | \( FS_1 \) |
has the meaning

If \((S_1 \text{ is the current state AND transition condition } C_1 \text{ is fulfilled})\)  
then proceed to the following state \(F S_1\).

The above described meaning of the table row holds for \(C_1 < 1000\). Negation of the \((C_1 - 1000)\)-th transition condition is assumed for \(C_1 \geq 1000\).

The above mentioned table can be easily constructed from the automat state diagram or SFC description (Sequential Function Charts, formerly Grafcet).

The \(R_1\) = \text{on} input resets the automat to the initial state \(S_0\). The \(\text{SET}\) input allows manual transition from the current state to the \(\text{ns0}\) state when rising edge occurs. The \(R_1\) input overpowers the \(\text{SET}\) input. The \(\text{HLD} = \text{on}\) input freezes the automat activity, the automat stays in the current state regardless of the \(c_i\) input signals and the \(t\text{step}\) timer is not incremented. The \(\text{TOUT}\) output indicates that the machine remains in the given state longer than expected. The time limits \(T O_i\) for individual states are defined by the \(t\text{outs}\) array. There is no time limit for the given state when \(T O_i\) is set to zero. The \(\text{TOUT}\) output is set to \text{off} whenever the automat changes its state.

It is possible to allow more state transitions in one cycle by the \(\text{morestps}\) parameter. However, this option must be thoroughly considered and tested, namely when the \(\text{TOUT}\) output is used in transition conditions. In such a case it is strongly recommended to incorporate the \(\text{ksa}\) output in the transition conditions as well.

The development tools of the REX Control System include also the \(\text{SFCEditor}\) program. You can create SFC schemes graphically using this tool. Run this editor from \(\text{RexDraw}\) by clicking the \text{Configure} button in the parameter dialog of the \(\text{EATMT}\) block.

**Inputs**

- \(R_1\)  
  Reset signal, \(R_1 = \text{on}\) brings the automat to the initial state \(S_0\); the \(R_1\) input overpowers the \(\text{SET}\) input

- \(\text{ns0}\)  
  This state is reached when rising edge occurs at the \(\text{SET}\) input

- \(\text{SET}\)  
  The rising edge of this signal forces the transition to the \(\text{ns0}\) state

- \(\text{HLD}\)  
  The \(\text{HLD} = \text{on}\) freezes the automat, no transitions occur regardless of the input signals, \(t\text{step}\) is not increasing

- \(c_0...c_{15}\)  
  Transition conditions, each input signal contains 16 transition conditions, see details above

**Outputs**

- \(q_0...q_{15}\)  
  Output signals indicating the current state of the automat, see details above

- \(\text{ksa}\)  
  Integer code of the active state

- \(t\text{step}\)  
  Time elapsed since the current state was reached; the timer is set to 0 whenever a state transition occurs

- \(\text{TOUT}\)  
  Flag indicating that the time limit for the current state was exceeded
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>morestsps</code></td>
<td>Allow multiple transitions in one cycle of the automat</td>
<td><code>bool</code></td>
</tr>
<tr>
<td><code>ntr</code></td>
<td>Number of state transition table rows</td>
<td><code>long</code></td>
</tr>
<tr>
<td><code>sfcname</code></td>
<td>Filename of block configurator data file (filename is generated by system if parameter is empty)</td>
<td><code>string</code></td>
</tr>
<tr>
<td><code>STT</code></td>
<td>State transition table (matrix)</td>
<td><code>short</code></td>
</tr>
<tr>
<td><code>touts</code></td>
<td>Vector of timeouts T00, T01, ..., T0255 for the states S0, S1, ..., S255</td>
<td><code>double</code></td>
</tr>
</tbody>
</table>

```plaintext
morestsps: off ... Disabled
on ... Enabled

ntr: 0 \[0 0 1; 1 1 2; 2 2 3; 3 3 0\] short

sfcname: "" string

STT: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16] double

touts: [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

\[
\begin{array}{c}
 U \\
 Y \\
 \text{EDGE}_\
\end{array}
\]

Licence: STANDARD

Function Description

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

Input

\( U \)  Logical input of the block  bool

Output

\( Y \)  Logical output of the block  bool

Parameter

\( \texttt{iedge} \)  Type of edges to detect  ○1  long

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 \( \text{shift} \) bits right (if \( \text{shift} \) is positive) or left (if \( \text{shift} \) is negative). Free space resulting from shifting is filled with zeros.

Output value \( n \) is calculated as bitwise AND of shifted input \( i \) and bit mask \( \text{mask} \).

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

Input

\( i \) Integer value to shift and mask \hspace{1cm} \text{long}

Parameters

\begin{align*}
\text{shift} & : \text{Bit shift (negative=left, positive=right)} \quad \llarrow 31 \rlarrow 31 \hspace{1cm} \text{long} \\
\text{mask} & : \text{Bit mask (applied after bit shift)} \quad \llarrow XXX \rlarrow XXX \circ XXX \hspace{1cm} \text{dword}
\end{align*}

Output

\( n \) Resulting integer value \hspace{1cm} \text{long}
ISSW – Simple switch for integer signals

Block Symbol

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

i1  First integer input of the block  long
i2  Second integer input of the block  long
SW  Signal selector  bool
    off  . . . The i1 signal is selected
    on  . . . The i2 signal is selected

Output

n  Integer output of the block  long
INTSM – Integer number bit shift and mask

Block Symbol

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

Output value \( n \) is calculated as bitwise AND of shifted input \( i \) and bit mask \( \text{mask} \).

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

Input
\( i \)  Integer value to shift and mask  long

Parameters
\( \text{shift} \)  Bit shift (negative=left, positive=right)  \( \downarrow -31 \uparrow 31 \) long
\( \text{mask} \)  Bit mask (applied after bit shift)  \( \downarrow \text{XXX} \uparrow \text{XXX} \odot \text{XXX} \) dword

Output
\( n \)  Resulting integer value  long
ITOI – Transformation of integer and binary numbers

Block Symbol

License: STANDARD

Function Description

The ITOI block transforms the input number $k$, or the binary number $(U3 \ U2 \ U1 \ U0)_2$, from the set $\{0, 1, 2, \ldots, 15\}$ to the output number $nk$ and its binary representation $(Y3 \ Y2 \ Y1 \ Y0)_2$ from the same set. The transformation is described by the following table

<table>
<thead>
<tr>
<th>$k$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>…</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$nk$</td>
<td>$n0$</td>
<td>$n1$</td>
<td>$n2$</td>
<td>…</td>
<td>$n15$</td>
</tr>
</tbody>
</table>

where $n0, \ldots, n15$ are given by the mapping table target vector $f_{k\text{tab}}$.

If $BINF = \text{off}$, then the integer input $k$ is active, while for $BINF = \text{on}$ the input is defined by the binary inputs $(U3 \ U2 \ U1 \ U0)_2$.

Inputs

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

Outputs

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

Parameters

- $BINF$: Enable the binary selectors
  - off: Disabled (integer input $k$)
  - on: Enabled (binary input signals $U3 \ldots U0$)
fktab Vector of mapping table target values
⊙[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
NOT_ – Boolean complementation

Block Symbol

\[ \text{NOT}_\{u, y\} \]

Licence: STANDARD

Function Description

The NOT block negates the input signal.

Input

\( u \) Logical input of the block \( \text{bool} \)

Output

\( y \) Logical output of the block (\( y = \neg u \)) \( \text{bool} \)
**OR_** – Logical sum of two signals

**Block Symbol**

![Block Symbol](image)

**Licence:** STANDARD

**Function Description**

The **OR** block computes the logical sum of two input signals $U_1$ and $U_2$.

If you need to work with more input signals, use the **OROCT** block.

**Inputs**

- **$U_1$**: First logical input of the block  
- **$U_2$**: Second logical input of the block

**Outputs**

- **$Y$**: Logical output of the block ($U_1 \lor U_2$)  
- **NY**: Boolean complementation of $Y$ ($NY = \neg Y$)
OROCT – Logical sum of eight signals

Function Description

The OROCT block computes the logical sum of eight input signals $U_1, U_2, \ldots, U_8$. The signals listed in the $nl$ parameter are negated prior to computing the logical sum.

For an empty $nl$ parameter a simple logical sum $Y = U_1 \lor U_2 \lor U_3 \lor U_4 \lor U_5 \lor U_6 \lor U_7 \lor U_8$ is computed. For e.g. $nl=2,6..8$, the logical function is $Y = U_1 \lor \neg U_2 \lor U_3 \lor U_4 \lor U_5 \lor \neg U_6 \lor \neg U_7 \lor \neg U_8$.

If you have only two input signals, consider using the OR_ block.

Inputs

- $U_1$: First logical input of the block  
  - Type: bool
- $U_2$: Second logical input of the block  
  - Type: bool
- $U_3$: Third logical input of the block  
  - Type: bool
- $U_4$: Fourth logical input of the block  
  - Type: bool
- $U_5$: Fifth logical input of the block  
  - Type: bool
- $U_6$: Sixth logical input of the block  
  - Type: bool
- $U_7$: Seventh logical input of the block  
  - Type: bool
- $U_8$: Eighth logical input of the block  
  - Type: bool

Outputs

- $Y$: Output signal, logical sum  
  - Type: bool
- $NY$: Boolean complementation of $Y$  
  - Type: bool

Parameter

- $nl$: List of signals to negate. The format of the list is e.g. 1,3..5,8.  
  - Type: 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.
**RS – Reset-set flip-flop circuit**

**Block Symbol**

[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
  - Type: bool

- **R1**
  - Priority flip-flop reset, sets the Q output to off, overpowers the S input
  - Type: bool

**Outputs**

- **Q**
  - Flip-flop circuit state
  - Type: bool

- **NQ**
  - Boolean complementation of Q
  - Type: bool
SR – Set-reset flip-flop circuit

Block Symbol

Function Description

The SR block is a flip-flop circuit, which sets its output permanently to on as soon as the input signal S1 is on. The other input signal R resets the Q output to off, but only if the S1 input is off. The NQ output is simply the negation of the signal Q.

The block function is evident from the inner block structure depicted below.

Inputs

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

Outputs

Q     Flip-flop circuit state   bool
NQ    Boolean complementation of Q   bool
**CHAPTER 8. LOGIC – LOGIC CONTROL**

**TIMER_ – Multipurpose timer**

**Block Symbol**

![Block Symbol](image)

**Licence:** STANDARD

**Function Description**

The **TIMER_** block either generates an output pulse of the given width \( pt \) (in seconds) or filters narrow pulses in the \( U \) input signal whose width is less than \( pt \) seconds. The operation mode is determined by the **mode** parameter. The timer can be paused by the **HLD** input.

The graph illustrates the behaviour of the block in individual modes for \( pt = 3 \):

![Graph](image)

**Inputs**

- \( U \): Trigger of the timer
- \( HLD \): Timer hold

**Outputs**

- \( Q \): Timer output
- \( et \): Elapsed time [s]
Parameters

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

<table>
<thead>
<tr>
<th>$pt$</th>
<th>Timer interval $[s]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\circ 1.0$</td>
<td>double</td>
</tr>
</tbody>
</table>
Chapter 9

TIME – Blocks for handling time

Contents

<table>
<thead>
<tr>
<th>Block</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE_ – Current date</td>
<td>262</td>
</tr>
<tr>
<td>DATETIME – Get, set and convert time</td>
<td>263</td>
</tr>
<tr>
<td>TIME – Current time</td>
<td>266</td>
</tr>
<tr>
<td>WSCH – Weekly schedule</td>
<td>267</td>
</tr>
</tbody>
</table>
DATE_ – Current date

Block Symbol Licence: STANDARD

Function Description
The outputs of the DATE_ function block correspond with the actual date of the operating system. Use the DATETIME block for advanced operations with date and time.

Outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>Year</td>
<td>long</td>
</tr>
<tr>
<td>month</td>
<td>Month</td>
<td>long</td>
</tr>
<tr>
<td>day</td>
<td>Day</td>
<td>long</td>
</tr>
<tr>
<td>dow</td>
<td>Day of week, first day of week is Sunday (1)</td>
<td>long</td>
</tr>
</tbody>
</table>

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tz</td>
<td>Timezone</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Local time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UTC</td>
<td>2</td>
</tr>
</tbody>
</table>
DATETIME – Get, set and convert time

Function Description

The DATETIME block is intended for advanced date/time operations in the REX control system.

It allows synchronization of the operating system clock and the clock of the REX control system. When the executive of the REX control system is initialized, both clocks are the same. But during long-term operation the clocks may lose 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 REX control system according to the block inputs and parameters.

It is highly recommended not to adjust the REX control 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 REX control system is not adjusted at once, a gradual synchronization is used instead. In such a case, the timing of the REX control system executive is negligibly altered and the clocks synchronization is achieved after some time. Afterwards the timing of the REX 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 |
CHAPTER 9. TIME – BLOCKS FOR HANDLING TIME

<table>
<thead>
<tr>
<th>Input for setting day</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input for setting hours</td>
<td>long</td>
</tr>
<tr>
<td>Input for setting minutes</td>
<td>long</td>
</tr>
<tr>
<td>Input for setting seconds</td>
<td>long</td>
</tr>
<tr>
<td>Input for setting nanoseconds</td>
<td>↓-9.22E+18 ↑9.22E+18 large</td>
</tr>
</tbody>
</table>

| Trigger for setting time | bool |
| Trigger for getting time | bool |

Outputs

| Year | long |
| Month | long |
| Day | long |
| Hours | long |
| Minutes | long |
| Seconds | long |
| Nanoseconds | long |
| Day of week | long |
| Week of year | long |
| Total number of days | long |
| Total number of seconds | long |
| Total number of nanoseconds | large |
| Number of seconds since midnight | long |

Parameters

| Source for setting time | long |
| Target for setting the REX clock [s] | double |
| Period for setting time [s] (0=not periodic) | double |
| Period for getting time [s] (0=not periodic) | double |
| First day of week is Sunday | bool |

- Week starts on Monday
- Week starts on Sunday
tz | Timezone | ☀1 | long
---|----------|----|---
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

<table>
<thead>
<tr>
<th>hour</th>
<th>Hours</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>Minutes</td>
<td>long</td>
</tr>
<tr>
<td>sec</td>
<td>Seconds</td>
<td>long</td>
</tr>
</tbody>
</table>

Parameter

<table>
<thead>
<tr>
<th>tz</th>
<th>Timezone</th>
<th>⊙1</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Local time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>UTC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**WSCH – Weekly schedule**

Block Symbol

Function Description

The WSCH function block is a weekly scheduler for e.g. heating (day, night, eco), ventilation (high, low, off), lighting, irrigation etc. Its outputs can be used for switching individual appliances on/off or adjusting the intensity or power of the connected devices.

During regular weekly schedule the outputs \(iy\) and \(y\) reflect the values from the \(wst\) table. This table contains triplets \(day\)-\(hour\)-\(value\). E.g. the notation \([2\ 6.5\ 21.5]\) states that on Tuesday, at 6:30 in the morning (24-hour format), the output \(y\) will be set to 21.5. The output \(iy\) will be set to 22 (rounding to nearest integer). The individual triplets are separated by semicolons.

The days in a week are numbered from 1 (Monday) to 7 (Sunday). Higher values can be used for special daily schedules, which can be forced using the \(fsch\) input or the \(specdays\) table. The active daily program is indicated by the \(isch\) output.

Alternatively it is possible to temporarily force a specific output value using the \(val\) input and a rising edge at the \(SET\) input (\(\text{off} \rightarrow \text{on}\)). 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 \(\text{long}\) to type \(\text{bool}\) is done automatically).
The \texttt{nmax} parameter defines memory allocation for the \texttt{wst} and \texttt{specdays} arrays. For \texttt{nmax} = 100 the \texttt{wst} list can contain up to 100 triplets \textit{day-hour-value}. In typical applications there is no need to modify the \texttt{nmax} parameter.

\textbf{Inputs}

\begin{itemize}
  \item \texttt{SET} Trigger for setting the \texttt{y} and \texttt{iy} outputs \hspace{1cm} \texttt{bool}
  \item \texttt{val} Temporary value to set the \texttt{y} and \texttt{iy} outputs to \hspace{1cm} \texttt{double}
  \item \texttt{fsch} Forced schedule \hspace{1cm} \texttt{long}
    \begin{itemize}
      \item 0 ..... standard weekly schedule
      \item 1 ..... Monday
      \item 2 ..... Tuesday
      \item ..... ...
      \item 7 ..... Sunday
      \item 8 and above additional daily programs as defined by the \texttt{wst} table
    \end{itemize}
\end{itemize}

\textbf{Outputs}

\begin{itemize}
  \item \texttt{iy} Integer output value \hspace{1cm} \texttt{long}
  \item \texttt{y} Output value \hspace{1cm} \texttt{double}
  \item \texttt{isch} Daily schedule identifier \hspace{1cm} \texttt{long}
  \item \texttt{trem} Time remaining in the current section (in seconds) \hspace{1cm} \texttt{double}
  \item \texttt{ynext} Output in the next section \hspace{1cm} \texttt{double}
\end{itemize}

\textbf{Parameters}

\begin{itemize}
  \item \texttt{tz} Timezone \hspace{1cm} \texttt{⊙1 long}
    \begin{itemize}
      \item 1 ..... Local time
      \item 2 ..... UTC
    \end{itemize}
  \item \texttt{nmax} Allocated size of arrays \hspace{1cm} \texttt{⊙10↑1000000⊙100 long}
  \item \texttt{wst} Weekly schedule table (list of triplets \textit{day-hour-value}) \hspace{1cm} \texttt{double}
    \begin{itemize}
      \item \textcolor{red}{⊙1 [1 0.01 18.0; 2 6.0 22.0; 2 18.0 18.0; 3 6.0 22.0; 3 18.0 18.0; 4 6.0 22.0; 4 18.0 18.0; 5 6.0 22.0; 5 18.0 18.0; 6 6.0 22.0; 6 18.0 18.0; 1 0.01 18.0]}
    \end{itemize}
  \item \texttt{specdays} List of special days (list of pairs \textit{date-daily program}) \hspace{1cm} \texttt{long}
    \begin{itemize}
      \item \textcolor{red}{⊙[20150406 1; 20151224 1; 20151225 1; 20151226 1; 20160101 1; 20160328 1; 20170417 1; 20180402 1; 20190422 1; 20200413 1]}
    \end{itemize}
\end{itemize}
Chapter 10

ARC – Data archiving

Contents

10.1 Functionality of the archiving subsystem ........................................ 270
10.2 Generating alarms and events ......................................................... 271
   ALB, ALBI – Alarms for Boolean value ........................................... 271
   ALN, ALNI – Alarms for numerical value ........................................ 273
10.3 Trends recording ............................................................................. 275
   ACD – Archive compression using Delta criterion .............................. 275
   TRND – Real-time trend recording ................................................... 277
   TRNDV – Real-time trend recording with vector input ....................... 280
   TRNDLF – ° Real-time trend recording (lock-free) ............................... 282
   TRNDVLF – ° Real-time trend recording (for vector signals, lock-free) ........ 284
10.4 Archive management ................................................................. 285
   AFLUSH – Forced archive flushing ................................................. 285

The RexCore executive of the REX Control 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 REX Control 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

269
10.1 Functionality of the archiving subsystem

The archive in the REX Control 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 RexDraw 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

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 $lvl$ parameter in the REX Control System. The range $1 \leq lvl \leq 127$ is reserved for alarms. All starts, ends and acknowledgements of the alarms are stored in the archive. On the contrary, the range $128 \leq lvl \leq 255$ indicates events. Only the start (the time instant) of the event is stored in the archive.

Inputs

$U$ Logical input of the block whose changes are watched  
$men$ Enable alarms  
$iACK$ Acknowledge alarm

<table>
<thead>
<tr>
<th>$men$</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>All alarms disabled</td>
</tr>
<tr>
<td>1</td>
<td>Low-alarm enabled (LA) (falling edge in the input signal $U$)</td>
</tr>
<tr>
<td>2</td>
<td>High-alarm enabled (HA) (rising edge in the input signal $U$)</td>
</tr>
<tr>
<td>3</td>
<td>All alarms enabled</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$iACK$</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Low-alarm acknowledge</td>
</tr>
<tr>
<td>2</td>
<td>High-alarm acknowledge</td>
</tr>
<tr>
<td>3</td>
<td>Both alarms acknowledge</td>
</tr>
</tbody>
</table>

Alarm is acknowledged on rising edge
CHAPTER 10. ARC – DATA ARCHIVING

Outputs

<table>
<thead>
<tr>
<th>iac</th>
<th>Current alarm code</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>All alarms inactive</td>
<td>. .</td>
</tr>
<tr>
<td>1</td>
<td>Low-alarm active (LA)</td>
<td>. .</td>
</tr>
<tr>
<td>2</td>
<td>High-alarm active (HA)</td>
<td>. .</td>
</tr>
<tr>
<td>256</td>
<td>Low-alarm not acknowledged (NACK)</td>
<td>. .</td>
</tr>
<tr>
<td>512</td>
<td>High-alarm not acknowledged (NACK)</td>
<td>. .</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HA</th>
<th>High-alarm indicator</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA</td>
<td>Low-alarm indicator</td>
<td>bool</td>
</tr>
<tr>
<td>NACK</td>
<td>Alarm-not-acknowledged indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>

Parameters

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

<table>
<thead>
<tr>
<th>id</th>
<th>Identification code of the alarm in the archive. This identifier must be unique in the whole target device with the REX control system (i.e. in all archiving blocks).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>word</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lvl</th>
<th>The level of the alarms (HA and LA) which differentiates alarms from events and defines the severity of the alarm/event</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>byte</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Desc</th>
<th>Extended description of the alarm which is displayed by the diagnostic tools of the REX control system</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>string</td>
</tr>
</tbody>
</table>

⊙1 | Alarm Description
10.2. GENERATING ALARMS AND EVENTS

ALN, ALNI – Alarms for numerical value

Block Symbols

Function Description

The ALN and ALNI blocks generate two-level alarms or events when a limit value is exceeded (or not reached). There are four limit values the input signal $u$ is compared to, namely high-limits $h$ and $hh$ and low-limits $l$ and $ll$. 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 $lvl$ parameter in the REX Control System. The range $1 \leq lvl \leq 127$ is reserved for alarms. All starts, ends and acknowledgements of the alarms are stored in the archive. On the contrary, the range $128 \leq lvl \leq 255$ indicates events. Only the start (the time instant) of the event is stored in the archive.

Inputs

$u$ Analog input of the block which is checked to remain within the given limits
$hys$ Alarm hysteresis for switching the alarm off $\downarrow0.0\uparrow10000000000.0$ double
$hh$ The second high-alarm limit, must be greater than $h$ double
$h$ High-alarm limit, must be greater than $l$ double
$l$ Low-alarm limit, must be greater than $ll$ double
$ll$ The second low-alarm limit double
$iack$ Alarm is acknowledged on rising edge of the individual bits of this input/parameter. E.g. value 15 acknowledges all alarms.

byte

1 ....... Second low-alarm acknowledge
2 ....... Low-alarm acknowledge
4 ....... High-alarm acknowledge
8 ....... Second high-alarm acknowledge
In case a one-level alarm is required, it is sufficient to set \( lvl2 = 0 \) or set the \( hh \) and \( ll \) limits to extreme values which can never be reached by the input signal.

**Outputs**

- **iac**: Current alarm code. Additional bitwise combinations of the codes may appear. E.g. 12 means both high alarms.
  - 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

- **long**: Signal within limits
- **ha**: Low-alarm active
- **long**: High-alarm active
- **ha**: Second low-alarm active
- **ha**: Second high-alarm active
- **ha**: Low-alarm not acknowledged
- **ha**: High-alarm not acknowledged
- **ha**: Second low-alarm not acknowledged
- **ha**: Second high-alarm not acknowledged
- **E**: Error flag
  - off ... No error
  - on ... An error occurred, alarm limits disordered

- **HHA**: The second high-alarm indicator
- **long**: HA: High-alarm indicator
- **long**: LA: Low-alarm indicator
- **long**: LLA: The second low-alarm indicator
- **long**: NACK: Alarm-not-acknowledged indicator

**Parameters**

- **acls**: Alarm class (data type to store)
  - 1 ..... Bool
  - 2 ..... Byte
  - 3 ..... Short
  - 4 ..... Long
  - 5 ..... Word
  - 6 ..... DWord
  - 7 ..... Float
  - 8 ..... Double
  - 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 ARC block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.

- **id**: Identification code of the alarm in the archive. This identifier must be unique in the whole target device with the REX control system (i.e. in all archiving blocks).

- **lvl1**: The level of first high- and low-alarms (HA and LA) which differentiates alarms from events and defines the severity of the alarm/event

- **lvl2**: The level of second high- and low-alarms (HHA and LLA) which differentiates alarms from events and defines the severity of the alarm/event

- **Desc**: Extended description of the alarm which is displayed by the diagnostic tools of the REX control system
10.3 Trends recording

**ACD – Archive compression using Delta criterion**

**Block Symbol**

![Block Symbol](image)

**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 \( (t_{\text{min}}, t_{\text{max}}) \) seconds (rounded to the nearest multiple of the sampling period). A constant input signal is stored every \( t_{\text{max}} \) seconds while rapidly changing signal is stored every \( t_{\text{min}} \) seconds.

When the execution of the block is started, the first input value is stored. This value will be referred to as \( u_0 \) in the latter. The rules for storing the following samples are given by the **delta** and **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 \( t_{\text{min}} \) seconds ago, the value of input \( u \) is stored and \( u_0 = u \) is set. If the condition is fulfilled sooner than \( t_{\text{min}} \) seconds after the last stored value, the error output \( E \) is set to 1 and the first value following the \( t_{\text{min}} \) interval is stored. At that time the output \( E \) is set back to 0 and the whole procedure is repeated.

For **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 ×.

![Diagram](image)

**Inputs**

- **u** Signal to compress and store \( \text{double} \)
- **delta** Threshold for storing the signal \( \downarrow0.0 \uparrow10000000000.0 \) \( \text{double} \)
CHAPTER 10. ARC – DATA ARCHIVING

Outputs

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

Parameters

acls  Archive class determining the variable type to store
    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 ARC block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.

id  Identification code of the event in the archive. This identifier must be unique in the whole target device with the REX control system (i.e. in all archiving blocks).

tmin  The shortest interval between two samples of the u input signal stored in the archive [s]
    0.001 1000000.0

tmax  The longest interval between two samples of the u input signal stored in the archive [s]
    1.0 1000000.0

TR  Trend evaluation flag
    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 REX control system
TRND – Real-time trend recording

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 ptype1 to ptype4 parameters. The other decimation factor afac can be used for storing data in archives.

The type of trend buffers can be specified in order to conserve memory of the target device. The memory requirements of the trend buffers are defined by the formula $s \cdot n \cdot l$, 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 14 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):

<table>
<thead>
<tr>
<th>Error</th>
<th>Range underflow</th>
<th>Range overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>u4 u3 u2 u1</td>
<td>u4 u3 u2 u1</td>
</tr>
<tr>
<td>Bit number</td>
<td>11 10 9 8</td>
<td>3 2 1 0</td>
</tr>
<tr>
<td>Bit weight</td>
<td>2048 1024 512</td>
<td>8 4 2 1</td>
</tr>
</tbody>
</table>

In case of simultaneous errors the resulting error code is given by the sum of the weights of individual errors. Note that underflow and overflow cannot happen simultaneously on
a single input.

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

**Inputs**

u1..u4 Analog inputs to be processed and stored in the trend double

RUN Enable execution. The data are processed and stored if and only if RUN = on. bool

R1 Input for clearing the trend contents. The buffers are cleared when R1 = on. This flag overpowers the RUN input. bool

**Outputs**

y1..y4 Analog outputs of the block set once in pfac executions of the block double

to the last values stored in the trend buffers

iE Error code, see the table above long

**Parameters**

n Number of signals to process and store in the trend buffers long

↓1 ↑4 ⊖4

l Number of samples reserved in memory for each trend buffer long

↓0 ↑268435000 ⊖1000

btype Type of all n trend buffers long

1 ....... Bool 4 ....... Long 7 ....... Float

2 ....... Byte 5 ....... Word 8 ....... Double

3 ....... Short 6 ....... DWord 10 ....... Large

ptypei The way the signal ui, i = 1...4, is processed. The last pfac samples are processed as selected and the result is stored in the i-th trend buffer. long

1 ....... No processing, just storing data

2 ....... Minimum from the last pfac samples

3 ....... Maximum from the last pfac samples

4 ....... Sum of the last pfac samples

5 ....... Simple average of the last pfac samples

6 ....... Root mean square of the last pfac samples

7 ....... Variance of the last pfac samples

pfac Multiple of the block execution period defining the period for storing the data in the trend buffers. Data are stored with the period of pfac · TS unless RUN = off, where TS is the block execution period in seconds. long

↓1 ↑1000000 ⊖1

afac Every afac-th sample stored in the trend buffer is also stored in the archives specified by the arc parameter. There are no data stored in the archives for afac = 0. Data are stored with the period of afac · pfac · TS, where TS is the block execution period in seconds. long

↓0 ↑1000000
### 10.3. TRENDS RECORDING

| **arc** | List of archives to store the trend data. The format of the list is e.g. 1,3...5,8. The data will be stored in all listed archives (see the **ARC** block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case. |
| **id** | Identification code of the trend block. This identifier must be unique in the whole target device with the **REX** control system (i.e. in all archiving blocks). |
| **Title** | Title of the trend to be displayed in the diagnostic tools of the **REX** Control System, e.g. in the **RexView** program |
| **timesrc** | Source of timestamps. Each data sample in trend buffer is stored with a timestamp. For fast or short term trends where you are interested in sample-by-sample timing more than in absolute time, choose **CORETIMER** — REX internal technological time which is incremented by nominal period each base tick. For long running trends where you are interested mostly in absolute time shared with operating system (and possibly synchronized by NTP), choose **RTC**. Other values are intended for debug or special purposes. |
| | 1. . . . **CORETIMER** — technological time — at current tick |
| | 2. . . . **CORETIMER-PRECISE** — technological time — at block execution |
| | 3. . . . **RTC** — real time clock (wallclock) from operating system — at current tick |
| | 4. . . . **RTC-PRECISE** — real time clock (wallclock) from operating system — at block execution |
| | 4. . . . **PFC** — raw high precision time (PerFormanceCounter) |
TRNDV – Real-time trend recording with vector input

Block Symbol

Function Description

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

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

The type of trend buffers can be specified in order to conserve memory of the target device. The memory requirements of the trend buffers are defined by the formula s · n · l, 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 l = 1000, 8 · 4 · 1000 = 32000 bytes are required. In case the input signals come from 16-bit A/D converter the Word type can be used and memory requirements drop to one quarter. Memory requirements and allowed ranges of individual types are summarized in table 1.1 on page 14 of this reference guide.

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

Inputs

<table>
<thead>
<tr>
<th>uVec</th>
<th>Vector signal to record</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLD</td>
<td>Input for freezing the cyclic buffers, no data is appended when HLD = true</td>
<td>bool on</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>iE</th>
<th>Error code</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>REX general error</td>
<td></td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Number of signals (trend buffers)</td>
<td>( \downarrow 1 \uparrow 64 \odot 8 ) long</td>
</tr>
<tr>
<td>l</td>
<td>Number of samples per trend buffer</td>
<td>( \downarrow 2 \uparrow 268435000 \odot 1000 ) long</td>
</tr>
<tr>
<td>btype</td>
<td>Type of all trend buffers</td>
<td>( \odot 8 ) long</td>
</tr>
<tr>
<td>pfac</td>
<td>Multiple of the block execution period defining the period for storing the data in the trend buffers. Data are stored with the period of ( \text{pfac} \cdot T_S ) unless ( \text{RUN} = \text{off} ), where ( T_S ) is the block execution period in seconds.</td>
<td>( \downarrow 1 \uparrow 1000000 \odot 1 ) long</td>
</tr>
<tr>
<td>afac</td>
<td>Every ( \text{afac} )-th sample stored in the trend buffer is also stored in the archives specified by the ( \text{arc} ) parameter. There are no data stored in the archives for ( \text{afac} = 0 ). Data are stored with the period of ( \text{afac} \cdot \text{pfac} \cdot T_S ), where ( T_S ) is the block execution period in seconds.</td>
<td>( \downarrow 0 \uparrow 1000000 ) long</td>
</tr>
<tr>
<td>arc</td>
<td>List of archives to store the trend data. The format of the list is e.g. 1,3,5,8. The data will be stored in all listed archives (see the ( \text{ARC} ) block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.</td>
<td>( \odot 1 ) word</td>
</tr>
<tr>
<td>id</td>
<td>Identification code of the trend block. This identifier must be unique in the whole target device with the ( \text{REX} ) control system (i.e. in all archiving blocks).</td>
<td>( \odot 1 ) word</td>
</tr>
<tr>
<td>Title</td>
<td>Title of the trend to be displayed in the diagnostic tools of the ( \text{REX} ) Control System, e.g. in the ( \text{RexView} ) program</td>
<td>( \odot ) Trend Title string</td>
</tr>
</tbody>
</table>
TRNDLF – * Real-time trend recording (lock-free)

Block Symbol

Function Description

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

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>First analog input of the block</td>
<td>double</td>
</tr>
<tr>
<td>u2</td>
<td>Second analog input of the block</td>
<td>double</td>
</tr>
<tr>
<td>u3</td>
<td>Third analog input of the block</td>
<td>double</td>
</tr>
<tr>
<td>u4</td>
<td>Fourth analog input of the block</td>
<td>double</td>
</tr>
<tr>
<td>u5</td>
<td>Fifth analog input of the block</td>
<td>double</td>
</tr>
<tr>
<td>u6</td>
<td>Sixth analog input of the block</td>
<td>double</td>
</tr>
<tr>
<td>u7</td>
<td>Seventh analog input of the block</td>
<td>double</td>
</tr>
<tr>
<td>u8</td>
<td>Eighth analog input of the block</td>
<td>double</td>
</tr>
<tr>
<td>RUN</td>
<td>Enable execution</td>
<td>bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Number of signals (trend buffers)</td>
<td>long</td>
</tr>
<tr>
<td>l</td>
<td>Number of samples per trend buffer</td>
<td>long</td>
</tr>
</tbody>
</table>
10.3. TRENDS RECORDING

<table>
<thead>
<tr>
<th>btype</th>
<th>Type of all trend buffers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bool</td>
</tr>
<tr>
<td>2</td>
<td>Byte</td>
</tr>
<tr>
<td>3</td>
<td>Short</td>
</tr>
<tr>
<td>4</td>
<td>Long</td>
</tr>
<tr>
<td>5</td>
<td>Word</td>
</tr>
<tr>
<td>6</td>
<td>DWord</td>
</tr>
<tr>
<td>7</td>
<td>Float</td>
</tr>
<tr>
<td>8</td>
<td>Double</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Title</th>
<th>Trend title string</th>
<th>Trend Title</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>timesrc</td>
<td>Source of timestamps</td>
<td>○1</td>
<td>long</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>y1</th>
<th>First analog output of the block</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>y2</td>
<td>Second analog output of the block</td>
<td>double</td>
</tr>
<tr>
<td>y3</td>
<td>Third analog output of the block</td>
<td>double</td>
</tr>
<tr>
<td>y4</td>
<td>Fourth analog output of the block</td>
<td>double</td>
</tr>
<tr>
<td>y5</td>
<td>Fifth analog output of the block</td>
<td>double</td>
</tr>
<tr>
<td>y6</td>
<td>Sixth analog output of the block</td>
<td>double</td>
</tr>
<tr>
<td>y7</td>
<td>Seventh analog output of the block</td>
<td>double</td>
</tr>
<tr>
<td>y8</td>
<td>Eighth analog output of the block</td>
<td>double</td>
</tr>
<tr>
<td>iE</td>
<td>Error code (bitwise multiplexed)</td>
<td>long</td>
</tr>
</tbody>
</table>
TRNDVLF — * Real-time trend recording (for vector signals, lock-free)

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

Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uVec</td>
<td>Vector signal to record</td>
<td>reference</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Number of signals (trend buffers)</td>
<td>↓1 ↑64 ⊙8 long</td>
</tr>
<tr>
<td>l</td>
<td>Number of samples per trend buffer</td>
<td>↓2 ↑268435000 ⊙1024 long</td>
</tr>
<tr>
<td>btype</td>
<td>Type of all trend buffers</td>
<td>⊙8 long</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Trend title string</td>
<td>⊙Trend Title string</td>
</tr>
<tr>
<td>timesrc</td>
<td>Source of timestamps</td>
<td>⊙1 long</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>iE</td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td>i</td>
<td>REX general error</td>
<td>error</td>
</tr>
</tbody>
</table>
10.4 Archive management

AFLUSH – Forced archive flushing

Block Symbol

<table>
<thead>
<tr>
<th>Licence: STANDARD</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFLUSH</td>
</tr>
</tbody>
</table>

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

| FLUSH | Force archive flushing | bool |

Parameter

| arc    | List of archives to store the events. The format of the list is e.g. word 1,3..5,8. The event will be stored in all listed archives (see the ARC block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case. |
Chapter 11
STRING – Blocks for string operations

Contents

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNS</td>
<td>String constant</td>
<td>288</td>
</tr>
<tr>
<td>CONCAT</td>
<td>Concat string by pattern</td>
<td>289</td>
</tr>
<tr>
<td>FIND</td>
<td>Find a Substring</td>
<td>290</td>
</tr>
<tr>
<td>LEN</td>
<td>String length</td>
<td>291</td>
</tr>
<tr>
<td>MID</td>
<td>Substring Extraction</td>
<td>292</td>
</tr>
<tr>
<td>PJROCT</td>
<td>* Parse JSON string (real output)</td>
<td>293</td>
</tr>
<tr>
<td>PJSOCT</td>
<td>* Parse JSON string (string output)</td>
<td>294</td>
</tr>
<tr>
<td>REGEXP</td>
<td>Regular expression parser</td>
<td>295</td>
</tr>
<tr>
<td>REPLACE</td>
<td>Replace substring</td>
<td>296</td>
</tr>
<tr>
<td>RTOS</td>
<td>Real Number to String Conversion</td>
<td>297</td>
</tr>
<tr>
<td>SELSOCT</td>
<td>* String selector</td>
<td>298</td>
</tr>
<tr>
<td>STOR</td>
<td>String to real number conversion</td>
<td>300</td>
</tr>
</tbody>
</table>
CNS – String constant

Block Symbol

Licence: STANDARD

Function Description

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

Parameters

<table>
<thead>
<tr>
<th>scv</th>
<th>String (constant) value</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmax</td>
<td>Allocated size of string [bytes]</td>
<td>↓0 ↑65520 long</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>sy</th>
<th>String output value</th>
<th>string</th>
</tr>
</thead>
</table>
CONCAT – * Concat string by pattern

Block Symbol

Function Description
Concatenates up to 8 input strings $su_1$ to $su_8$ by pattern specified in $ptrn$ parameter.

Inputs

<table>
<thead>
<tr>
<th>su1..8</th>
<th>String input value</th>
</tr>
</thead>
</table>

Parameters

<table>
<thead>
<tr>
<th>ptrn</th>
<th>Concatenation pattern</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmax</td>
<td>Allocated size of string [bytes]</td>
<td>long</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>sy</th>
<th>String output value</th>
</tr>
</thead>
</table>
FIND – Find a Substring

Block Symbol Licence: STANDARD

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

Inputs
- su1: String input value
- su2: String input value

Parameter
- nmax: Allocated size of string [bytes] [0, 65520] long

Output
- pos: Position of substring long
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>su</td>
<td>String input value</td>
<td>string</td>
</tr>
</tbody>
</table>

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmax</td>
<td>Allocated size of string [bytes]</td>
<td>↓0 ↑65520</td>
<td>long</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>len</td>
<td>Length of input string</td>
<td>long</td>
</tr>
</tbody>
</table>
CHAPTER 11. STRING – BLOCKS FOR STRING OPERATIONS

MID – Substring Extraction

Block Symbol

Function Description
The MID block extracts a substring sy from su. The parameters l and p specify position and length of the string being extracted in UTF-8 characters. The parameter p is one-based.

Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>su</td>
<td>String input value</td>
<td>string</td>
</tr>
<tr>
<td>l</td>
<td>Length of output string</td>
<td>long</td>
</tr>
<tr>
<td>p</td>
<td>Position of output string (one-based)</td>
<td>long</td>
</tr>
</tbody>
</table>

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Lower</th>
<th>Upper</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmax</td>
<td>Allocated size of string [bytes]</td>
<td>0</td>
<td>65520</td>
<td>long</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sy</td>
<td>String output value</td>
<td>string</td>
</tr>
</tbody>
</table>
PJROCT – * Parse JSON string (real output)

Function Description

Parses input JSON string `jtxt` according to specified `name*` parameters when the input `RUN` is on. Output signals are `real` type.

Inputs

- `jtxt`: JSON formatted string
- `RUN`: Enable execution

Parameters

- `name1..8`: Name of JSON object
- `nmax`: Allocated size of string [bytes] (0-65520 long)
- `yerr`: Substitute value for an error case

Outputs

- `y1..8`: Block output signal
- `iE`: Error code
CHAPTER 11. STRING – BLOCKS FOR STRING OPERATIONS

PJSOCT – * Parse JSON string (string output)

Block Symbol

---

Function Description

Parses input JSON string \( jtxt \) according to specified \( \text{name}^* \) parameters when the input \( \text{RUN} \) is on. Output signals are \text{string} type.

Inputs

- \( jtxt \) : JSON formatted string
- \( \text{RUN} \) : Enable execution

Parameters

- \( \text{name1..8} \) : Name of JSON object
- \( \text{nmax} \) : Allocated size of string [bytes]

Outputs

- \( \text{sy1..8} \) : String output value
- \( \text{iE} \) : Error code

Licence: STANDARD
REGEXP – Regular expression parser

Block Symbol

Licence: ADVANCED
REPLACE — Replace substring

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( su1 )</td>
<td>String input value</td>
<td>string</td>
</tr>
<tr>
<td>( su2 )</td>
<td>String input value</td>
<td>string</td>
</tr>
<tr>
<td>( l )</td>
<td>Length of origin text</td>
<td>long</td>
</tr>
<tr>
<td>( p )</td>
<td>Position of origin text (one-based)</td>
<td>long</td>
</tr>
</tbody>
</table>

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( nmax )</td>
<td>Allocated size of string [bytes]</td>
<td>( \downarrow 0 \uparrow 65520 ) long</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( sy )</td>
<td>String output value</td>
<td>string</td>
</tr>
</tbody>
</table>
RTOS – Real Number to String Conversion

Block Symbol

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**

| **u** | Analog input of the block | double |

**Parameters**

| **prec** | Precision (number of digits) | ↓0 ↑20 | long |
| **mode** | Output string format | ⊙1 | long |
| 1 | best fit | |
| 2 | normal | |
| 3 | exponencial | |

**Output**

| **su** | String output value | string |
**SELSOCT** – *String selector*

**Function Description**

The SELSOCT block selects one of the input strings and copies it to the output string sy. The selection of the active signal u0...u15 is based on the iSW input or the binary inputs SW1...SW3. These two modes are distinguished by the BINF binary flag. The signal is selected according to the following table:

<table>
<thead>
<tr>
<th>iSW</th>
<th>SW1</th>
<th>SW2</th>
<th>SW3</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>u0</td>
</tr>
<tr>
<td>1</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>u1</td>
</tr>
<tr>
<td>2</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>u2</td>
</tr>
<tr>
<td>3</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>u3</td>
</tr>
<tr>
<td>4</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>u4</td>
</tr>
<tr>
<td>5</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>u5</td>
</tr>
<tr>
<td>6</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>u6</td>
</tr>
<tr>
<td>7</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>u7</td>
</tr>
</tbody>
</table>

**Inputs**

- **su0...7**  String input value  *string*
- **iSW**  Active signal selector  *long*
- **SW1...3**  Binary signal selector  *bool*

**Parameters**

- **BINF**  Enable the binary selectors  *bool*
- **nmax**  Allocated size of string [bytes]  0 65520  *long*
Output

$s_y$  The selected input signal

$\text{string}$
**STOR – String to real number conversion**

**Block Symbol**

```
 STOR
```

**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**

- **su**
  
  String input value

**Parameter**

- **yerr**
  
  Substitute value for an error case

**Outputs**

- **y**
  
  Analog output of the block

- **E**
  
  Error indicator
Chapter 12

PARAM – Blocks for parameter handling

Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETPA</td>
<td>Block for remote array parameter acquirement</td>
<td>302</td>
</tr>
<tr>
<td>GETPR, GETPI, GETPB</td>
<td>Blocks for remote parameter acquirement</td>
<td>304</td>
</tr>
<tr>
<td>GETPS</td>
<td>* Block for remote string parameter acquirement</td>
<td>306</td>
</tr>
<tr>
<td>PARA</td>
<td>Block with input-defined array parameter</td>
<td>307</td>
</tr>
<tr>
<td>PARR, PARI, PARB</td>
<td>Blocks with input-defined parameter</td>
<td>308</td>
</tr>
<tr>
<td>PARS</td>
<td>* Block with input-defined string parameter</td>
<td>310</td>
</tr>
<tr>
<td>SETPA</td>
<td>Block for remote array parameter setting</td>
<td>311</td>
</tr>
<tr>
<td>SETPR, SETPI, SETPB</td>
<td>Blocks for remote parameter setting</td>
<td>313</td>
</tr>
<tr>
<td>SETPS</td>
<td>* Block for remote string parameter setting</td>
<td>315</td>
</tr>
<tr>
<td>SGSLP</td>
<td>Set, get, save and load parameters</td>
<td>316</td>
</tr>
<tr>
<td>SILO</td>
<td>Save input value, load output value</td>
<td>320</td>
</tr>
</tbody>
</table>
GETPA – Block for remote array parameter acquirement

Block Symbol

<table>
<thead>
<tr>
<th>GET</th>
<th>arrRef</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GETPA</td>
</tr>
</tbody>
</table>

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 RexView program.

Input

- GET: Input for initiating one-shot parameter read

Outputs

- arrRef: Array reference
- E: Error flag
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, the remote parameter is read only when
           forced to by the GET input (rising edge)

nmax  Maximum size of array  ⊗256  long
GETPR, GETPI, GETPB – Blocks for remote parameter acquirement

Block Symbols

Function Description

The GETPR, GETPI and GETPB blocks are used for acquiring the parameters of other blocks in the model remotely. The only difference among the three blocks is the type of parameter which they are acquiring. The GETPR block is used for obtaining real parameters, the GETPI block for integer parameters and the GETPB block for Boolean parameters.

The blocks operate in two modes, which are switched by the GETF parameter. For GETF = off the output y (or k, Y) is set to the value of the remote parameter at the start and every time when the remote parameter changes. If the GETF parameter is set to on, then the blocks work in single-shot read mode. In that case the remote parameter is read only when rising edge (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 RexView program.

Input

GET Input for initiating one-shot parameter read (off→on) bool
Outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Parameter value, output of the GETPR block</td>
<td>double</td>
</tr>
<tr>
<td>k</td>
<td>Parameter value, output of the GETPI block</td>
<td>long</td>
</tr>
<tr>
<td>Y</td>
<td>Parameter value, output of the GETPB block</td>
<td>bool</td>
</tr>
<tr>
<td>E</td>
<td>Error flag</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... No error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... An error occurred</td>
<td></td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc</td>
<td>String connection to the remote parameter respecting the above mentioned notation</td>
<td>string</td>
</tr>
<tr>
<td>GETF</td>
<td>Continuous or one-shot mode</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... Remote parameter is continuously read</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... One-shot mode, the remote parameter is read only when forced to by the GET input (rising edge)</td>
<td></td>
</tr>
</tbody>
</table>
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

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 \( \text{apar} \).

The Boolean input \( \text{LOC} \) (LOCal) determines whether the value of the \( \text{apar} \) parameter is read from the input \( uRef \) or is input-independent (\( \text{LOC} = \text{on} \)). In the local mode \( \text{LOC} = \text{on} \) the parameter \( \text{apar} \) contains the last value of input \( uRef \) entering the block right before \( \text{LOC} \) was set to \( \text{on} \).

The output value is equivalent the value of the parameter (\( yRef = \text{apar} \)).

Inputs

- \( uRef \), Array reference
- \( \text{LOC} \), Activation of local mode
  - \( \text{off} \) ... The parameter follows the input
  - \( \text{on} \) ... Local mode active

Output

- \( yRef \), Array reference

Parameters

- \( \text{SETS} \), Set array size flag. Use this flag to adjust the size of array when setting the parameter.
  - \( \text{bool} \)
- \( \text{apar} \), Initial value of the parameter
  - \( [0.0\ 1.0\ 2.0\ 3.0\ 4.0\ 5.0] \) \( \text{double} \)
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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Parameter value (the PARR block)</td>
</tr>
<tr>
<td>ip</td>
<td>Parameter value (the PARI block)</td>
</tr>
<tr>
<td>P</td>
<td>Parameter value (the PARB block)</td>
</tr>
<tr>
<td>LOC</td>
<td>Activation of local mode</td>
</tr>
<tr>
<td></td>
<td>off ... The parameter follows the input</td>
</tr>
<tr>
<td></td>
<td>on .... Local mode active</td>
</tr>
</tbody>
</table>

Output

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Logical output of the PARR block</td>
</tr>
<tr>
<td>k</td>
<td>Logical output of the PARI block</td>
</tr>
<tr>
<td>Y</td>
<td>Logical output of the PARB block</td>
</tr>
</tbody>
</table>

Parameter

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>par</td>
<td>Initial value of the parameter (the PARR block)</td>
</tr>
<tr>
<td>ipar</td>
<td>Initial value of the parameter (the PARI block)</td>
</tr>
<tr>
<td>PAR</td>
<td>Initial value of the parameter (the PARB block)</td>
</tr>
</tbody>
</table>
SATF  Activation of the saturation limits for the PARR and PARI blocks bool
    off ... Signal not limited
    on ..... Saturation limits active

hilim  Upper limit of the output signal (the PARR and PARI blocks) ⊙1.0 double
lolim  Lower limit of the output signal (the PARR and PARI blocks) ⊙-1.0 double
**PARS – * Block with input-defined string parameter**

Block Symbol

![Block Symbol](image)

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 **RexView** program.

**Inputs**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrRef</td>
<td>Array reference</td>
<td>reference</td>
</tr>
<tr>
<td>SET</td>
<td>Input for initiating one-shot parameter write</td>
<td>bool</td>
</tr>
</tbody>
</table>

**Output**

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Error flag</td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc</td>
<td>String connection to the parameter</td>
<td>string</td>
</tr>
<tr>
<td>SETF</td>
<td>Continuous or one-shot mode</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off . . . Remote parameter is continuously updated</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on . . . One-shot mode, the remote parameter is updated only</td>
<td></td>
</tr>
<tr>
<td></td>
<td>when forced to by the SET input (rising edge)</td>
<td></td>
</tr>
<tr>
<td>SETS</td>
<td>Set array size flag. Use this flag to adjust the size of array when setting</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>the parameter.</td>
<td></td>
</tr>
</tbody>
</table>
**SETPR, SETPI, SETPB – Blocks for remote parameter setting**

**Block Symbols**

![Block Symbols](image)

**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 RexView program.
CHAPTER 12. PARAM – BLOCKS FOR PARAMETER HANDLING

Inputs

- **p**: Desired parameter value at the SETPR block input  
  - double
- **ip**: Desired parameter value at the SETPI block input  
  - long
- **P**: Desired parameter value at the SETPB block input  
  - bool
- **SET**: Input for initiating one-shot parameter write (off→on)  
  - bool

Outputs

- **y**: Parameter value (the SETPR block)  
  - double
- **k**: Parameter value (the SETPI block)  
  - long
- **Y**: Parameter value (the SETPB block)  
  - bool
- **E**: Error flag
  - off ... No error
  - on .... An error occurred

Parameters

- **sc**: String connection to the remote parameter respecting the above mentioned notation  
  - string
- **SETF**: Continuous or one-shot mode
  - off ... Remote parameter is continuously updated
  - on .... One-shot mode, the remote parameter is updated only when forced to by the SET input (rising edge)  
  - bool
SETPS — * Block for remote string parameter setting

Block Symbol

License: STANDARD

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

Inputs

| sp | Desired parameter value | string |
| SET | Input for initiating one-shot parameter write | bool |

Parameters

| sc | String connection to the parameter | string |
| SETF | Set parameter only when forced to | bool |
| nmax | Allocated size of string | long |

Outputs

| sy | Parameter value | string |
| E | Error indicator | bool |
SGSLP – Set, get, save and load parameters

Block Symbol

Function Description

The SGSLP block is a special function block for manipulation with parameters of other function blocks in the REX control 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 REX control system. An empty sci string means that no parameter is specified, otherwise one of the following syntaxes is used:

1. <block>:<param> – Specifies one function block named block and its parameter param. The same block and parameter are used for all nps parameter sets in this case.

2. <block>:<param><sep>...<block>:<param> – This syntax allows the parameters to differ among the parameter sets. In general, each sci string can contain up to 16 items in the form <block>:<param> separated by comma or semi-colon. E.g. the third item of these is active for ips = 2. There should be exactly nps items in each non-empty sci string. If there is less items than nps none of the below described operations can be executed on the incomplete parameter set.
It is recommended not to use both syntaxes in one SGSLP block, all 16 sci strings should have the same form. The first syntax is for example used when producing nps types of goods, where many parameters must be changed for each type of production. The second syntax is usually used for saving user-defined parameters to disk (see the SAVE operation below). In that case it is desirable to arrange automated switching of the ips input (e.g. using the ATMT block from the LOGIC library).

The broot parameter is suitable when all blocks whose parameters are to be controlled by the SGSLP block reside in the same subsystem or deeper in the hierarchy. It is inserted in front of each <block> substring in the sci parameters. The ‘.’ character stands for the subsystem where the SGSLP block is located. No quotation marks are used to define the parameter, they are used here solely to highlight a single character. If the broot parameter is an empty string, all <block> items must contain full path. For example, to create a connection to the CNR block and its parameter ycn located in the same subsystem as the SGSLP block, broot = . and sc0 = CNR:ycn must be set. Or it is possible to leave the broot parameter empty and put the ‘.’ character to the sc0 string. See the GETPR or SETPR blocks description for more details about full paths in the REX control system.

The SGSLP block executes one of the below described operations when a rising edge (off→on) occurs at the input of the same name. The operations are:

- **SET** – Sets the parameters of the corresponding parameter set ips to the values of the input signals ui. In case the parameter is successfully set, the same value is also sent to the yi output.

- **GET** – Gets the parameters of the corresponding parameter set ips. In case the parameter is successfully read, its value is sent to the yi output.

- **SAVE** – Saves the parameters of the corresponding parameter set ips to a file on the target platform. The parameters of the procedure and the format of the resulting file are described below.

- **LOAD** – Loads the parameters of the corresponding parameter set ips from a file on the target platform. This operation is executed also during the initialization of the block but only when 0 ≤ ips0 ≤ 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 REX Control 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 \( s_i, i = 0, \ldots, m \), where \( m < 16 \) defines the nonempty \( scm \) string with the highest number. The lines have the form:

\[
"<block>:<param>" , \ldots , "<block>:<param>"
\]

\[
<value>, \ldots , <value>
\]

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 \( s_i \) 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 \( s_i \) parameters syntax without modifying the file.

Consider using the SILO block if working with only a few values.

**Inputs**

- **ui**  
  - \( i \)-th analog input signal, \( i = 0, \ldots, 15 \)  
  - double

- **ips**  
  - Parameter set index (numbered from zero)  
  - long

- **SET**  
  - Set the parameters of the \( ips \) parameter set according to the values of the \( ui \) inputs. The values can be found at the \( yi \) outputs after a successful operation.  
  - bool

- **GET**  
  - Get the parameters of the \( ips \) parameter set. The values can be found at the \( yi \) outputs after a successful operation.  
  - bool

- **SAVE**  
  - Save the \( ips \) parameter set to a file on the target device  
  - bool

- **LOAD**  
  - Load the \( ips \) parameter set from a file on the target device  
  - bool

**Outputs**

- **yi**  
  - \( i \)-th analog output signal, \( i = 0, \ldots, 15 \)  
  - double

- **E**  
  - Error flag
    - off ... No error
    - on .... An error occurred (see iE)  
  - bool
Error or warning code of the last operation

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 ips 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 ips parameter is out of range

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>nps</td>
<td>Number of parameter sets</td>
<td>long</td>
</tr>
<tr>
<td>ips0</td>
<td>Index of parameter set to load and set during the block initialization. No set is read for ips0 &lt; 0 or ips0 ≥ nps</td>
<td>long</td>
</tr>
<tr>
<td>iprec</td>
<td>Precision (number of digits) for storing the values of double type in a file</td>
<td>long</td>
</tr>
<tr>
<td>icolw</td>
<td>Requested column width in the status file. Spaces are appended to the parameter value when necessary.</td>
<td>long</td>
</tr>
<tr>
<td>fname</td>
<td>Name of the file the SAVE and LOAD operations work with</td>
<td>string</td>
</tr>
<tr>
<td>broot</td>
<td>Root block in hierarchy, inserted at the beginning of all sci parameters, see the description above</td>
<td>string</td>
</tr>
<tr>
<td>sci</td>
<td>Strings defining the connection of ui inputs and yi outputs to the parameters, i = 0, . . . , 15, see details above</td>
<td>string</td>
</tr>
</tbody>
</table>
SILO – Save input value, load output value

Block Symbol

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. If an error occurs, 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 folder where the archives of the REX Control System are stored.

Use the SGSLP function block for advanced and complex operations.

Inputs

| u | Input signal | double |
| SAVE | Save value to file | bool |
| LOAD | Load value from file | bool |

Parameters

| fname | Name of persistent storage file | string |
| CSF | Flag for continuous saving | bool |
| CLF | Flag for continuous loading | bool |
| yerr | Substitute value for an error case | double |

Outputs

| y | Output signal | double |
| E | Error flag | bool |
| iE | Error code of the operating system | long |
Chapter 13

MODEL – Dynamic systems simulation

Contents

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDELSSM</td>
<td>Continuous state space model of a linear system with time delay</td>
<td>322</td>
</tr>
<tr>
<td>CSSM</td>
<td>Continuous state space model of a linear system</td>
<td>325</td>
</tr>
<tr>
<td>DDELSSM</td>
<td>Discrete state space model of a linear system with time delay</td>
<td>327</td>
</tr>
<tr>
<td>DSSM</td>
<td>Discrete state space model of a linear system</td>
<td>329</td>
</tr>
<tr>
<td>FOPDT</td>
<td>First order plus dead-time model</td>
<td>331</td>
</tr>
<tr>
<td>MDL</td>
<td>Process model</td>
<td>332</td>
</tr>
<tr>
<td>MDLI</td>
<td>Process model with input-defined parameters</td>
<td>333</td>
</tr>
<tr>
<td>MVD</td>
<td>Motorized valve drive</td>
<td>334</td>
</tr>
<tr>
<td>SOPDT</td>
<td>Second order plus dead-time model</td>
<td>335</td>
</tr>
</tbody>
</table>
CDELSSM – Continuous state space model of a linear system with time delay

Block Symbol

Function Description

The CDELSSM block (Continuous State Space Model with time DELay) simulates behavior of a linear system with time delay \( del \)

\[
\begin{align*}
\frac{dx(t)}{dt} &= A_c x(t) + B_c u(t - del), \quad x(0) = x_0 \\
y(t) &= C_c x(t) + D_c u(t),
\end{align*}
\]

where \( x(t) \in \mathbb{R}^n \) is the state vector, \( x_0 \in \mathbb{R}^n \) is the initial value of the state vector, \( u(t) \in \mathbb{R}^m \) is the input vector, \( y(t) \in \mathbb{R}^p \) is the output vector. The matrix \( A_c \in \mathbb{R}^{n \times n} \) is the system dynamics matrix, \( B_c \in \mathbb{R}^{n \times m} \) is the input matrix, \( C_c \in \mathbb{R}^{p \times n} \) is the output matrix and \( D_c \in \mathbb{R}^{p \times m} \) is the direct transmission (feedthrough) matrix.

All matrices are specified in the same format as in Matlab, i.e. the whole matrix is placed in brackets, elements are entered by rows, elements of a row are separated by spaces (blanks), rows are separated by semicolons. The \( x_0 \) vector is a column, therefore the elements are separated by semicolons (each element is in a separate row).

The simulated system is first converted to the discrete (discretized) state space model

\[
\begin{align*}
x((k+1)T) &= A_d x(kT) + B_{d1} u((k-d)T) + B_{d2} u((k-d+1)T), \quad x(0) = x_0 \\
y(kT) &= C_c x(kT) + D_c u(kT),
\end{align*}
\]

where \( k \in \{1, 2, \ldots\} \) is the simulation step, \( T \) is the execution period of the block in seconds and \( d \) is a delay in simulation step such that \( (d-1)T < del \leq dT \). The period \( T \)
is not entered in the block, it is determined automatically as a period of the task (\textsc{Task}, \textsc{QTask} nebo \textsc{IOTask}) containing the block.

If the input $u(t)$ is changed only in the moments of sampling and between two consecutive sampling instants is constant, i.e. $u(t) = u(kT)$ for $t \in [kT, (k + 1)T)$, then the matrices $A_d$, $B_{d1}$ and $B_{d2}$ are determined by

\begin{align*}
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{align*}

where $\Delta = del - (d - 1)T$.

Computation of discrete matrices $A_d$, $B_{d1}$ and $B_{d2}$ is based on a method described in [6], 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**

\begin{itemize}
  \item \texttt{R1} Reset signal. When \texttt{R1} = \texttt{on}, the state vector $x$ is set to its initial value $x_0$. The simulation continues on the falling edge of \texttt{R1} (on→off).
  \item \texttt{HLD} Simulation output holds its value if \texttt{HLD} = \texttt{on}.
  \item \texttt{u1..u16} Simulated system inputs. First $m$ simulation inputs are used where $m$ is the number of columns of the matrix $B_c$.
\end{itemize}

**Outputs**

\begin{itemize}
  \item \texttt{iE} Block error code
    \begin{itemize}
      \item 0 . . . O.K., the simulation runs correctly
      \item -213 . . incompatibility of the state space model matrices dimensions
      \item -510 . . the model is badly conditioned (some working matrix is singular or nearly singular)
      \item xxx . . error code xxx of \textsc{REX}, see appendix B for details
    \end{itemize}
  \item \texttt{y1..y16} Simulated system outputs. First $p$ simulation outputs are used where $p$ is the number of rows of the matrix $C_c$.
\end{itemize}

**Parameters**

\begin{itemize}
  \item \texttt{UD} Matrix $D_c$ usage flag. If \texttt{UD} = \texttt{off} then the $D_c$ matrix is not used for simulation (simulation behaves as if the $D_c$ matrix is zero).
  \item \texttt{del} Model time delay [s].
  \item \texttt{is} Order of the Padé approximation of the matrix exponential for the computation of the discretized system matrices.
\end{itemize}
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>eps</code></td>
<td>Required accuracy of the Padé approximation. ( \downarrow 0.0 \uparrow 1.0 \odot 0.0 )</td>
<td>double</td>
</tr>
<tr>
<td><code>Ac</code></td>
<td>Matrix ((n \times n)) of the continuous linear system dynamics.</td>
<td>double</td>
</tr>
<tr>
<td><code>Bc</code></td>
<td>Input matrix ((n \times m)) of the continuous linear system.</td>
<td>double</td>
</tr>
<tr>
<td><code>Cc</code></td>
<td>Output matrix ((p \times n)) of the continuous linear system.</td>
<td>double</td>
</tr>
<tr>
<td><code>Dc</code></td>
<td>Direct transmission (feedthrough) matrix ((p \times m)) of the continuous</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>linear system. The matrix is used only if the parameter ( \texttt{UD=on} ).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If ( \texttt{UD=off} ), the dimensions of the ( Dc ) matrix are not</td>
<td></td>
</tr>
<tr>
<td></td>
<td>checked.</td>
<td></td>
</tr>
<tr>
<td><code>x0</code></td>
<td>Initial value of the state vector ((\text{of dimension } n)) of the</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>continuous linear system.</td>
<td></td>
</tr>
</tbody>
</table>
CSSM – Continuous state space model of a linear system

Block Symbol

Function Description

The CSSM block (Continuous State Space Model) simulates behavior of a linear system

\[
\frac{dx(t)}{dt} = A_c x(t) + B_c u(t), \quad x(0) = x_0
\]
\[
y(t) = C_c x(t) + D_c u(t),
\]

where \(x(t) \in \mathbb{R}^n\) is the state vector, \(x_0 \in \mathbb{R}^n\) is the initial value of the state vector, \(u(t) \in \mathbb{R}^m\) is the input vector, \(y(t) \in \mathbb{R}^p\) is the output vector. The matrix \(A_c \in \mathbb{R}^{n \times n}\) is the system dynamics matrix, \(B_c \in \mathbb{R}^{n \times m}\) is the input matrix, \(C_c \in \mathbb{R}^{p \times n}\) is the output matrix and \(D_c \in \mathbb{R}^{p \times m}\) is the direct transmission (feedthrough) matrix.

All matrices are specified in the same format as in Matlab, i.e. the whole matrix is placed in brackets, elements are entered by rows, elements of a row are separated by spaces (blanks), rows are separated by semicolons. The \(x_0\) vector is a column, therefore the elements are separated by semicolons (each element is in a separate row).

The simulated system is first converted to the discrete (discretized) state space model

\[
x((k + 1)T) = A_d x(kT) + B_d u(kT), \quad x(0) = x_0
\]
\[
y(kT) = C_d x(kT) + D_d u(kT),
\]

where \(k \in \{1, 2, \ldots\}\) is the simulation step, \(T\) is the execution period of the block in seconds. The period \(T\) is not entered in the block, it is determined automatically as a period of the task (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

$$A_d = e^{AcT}$$
$$B_d = \int_0^T e^{Ac\tau} B_c d\tau$$

Computation of discrete matrices $A_d$ and $B_d$ is based on a method described in [6], which uses Padé approximations of matrix exponential and its integral and scaling technique.

During the real-time simulation, single simulation step of the above discrete state space model is computed in each execution time instant.

**Inputs**

- **R1**: Reset signal. When $R1 = \text{on}$, the state vector $x$ is set to its initial value $x_0$. The simulation continues on the falling edge of $R1$ ($\text{on} \rightarrow \text{off}$).
- **HLD**: Simulation output holds its value if $\text{HLD} = \text{on}$. bool
- **u1..u16**: Simulated system inputs. First $m$ simulation inputs are used where $m$ is the number of columns of the matrix $B_c$. double

**Outputs**

- **iE**: 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 REX, see appendix B for details

- **y1..y16**: Simulated system outputs. First $p$ simulation outputs are used where $p$ is the number of rows of the matrix $C_c$. double

**Parameters**

- **UD**: Matrix $D_c$ usage flag. If $\text{UD} = \text{off}$ then the $D_c$ matrix is not used for simulation (simulation behaves as if the $D_c$ matrix is zero). bool
- **is**: Order of the Padé approximation of the matrix exponential for the computation of the discretized system matrices. $\downarrow 0 \uparrow 4 \odot 2$ long
- **eps**: Required accuracy of the Padé approximation. $\downarrow 0.0 \uparrow 1.0 \odot 0.0$ double
- **Ac**: Matrix $(n \times n)$ of the continuous linear system dynamics. double
- **Bc**: Input matrix $(n \times m)$ of the continuous linear system. double
- **Cc**: Output matrix $(p \times n)$ of the continuous linear system. double
- **Dc**: Direct transmission (feedthrough) matrix $(p \times m)$ of the continuous linear system. The matrix is used only if the parameter $\text{UD} = \text{on}$. If $\text{UD} = \text{off}$, the dimensions of the $D_c$ matrix are not checked. double
- **x0**: Initial value of the state vector (of dimension $n$) of the continuous linear system. double
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 $\text{del}$

$$
\begin{align*}
  x(k + 1) &= Ax(k) + Bu(k - d), \quad x(0) = x_0 \\
  y(k) &= Cx(k) + Du(k),
\end{align*}
$$

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 $A \in \mathbb{R}^{n \times n}$ is the system dynamics matrix, $B \in \mathbb{R}^{n \times m}$ is the input matrix, $C \in \mathbb{R}^{p \times n}$ is the output matrix and $D \in \mathbb{R}^{p \times m}$ is the direct transmission (feedthrough) matrix. Number of steps of the delay $d$ is the largest integer such that $d.T \leq \text{del}$, 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 $x$ is set to its initial value $x_0$. The simulation continues on the falling edge of R1 (on→off).
CHAPTER 13. MODEL – DYNAMIC SYSTEMS SIMULATION

HLD Simulation output holds its value if HLD=on.

\[ u_1 \ldots u_{16} \] Simulated system inputs. First \( m \) simulation inputs are used where \( m \) is the number of columns of the matrix \( B_d \).

**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 REX, see appendix B for details

\[ y_1 \ldots y_{16} \] Simulated system outputs. First \( p \) simulation outputs are used where \( p \) is the number of rows of the matrix \( C_d \).

**Parameters**

\[ UD \] Matrix \( D_d \) usage flag. If \( UD=off \) then the \( D_d \) matrix is not used for simulation (simulation behaves as if the \( D_d \) matrix is zero).

\[ del \] Model time delay [s].

\[ Ad \] Matrix \((n \times n)\) of the discrete linear system dynamics.

\[ Bd \] Input matrix \((n \times m)\) of the discrete linear system.

\[ Cd \] Output matrix \((p \times n)\) of the discrete linear system.

\[ Dd \] 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 \( D_d \) matrix are not checked.

\[ x_0 \] Initial value of the state vector (of dimension \( n \)) of the discrete linear system.
**DSSM – Discrete state space model of a linear system**

**Block Symbol**

---

**Function Description**

The **DSSM** block (Discrete State Space Model) simulates behavior of a linear system

\[
\begin{align*}
    x(k+1) &= A_d x(k) + B_d u(k), \quad x(0) = x_0 \\
    y(k) &= C_d x(k) + D_d u(k),
\end{align*}
\]

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 \(x\) is set to its initial value \(x_0\). The simulation continues on the falling edge of **R1** (**on→off**).
  - **bool**

- **HLD**
  - Simulation output holds its value if **HLD**=**on**.
  - **bool**

- **u1..u16**
  - Simulated system inputs. First \(m\) simulation inputs are used where \(m\) is the number of columns of the matrix \(B_d\).
  - **double**
Outsuts

\[ iE \]
Block error code

- 0 . . . . . O.K., the simulation runs correctly
- 213 . . . incompatibility of the state space model matrices dimensions
- xxx . . . error code xxx of REX, see appendix B for details

\[ y1..y16 \]
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
**FOPDT – First order plus dead-time model**

Block Symbol

![Block Symbol](image)

**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:

\[ P(s) = \frac{k_0}{(\tau s + 1)} \cdot e^{-\delta t s} \]

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

**Input**

- **u** Analog input of the block 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 (number of samples) of delay buffer (used for internal memory allocation) ↓1 ↑10000000 ⊗10 long
MDL – Process model

Block Symbol License: STANDARD

Function Description

The MDL block is a discrete simulator of continuous-time system with transfer function

\[ F(s) = \frac{K_0 e^{-D_s}}{(\tau_1 s + 1)(\tau_2 s + 1)} \]

where \( K_0 > 0 \) is the static gain \( k_0 \), \( D \geq 0 \) is the time-delay \( \delta_1 \) and \( \tau_1, \tau_2 > 0 \) are the system time-constants \( \tau_1 \) and \( \tau_2 \).

Input

\( u \) Analog input of the block double

Output

\( y \) Analog output of the block double

Parameters

\( k_0 \) Static gain \( \circ 1.0 \) double
\( \delta_1 \) Dead time [s] double
\( \tau_1 \) The first time constant \( \circ 1.0 \) double
\( \tau_2 \) The second time constant \( \circ 2.0 \) double
\( n_{max} \) Size (number of samples) of delay buffer (used for internal memory allocation) \( \downarrow 1 \uparrow 1000000 \circ 10 \) long
MDLI – Process model with input-defined parameters

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 \( k_0 \), \( D \geq 0 \) is the time-delay \( \text{del} \) and \( \tau_1, \tau_2 > 0 \) are the system time-constants \( \text{tau1} \) and \( \text{tau2} \). In contrary to the MDL block the system is time variant. The system parameters are determined by the input signals.

Inputs

- \( u \) Analog input of the block
- \( k_0 \) Static gain
- \( \text{del} \) Dead time [s]
- \( \text{tau1} \) The first time constant
- \( \text{tau2} \) The second time constant

Output

- \( y \) Analog output of the block

Parameters

- \( \text{nmax} \) Size (number of samples) of delay buffer (used for internal memory allocation)
**MVD – Motorized valve drive**

**Block Symbol**

```
  UP
 y
 HS
 DN LS
 MVD
```

**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 signalized 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
- **DN**: Close

**Outputs**

- **y**: Valve position
- **HS**: Upper end switch
- **LS**: Lower end switch

**Parameters**

- **y0**: Initial valve position
- **tv**: Time required for transition between $y = 0$ and $y = 1$ [s] $\circ 10.0$ double
- **hilim**: Upper limit position (open) $\circ 1.0$ double
- **lolim**: Lower limit position (closed) double
SOPDT – Second order plus dead-time model

Block Symbol

Function Description

The SOPDT block is a discrete simulator of a second order continuous-time system with time delay, which can be described by one of the transfer functions below. The type of the model is selected by the \( \text{itf} \) parameter.

\[
\begin{align*}
\text{itf} = 1 : & \quad P(s) = \frac{pb_1 \cdot s + pb_0}{s^2 + pa_1 \cdot s + pa_0} \cdot e^{-\text{del} \cdot s} \\
\text{itf} = 2 : & \quad P(s) = \frac{k_0 (\tau \cdot s + 1)}{(\tau_1 \cdot s + 1)(\tau_2 \cdot s + 1)} \cdot e^{-\text{del} \cdot s} \\
\text{itf} = 3 : & \quad P(s) = \frac{k_0 \cdot \omega_m^2 \cdot (\tau \cdot s + 1)}{(s^2 + 2 \cdot \xi \cdot \omega_m \cdot s + \omega_m^2)} \cdot e^{-\text{del} \cdot s} \\
\text{itf} = 4 : & \quad P(s) = \frac{k_0 (\tau \cdot s + 1)}{(\tau_1 \cdot s + 1) \cdot s} \cdot e^{-\text{del} \cdot s}
\end{align*}
\]

For simulation of first order plus dead time systems (FOPDT) use the LLC block with parameter \( \text{a} \) set to zero.

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

Input

\( u \) Analog input of the block \quad \text{double}

Output

\( y \) Analog output of the block \quad \text{double}
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>itf</td>
<td>Transfer function form</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>A general form</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>A form with real poles</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>A form with complex poles</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>A form with integrator</td>
</tr>
<tr>
<td>k0</td>
<td>Static gain</td>
<td>double</td>
</tr>
<tr>
<td>tau</td>
<td>Numerator time constant</td>
<td>double</td>
</tr>
<tr>
<td>tau1</td>
<td>The first time constant</td>
<td>double</td>
</tr>
<tr>
<td>tau2</td>
<td>The second time constant</td>
<td>double</td>
</tr>
<tr>
<td>om</td>
<td>Natural frequency</td>
<td>double</td>
</tr>
<tr>
<td>xi</td>
<td>Relative damping coefficient</td>
<td>double</td>
</tr>
<tr>
<td>pb0</td>
<td>Numerator coefficient: $s^0$</td>
<td>double</td>
</tr>
<tr>
<td>pb1</td>
<td>Numerator coefficient: $s^1$</td>
<td>double</td>
</tr>
<tr>
<td>pa0</td>
<td>Denominator coefficient: $s^0$</td>
<td>double</td>
</tr>
<tr>
<td>pa1</td>
<td>Denominator coefficient: $s^1$</td>
<td>double</td>
</tr>
<tr>
<td>del</td>
<td>Dead time $[\text{s}]$</td>
<td>double</td>
</tr>
<tr>
<td>nmax</td>
<td>Size (number of samples) of delay buffer (used for internal memory allocation)</td>
<td>long</td>
</tr>
</tbody>
</table>
Chapter 14

MATRIX – Blocks for matrix and vector operations

Contents

<table>
<thead>
<tr>
<th>Block Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNA – * Array (vector/matrix) constant</td>
<td>338</td>
</tr>
<tr>
<td>RTOV – Vector multiplexer</td>
<td>339</td>
</tr>
<tr>
<td>SWVMR – Vector/matrix/reference signal switch</td>
<td>340</td>
</tr>
<tr>
<td>VTOR – Vector demultiplexer</td>
<td>341</td>
</tr>
</tbody>
</table>
CNA – * Array (vector/matrix) constant

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.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>CSV data file</td>
<td>string</td>
</tr>
<tr>
<td>TRN</td>
<td>Transpose loaded matrix</td>
<td>bool</td>
</tr>
<tr>
<td>nmax</td>
<td>Allocated size of array</td>
<td>num</td>
</tr>
<tr>
<td>etype</td>
<td>Type of elements</td>
<td>enum</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Bool</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Byte</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Short</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Long</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Word</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>DWord</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Float</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>Double</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>....</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Large</td>
</tr>
<tr>
<td>acn</td>
<td>Initial array value</td>
<td>double</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>vec</td>
<td>Reference to vector/matrix data</td>
<td>reference</td>
</tr>
</tbody>
</table>
**RTOV – Vector multiplexer**

**Block Symbol**

![Block Symbol]

**Function Description**

The RTOV block can be used to create vector signals in the REX Control 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 \( u_1 \) in the resulting signal. Only the first \( N \) input signals are combined into the resulting \( yVec \) vector signal.

**Inputs**

- **uVec** Vector signal
- \( u_1 \) Analog input of the block
- \( u_2 \) Analog input of the block
- \( u_3 \) Analog input of the block
- \( u_4 \) Analog input of the block
- \( u_5 \) Analog input of the block
- \( u_6 \) Analog input of the block
- \( u_7 \) Analog input of the block
- \( u_8 \) Analog input of the block

**Parameters**

- **nmax** Allocated size of vector
- **offset** Index of the first input in vector
- **n** Number of valid inputs

**Output**

- **yVec** Vector signal
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

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uRef0</td>
<td>Vector signal</td>
</tr>
<tr>
<td>uRef1</td>
<td>Vector signal</td>
</tr>
<tr>
<td>uRef2</td>
<td>Vector signal</td>
</tr>
<tr>
<td>uRef3</td>
<td>Vector signal</td>
</tr>
<tr>
<td>uRef4</td>
<td>Vector signal</td>
</tr>
<tr>
<td>uRef5</td>
<td>Vector signal</td>
</tr>
<tr>
<td>uRef6</td>
<td>Vector signal</td>
</tr>
<tr>
<td>uRef7</td>
<td>Vector signal</td>
</tr>
<tr>
<td>iSW</td>
<td>Active signal selector</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Output</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yRef</td>
<td>Vector signal</td>
</tr>
</tbody>
</table>
VTOR – Vector demultiplexer

Block Symbol

Function Description

The VTOR block splits the input vector signal into individual signals. The user defines the starting item and the number of items to feed to the output signals using the offset and N parameter respectively.

Input

uVec Vector signal

Parameters

n Number of valid outputs
offset Index of the first output

Outputs

y1 Analog output of the block
y2 Analog output of the block
y3 Analog output of the block
y4 Analog output of the block
y5 Analog output of the block
y6 Analog output of the block
y7 Analog output of the block
y8 Analog output of the block

Licence: STANDARD
Chapter 15

SPEC – Special blocks

Contents

- EPC – External program call ........................................... 344
- HTTP – HTTP GET or POST request .............................. 347
- SMTP – Send email message via SMTP ............................ 349
- RDC – Remote data connection ...................................... 351
- REXLANG – User programmable block ......................... 356
EPC – External program call

Block Symbol

<table>
<thead>
<tr>
<th>EPC</th>
<th>Licence: ADVANCED</th>
</tr>
</thead>
<tbody>
<tr>
<td>uVec1</td>
<td></td>
</tr>
<tr>
<td>uVec2</td>
<td></td>
</tr>
<tr>
<td>uVec3</td>
<td></td>
</tr>
<tr>
<td>uVec4</td>
<td></td>
</tr>
<tr>
<td>uVec5</td>
<td></td>
</tr>
<tr>
<td>uVec6</td>
<td></td>
</tr>
<tr>
<td>uVec7</td>
<td></td>
</tr>
<tr>
<td>uVec8</td>
<td></td>
</tr>
<tr>
<td>EXEC</td>
<td></td>
</tr>
<tr>
<td>RESET</td>
<td></td>
</tr>
<tr>
<td>DSI</td>
<td></td>
</tr>
<tr>
<td>DSO</td>
<td></td>
</tr>
<tr>
<td>yVec1</td>
<td></td>
</tr>
<tr>
<td>yVec2</td>
<td></td>
</tr>
<tr>
<td>yVec3</td>
<td></td>
</tr>
<tr>
<td>yVec4</td>
<td></td>
</tr>
<tr>
<td>yVec5</td>
<td></td>
</tr>
<tr>
<td>yVec6</td>
<td></td>
</tr>
<tr>
<td>yVec7</td>
<td></td>
</tr>
<tr>
<td>yVec8</td>
<td></td>
</tr>
<tr>
<td>DONE</td>
<td></td>
</tr>
<tr>
<td>BUSY</td>
<td></td>
</tr>
<tr>
<td>ERR</td>
<td></td>
</tr>
<tr>
<td>errID</td>
<td></td>
</tr>
<tr>
<td>res</td>
<td></td>
</tr>
<tr>
<td>icnt</td>
<td></td>
</tr>
<tr>
<td>ocnt</td>
<td></td>
</tr>
</tbody>
</table>

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 REX Control 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 REX Control System can read the files in the same format.

The block works in two modes. In basic mode, the rising edge on the EXEC input triggers reading the data on inputs and storing them in the ifns file. The values of the i-th input vector uVec<i> are stored in the i-th file from the ifns list. In sampling mode, the data from the input vectors are stored in each period of the control algorithm. In both cases the values from one time instant form one line in the file.

Analogically, the data from output files are copied to the outputs of the block (one line from the i-th file in the ofns list to the i-th output vector yVec<i>).

The inputs working in the sampling mode are defined by the sl list (comma-separated numbers). The outputs work always in the sampling mode – the last values are kept when
the end of file is reached. The copying of data to input files can be blocked by the DSI input, the same holds for output data and the DSO input.

Use the RTDV block to combine individual signals into a vector one for the uVec input. The RTDV 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uVec1..uVec8</td>
<td>Input vector signal</td>
</tr>
<tr>
<td>EXEC</td>
<td>External program is called on rising edge</td>
</tr>
<tr>
<td>RESET</td>
<td>Block reset (deletes the input and output files and terminates the external program)</td>
</tr>
<tr>
<td>DSI</td>
<td>Disable inputs sampling</td>
</tr>
<tr>
<td>DSO</td>
<td>Disable outputs sampling</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>yVec1..yVec8</td>
<td>Output vector signal</td>
</tr>
<tr>
<td>DONE</td>
<td>External program finished</td>
</tr>
<tr>
<td>BUSY</td>
<td>External program running</td>
</tr>
<tr>
<td>ERR</td>
<td>Error flag</td>
</tr>
<tr>
<td>errID</td>
<td>Error code</td>
</tr>
<tr>
<td>res</td>
<td>External program return code</td>
</tr>
<tr>
<td>icnt</td>
<td>Current input sample</td>
</tr>
<tr>
<td>ocnt</td>
<td>Current output sample</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd</td>
<td>Operating system command to execute</td>
</tr>
<tr>
<td>ifns</td>
<td>Input filenames (separated by semicolon)</td>
</tr>
<tr>
<td>ofns</td>
<td>Output filenames (separated by semicolon)</td>
</tr>
<tr>
<td>sl</td>
<td>List of inputs working in the sampling mode. The format of the list is e.g. 1,3..5,8. Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.</td>
</tr>
<tr>
<td>ifm</td>
<td>Maximum number of input samples</td>
</tr>
<tr>
<td>format</td>
<td>Format of input and output files</td>
</tr>
<tr>
<td>nmax</td>
<td>Maximum output vectors length</td>
</tr>
</tbody>
</table>

**Notes**
• The called external program has the same priority as the calling task. This priority is high, in some cases higher than operating-system-kernel tasks. On Linux based systems, it is possible to lower the priority by using the chrt command:
  
  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 REX Control 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

Function Description

The HTTP block performs a single HTTP GET or POST request. Target address (URL) is defined by \texttt{url} parameter and \texttt{urldata} input. A final URL is formed in the way so that \texttt{urldata} input is simply added to \texttt{url} parameter.

HTTP request is started by the \texttt{TRG} parameter. Then the \texttt{BUSY} output is set until a request is finished, which is signaled by the \texttt{DONE} output. In case of an error, the \texttt{ERROR} output is set. The \texttt{errId} output carries last error identified by REX Control System error code. The \texttt{hterror} carries a HTTP status code. All data sent back by server to client is stored in the \texttt{data} output.

The block may be run in blocking or non-blocking mode which is specified by the \texttt{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 \texttt{timeout} parameter.

The block supports user authentication using basic HTTP authentication method. User name and password may be specified by \texttt{user} and \texttt{password} parameters. The block also supports secure HTTP (HTTPS). It is also possible to let the block verify server’s certificate by setting the \texttt{VERIFY} parameter. SSL certificate of a server or server’s trusted certificate authority must be stored in the \texttt{certificate} parameter in a PEM format. The block does not support any certificate storage.

Parameters \texttt{postmime} and \texttt{acceptmime} specify MIME encoding of data being sent to server or expected encoding of a HTTP response.

Parameters \texttt{nmax}, \texttt{postmax}, and \texttt{datamax} specify maximum sizes of buffers allocated by the block. The \texttt{nmax} parameter is maximal size of any string parameter. The \texttt{postmax} parameter specifies a maximal size of \texttt{postdata}. The \texttt{datamax} parameter specifies a
maximal size of data.

Inputs

**postdata** Data to put in HTTP POST request string

**urldata** Data to append to URL address string

**TRG** Trigger of the selected action bool

Parameters

**url** URL address to send the HTTP request to string

**method** HTTP request type long ◎1

1 .... GET
2 .... POST

**user** User name string

**password** Password string

**certificate** Authentication certificate string

**VERIFY** Enable server verification (valid certificate) bool

**postmime** MIME encoding for POST request ◊application/json string

**acceptmime** MIME encoding of HTTP response ◊application/json string

**timeout** Timeout interval ◊5.0 double

**BLOCKING** Wait for the operation to finish bool

**nmax** Allocated size of string ↓0 ↑65520 long

**postmax** Allocated memory for POST request data ↓128 ↑65520 ◊256 long

**datamax** Allocated memory for HTTP response ↓128 ↑10000000 ◊1024 long

Outputs

**data** Response data string

**BUSY** Sending HTTP request bool

**DONE** HTTP request processed bool

**ERROR** Error indicator bool

**errId** Error code error

**hterror** HTTP response long
SMTP – Send email message via SMTP

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 REX Control 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>subj</td>
<td>Subject of the e-mail message</td>
<td>string</td>
</tr>
<tr>
<td>body</td>
<td>Body of the e-mail message</td>
<td>string</td>
</tr>
<tr>
<td>TRG</td>
<td>Trigger of the selected action</td>
<td>bool</td>
</tr>
</tbody>
</table>

### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>SMTP server address</td>
<td>string</td>
</tr>
<tr>
<td>to</td>
<td>E-mail of the recipient</td>
<td>string</td>
</tr>
<tr>
<td>from</td>
<td>E-mail of the sender</td>
<td>string</td>
</tr>
<tr>
<td>tls</td>
<td>Encryption method</td>
<td>●1 long</td>
</tr>
<tr>
<td></td>
<td>1 . . . None</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 . . . StartTLS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 . . . TLS</td>
<td></td>
</tr>
<tr>
<td>user</td>
<td>User name</td>
<td>string</td>
</tr>
<tr>
<td>password</td>
<td>Password</td>
<td>string</td>
</tr>
<tr>
<td>domain</td>
<td>Domain name or identification of the target device</td>
<td>string</td>
</tr>
<tr>
<td>auth</td>
<td>Authentication method</td>
<td>●1 long</td>
</tr>
<tr>
<td></td>
<td>1 . . . Login</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 . . . Plain</td>
<td></td>
</tr>
<tr>
<td>certificate</td>
<td>Authentication certificate</td>
<td>string</td>
</tr>
<tr>
<td>VERIFY</td>
<td>Enable server verification (valid certificate)</td>
<td>bool</td>
</tr>
<tr>
<td>timeout</td>
<td>Timeout interval</td>
<td>double</td>
</tr>
<tr>
<td>BLOCKING</td>
<td>Wait for the operation to finish</td>
<td>bool</td>
</tr>
<tr>
<td>nmax</td>
<td>Allocated size of string</td>
<td>●0 &lt;65520 ●512 long</td>
</tr>
<tr>
<td>datamax</td>
<td>Allocated memory for HTTP response</td>
<td>●128 &lt;65520 long</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSY</td>
<td>Sending e-mail</td>
<td>bool</td>
</tr>
<tr>
<td>DONE</td>
<td>E-mail has been sent</td>
<td>bool</td>
</tr>
<tr>
<td>ERROR</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
<tr>
<td>errId</td>
<td>Error code</td>
<td>error</td>
</tr>
</tbody>
</table>
**RDC – Remote data connection**

**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 REX control 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 $\text{HLD} = \text{on}$, the block execution is terminated.

- If the $\text{period}$ 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 $\text{period}$ parameter. If the $\text{period}$ parameter is zero or negative, the time difference is not checked.

- A data packet is created. The packet includes block number, the so-called $\text{invoke}$ number (serial number of the packet) and the values $\text{u0}$ to $\text{u15}$. 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 $\text{invoke}$ 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**

| HLD | Input for disabling the execution of the block. No packets are received nor transmitted when HLD = on. bool |
| u0..u15 | Values which are sent/written to the output values y0 to y15 of the paired block double |

**Outputs**

| iE | Displays the code of the last error. The error codes are listed below: long |
| 0 ...... No error |
Persistent errors originating in the initialization phase (< 0). Cannot be fixed automatically.

-1 .... Maximum number of blocks exceeded (> 64)
-2 .... Local ports mismatch; the lport parameter must be the same for all RDC blocks within one application
-3 .... Error opening socket (the UDP/IP protocol is not available)
-4 .... Error assigning local port (port already occupied by another service or application)
-5 .... Error setting the so-called non-blocking socket mode (the RDC block requires this mode)
-10 ... Error initializing the socket library
-11 ... Error initializing the socket library
-12 ... Error initializing the socket library

Temporary errors originating in any cycle of the code (> 0). Can be fixed automatically.

1 ...... Initialization successful, yet no data packet has been received
2 ...... Packet consistency error (incorrect length – transmission error or conflicting service/application is running)
4 ...... Error receiving packet (socket library error)
8 ...... Error sending packet (socket library error)

**:fresh** Elapsed time (in seconds) since the last received packet. Can be used for detection of an error in the paired block.

**:y0..y15** Values transmitted from the input ports u0 to u15 of the paired RDC block – data from the last packet received

**Parameters**

- **target**: Name or IP address running the paired RDC block. Broadcast address is allowed.
- **rport**: Remote port – address of the UDP/IP protocol service, it is recommended to keep the default value unless necessary (service/application conflict)
- **lport**: Local port – similar meaning as the rport parameter; remote port applies to the receiving machine, local port applies to the machine sending the packet
- **id**: Block ID – this number is contained within the data packet in order to reach the proper target block (all blocks on the target receive the packet but only the one with the corresponding id decomposes it and uses the data contained to update its outputs)
- **period**: The shortest time interval between transmitting/receiving packets (in seconds). The packets are transmitted/received during each execution of the block for period ≤ 0 while the positive values of this parameter are extremely useful when sending data out of the Simulink continuous models based on a Variable step solver.
Example

The following example explains the function of the RDC block. The constants 3 and 5 are sent from Computer1 to Computer2, where they appear at the 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 REX control system, the RexView 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 REX control 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 REX 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:

```c
double input(1) input_signal; /* declaration of a variable of type double, which corresponds with the u1 input of the block */
long output(2) output_signal; /* declaration of a variable of type long, which corresponds with the y2 output of the block */
```

```c
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 $y = an \cdot x^n + ... + a1 \cdot x + a0$. The first parameter defining the number of items is optional.
  - `double poly(n,x,vec)`
    - Evaluates the polynomial $y = vec[n] \cdot x^n + ... + vec[1] \cdot x + vec[0]$. 
  - `double scal(n,vec1,vec2)`
    - Evaluates the scalar product $y = vec1[0] \cdot vec2[0] + ... + vec1[n-1] \cdot vec2[n-1]$. 
  - `double scal(n,vec1,vec2,skip1,skip2)`
    - Evaluates the scalar product $y = vec1[0] \cdot vec2[0] + vec1[skip1] \cdot vec2[skip2] + ... + vec1[(n-1)\cdotskip1] \cdot vec2[(n-1)\cdotskip2]$. 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 convoluted product $y = vec1[0] \cdot vec2[n-1] + vec1[1] \cdot vec2[n-1] + ... + vec1[n-1] \cdot vec2[0]$. 
  - `double sum(n,vec)`
    - Sums the items in a vector, i.e. $y = vec[0] + vec[1] + ... + vec[n-1]$. 
  - `double sum([n,]val1,...,valn)`
    - Sums the items, i.e. $y = val1 + val2 + ... + valn$. 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 strsub(str, idx, len)
     Returns a substring of length len starting at index idx.

  long strlen(str)
     Returns string length (number of characters).

  long strfind(str, substr)
     Returns the position of a substring (starting at 0).

  long strfind(str, substr)
     Returns the position of a substring relative to the end of the string.

  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.
**CHAPTER 15. SPEC – SPECIAL BLOCKS**

`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
- `\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)
- `\[\...]` ... 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 (subtems 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 the REX Control System. In Simulink the output is displayed directly in the command window of Matlab.

  In order to view these debugging messages in the RexView 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 REX 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 num-
   ber. 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 REX algorithm. The data item is defined 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.

long GetExtLong(ItemID)
   See GetExtLong(ItemID).

double GetExtReal(ItemID)
   Similar to GetExtInt(ItemID) but for decimal numbers.

double GetExtDouble(ItemID)
   See GetExtReal(ItemID).

double GetExtString(ItemID)
   Similar to GetExtInt(ItemID) but for strings.

void SetExt(ItemID, value)
Sets the input/output/parameter of arbitrary block in REX algorithm to value. The data item is defined 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 item (long/double/string) is defined by the type of the value parameter.

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 or serial line (RS-232 or RS-485). Only a brief list of available functions is given below, see the example projects of the REX Control System for more details.

  long Open(long type, long 1clIP, long 1clPort, long rmtIP, long rmtPort)
  Opens a socket or COM port according to the type parameter. Connect is performed for TCP client. Identification number (the so-called handle) of socket or COM port is returned. If a negative value is returned, the opening/connection was not successful.

  long Open(long type, string comname, long baudrate, long parity)
  Modification of the Open() function for opening a serial line.

  long Open(long type, string filename)
  Modification of the Open() function for opening a file.

  long Open(long type, string localname, long locPort, string remotename, long remPort)
  Modification of the Open() function for opening a TCP or UDP socket.
long OpenFile(string filename)
    Modification of the Open() function for opening a file.

long OpenCom(string comname, long baudrate, long parity)
    Modification of the Open() function for opening a serial line.

long OpenUDP(string localname, long lolPort, string remotename, long remPort)
    Modification of the Open() function for opening a UDP socket.

long OpenTCPsvr(string localname, long lolPort)
    Modification of the Open() function for opening a TCP socket - server, listening.

long OpenTCPcli(string remotename, long remPort)
    Modification of the Open() function for opening a TCP socket - client.

long OpenI2C(string devicename)
    Modification of the Open() function for opening an I2C bus.

long OpenMemory(string devicename, long baseaddr, long size)
    Modification of the Open() function for mapping physical memory.

long OpenSPI(string devicename)
    Modification of the Open() function for opening a SPI bus.

long Close(long handle)
    Closes the socket, serial line, file or any device opened by the Open function or its modifications.

void GetOptions(long handle, long params[])
    Reads parameters to the params array. The array size must be at least 2 for a socket and 22 for serial line.

void SetOptions(long handle, long params[])
    Sets the parameters of a socket or serial line.

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 WriteRead(long handle, long addr, long bufW[], long cntW, long bufR[], long cntR)
Communication over the I2C or SPI bus. Works only in Linux operating
system on devices with the I2C or SPI bus (e.g. Raspberry Pi or ALIX).
Sends and receives data to/from the slave device with address addr. The
parameter handle is returned by the OpenI2C or OpenSPI functions, whose
parameter defines the device name (according to the operating system).
The function returns 0 or an error code.

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 REX control system 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 ex-
pected directly in the project directory or in the directories specified by the -I
command line option of the RexComp 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:

\[
\text{double function(val1,\ldots,valn)}
\]

Vector is defined as a sequence of values of type double.

\[
\text{double function(n,val1,\ldots,valn)}
\]

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.

\[
\text{double function(n,vec)}
\]

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 \( x=\text{vec}[10] \); is neither a syntax nor runtime error, the value is not defined. More importantly, it is possible to write \( \text{vec}[11]=x; \), 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 3rd (parity) parameters 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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HLD</strong></td>
<td>Hold – the block code is not executed if the input is set to <strong>on</strong></td>
</tr>
<tr>
<td><strong>RESET</strong></td>
<td>Rising edge resets the block. The block gets initialized again (all global variables are cleared and the Init() function is called).</td>
</tr>
<tr>
<td><strong>u0..u15</strong></td>
<td>Input signals which are accessible from the script</td>
</tr>
</tbody>
</table>

Outputs

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>iE</strong></td>
<td>Runtime error code. Unless ( iE = 0 ) or ( iE = -1 ) the algorithm is stopped until it is reinitialized by the RESET input or by restarting the executive</td>
</tr>
</tbody>
</table>

- 0 ..... No error occurred, the whole main() function was executed (also the init() function).
- -1 .... The execution was suspended using the Suspend() command, i.e. the execution will resume as soon as the REXLANG block is executed again
- xxx ... Error code of the REX Control System, see Appendix B

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>y0..y15</strong></td>
<td>Output signals which can be set from within the script</td>
</tr>
</tbody>
</table>

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>srcname</strong></td>
<td>Source file name</td>
</tr>
<tr>
<td><strong>srctype</strong></td>
<td>Coding of source file</td>
</tr>
</tbody>
</table>

- 1: **C-like** Text file respecting the C-like syntax described above
- 2: **STL** Text file respecting the IEC61131-3 standard. The standard is implemented with the same limitations as the C-like script (i.e. no structures, only INT, REAL and STRING data types, function blocks are global variables VAR_INPUT, outputs are global variables VAR_OUTPUT, parameters are global variables VAR_PARAMETER, standard functions according to specification, system and communication functions are the same as in C-like).
- 3: **RLB** 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.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>stack</strong></td>
<td>Stack size defined as number of variables. Default and recommended value is 0, which enables automatic estimation of the necessary stack size.</td>
</tr>
</tbody>
</table>
**debug**

Debug level – checking is safer but slows down the execution of the algorithm. Option **No check** can crash REX application on target platform if code is incorrect.

1 .... No check
2 .... Basic check
3 .... Full check

**strs**

Total size of buffer for strings. Enter the maximum number of characters to allocate memory for. The default value 0 means that the buffer size is determined automatically.

**p0..p15**

Parameters which are accessible from the script

---

**Example C-like**

The following example shows a simple code to sum two input signals and also sum two user-defined parameters.

```c
#include <stdio.h>

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

<table>
<thead>
<tr>
<th>Function/Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM_Axis – Motion control axis</td>
<td>374</td>
</tr>
<tr>
<td>MC_AccelerationProfile, MCP_AccelerationProfile – Acceleration profile</td>
<td>380</td>
</tr>
<tr>
<td>MC_Halt, MCP_Halt – Stopping a movement (interruptible)</td>
<td>384</td>
</tr>
<tr>
<td>MC_HaltSuperimposed, MCP_HaltSuperimposed – Stopping a movement (superimposed and interruptible)</td>
<td>385</td>
</tr>
<tr>
<td>MC_Home, MCP_Home – Homing</td>
<td>386</td>
</tr>
<tr>
<td>MC_MoveAbsolute, MCP_MoveAbsolute – Move to position (absolute coordinate)</td>
<td>388</td>
</tr>
<tr>
<td>MC_MoveAdditive, MCP_MoveAdditive – Move to position (relative to previous motion)</td>
<td>392</td>
</tr>
<tr>
<td>MC_MoveRelative, MCP_MoveRelative – Move to position (relative to execution point)</td>
<td>395</td>
</tr>
<tr>
<td>MC_MoveSuperimposed, MCP_MoveSuperimposed – Superimposed move</td>
<td>398</td>
</tr>
<tr>
<td>MC_MoveContinuousAbsolute, MCP_MoveContinuousAbsolute – Move to position (absolute coordinate)</td>
<td>401</td>
</tr>
<tr>
<td>MC_MoveContinuousRelative, MCP_MoveContinuousRelative – Move to position (relative to previous motion)</td>
<td>404</td>
</tr>
<tr>
<td>MC_MoveVelocity, MCP_MoveVelocity – Move with constant velocity</td>
<td>408</td>
</tr>
<tr>
<td>MC_PositionProfile, MCP_PositionProfile – Position profile</td>
<td>412</td>
</tr>
<tr>
<td>MC_Power – Axis activation (power on/off)</td>
<td>416</td>
</tr>
<tr>
<td>MC_ReadActualPosition – Read actual position</td>
<td>417</td>
</tr>
<tr>
<td>MC_ReadAxisError – Read axis error</td>
<td>418</td>
</tr>
<tr>
<td>MC_ReadBoolParameter – Read axis parameter (bool)</td>
<td>419</td>
</tr>
<tr>
<td>MC_ReadParameter – Read axis parameter</td>
<td>420</td>
</tr>
</tbody>
</table>
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 REX control system, 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 the REX control system 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 REX control 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 the REX control system, 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 REX control 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

Function Description

The RM_AXIS block is a cornerstone of the motion control solution within the REX control 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

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLD</td>
<td>Hold</td>
</tr>
<tr>
<td></td>
<td>off ... Motion is allowed</td>
</tr>
<tr>
<td></td>
<td>on ...  Axis is halted and no motion is possible</td>
</tr>
<tr>
<td>ActualPos</td>
<td>Current position of the axis (feedback)</td>
</tr>
<tr>
<td>ActualVelocity</td>
<td>Current velocity of the axis (feedback)</td>
</tr>
</tbody>
</table>
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 reference connections are allowed)  double
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  Error code  error
i  REX 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
MaxPositionLag  Maximal position lag  double
MaxVelocitySystem  Maximal allowed velocity for system  double
MaxVelocityAppl  Maximal allowed velocity for application (MC blocks)  double
MaxAccelerationSystem  Maximal allowed acceleration for system  double
MaxAccelerationAppl  Maximal allowed acceleration for application (MC blocks)  double
MaxDecelerationSystem  Maximal allowed deceleration for system  double
**MaxDecelerationAppl**
Maximal allowed deceleration for application (MC blocks)

**MaxJerk**
Maximal allowed jerk [unit/s³]

**MaxTorque**
Maximal motor torque/force (0=not used)

**kta**
Torque-Acceleration ratio

**ReverseLimit**
Invert meaning of LIMN, LIMZ and LIMP inputs

---

**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 REX control 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.
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.

<table>
<thead>
<tr>
<th>BufferMode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aborting</td>
<td>The new motion is executed immediately</td>
</tr>
<tr>
<td>Buffered</td>
<td>The new motion is executed immediately after finishing the previous one, there is no blending</td>
</tr>
<tr>
<td>Blending low</td>
<td>The new motion is executed immediately after finishing the previous one, but the axis will not stop between the movements, the first motion ends with the lower limit for maximum velocity of both blocks at the first end-position</td>
</tr>
<tr>
<td>Blending high</td>
<td>The new motion is executed immediately after finishing the previous one, but the axis will not stop between the movements, the first motion ends with the higher limit for maximum velocity of both blocks at the first end-position</td>
</tr>
<tr>
<td>Blending previous</td>
<td>The new motion is executed immediately after finishing the previous one, but the axis will not stop between the movements, the first motion ends with the limit for maximum velocity of first block at the first end-position</td>
</tr>
<tr>
<td>Blending next</td>
<td>The new motion is executed immediately after finishing the previous one, but the axis will not stop between the movements, the first motion ends with the limit for maximum velocity of second block at the first end-position</td>
</tr>
</tbody>
</table>
Illustration of blending modes

- **Aborting**
- **Buffered**
- **Blending low**
- **Blending high**
- **Blending next**
- **Blending previous**

Legend:
- **Commanded velocity**
- **Active block 1 = false/Active block 2 = true**
- **Value of the maximum velocity \( v_1 = 30 \) (block 1)**
- **Value of the maximum velocity \( v_2 = 15 \) (block 2)**
MC_AccelerationProfile, MCP_AccelerationProfile — Acceleration profile

Block Symbols

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 than real size of arrays times and/or values) leads to unpredictable result and in some cases crashes whole runtime execution (The problem is platform dependent and currently it is known only for SIMULINK - crash of whole MATLAB).

Note 4: in the spline mode, polynomial is always 5th order and always in position (also for sibling block MC_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

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>TimeScale</td>
<td>Overall scale factor in time</td>
</tr>
<tr>
<td>AccelerationScale</td>
<td>Overall scale factor in value</td>
</tr>
<tr>
<td>Offset</td>
<td>Overall profile offset in value</td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
</tr>
<tr>
<td></td>
<td>1 . . . . . . Aborting (start immediately)</td>
</tr>
<tr>
<td></td>
<td>2 . . . . . . Buffered (start after finish of previous motion)</td>
</tr>
<tr>
<td></td>
<td>3 . . . . . . Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
</tr>
<tr>
<td></td>
<td>4 . . . . . . Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
</tr>
<tr>
<td></td>
<td>5 . . . . . . Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)</td>
</tr>
<tr>
<td></td>
<td>6 . . . . . . Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
</tbody>
</table>

Reference
### Error ID

<table>
<thead>
<tr>
<th>Error ID</th>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>error</td>
<td>REX general error</td>
</tr>
</tbody>
</table>

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>alg</td>
<td>Algorithm for interpolation</td>
<td>☁2 long</td>
</tr>
<tr>
<td></td>
<td>1. . . . Sequence of time/value pairs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. . . . Sequence of equidistant values</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. . . . Spline</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. . . . Equidistant spline</td>
<td></td>
</tr>
<tr>
<td>cSeg</td>
<td>Number of profile segments</td>
<td>☁3 long</td>
</tr>
<tr>
<td>times</td>
<td>Times when segments are switched</td>
<td>☁[0 30] double</td>
</tr>
<tr>
<td>values</td>
<td>Values or interpolating polynomial coefficients</td>
<td>☁[0 100 100 50]</td>
</tr>
</tbody>
</table>
MC_Halt, MCP_Halt – Stopping a movement (interruptible)

Block Symbols

Function Description

The MC_Halt and MCP_Halt blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The MC_Halt block commands a controlled motion stop and transfers the axis to the state DiscreteMotion. After the axis has reached zero velocity, the Done output is set to true immediately and the axis state is changed to Standstill.

Note 1: Block MC_Halt is intended for temporary stop of an axis under normal working conditions. Any next motion command which cancels the MC_Halt can be executed in nonbuffered mode (opposite to MC_Stop, which cannot be interrupted). The new command can start even before the stopping sequence was finished.

Inputs

- **uAxis** | Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)
- **Execute** | The block is activated on rising edge | bool
- **Deceleration** | Maximal allowed deceleration [unit/s²] | double
- **Jerk** | Maximal allowed jerk [unit/s³] | double

Outputs

- **yAxis** | Axis reference (only RM_Axis.axisRef–yAxis or yAxis–yAxis 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** | Error code | error

i ..... REX general error
MC_HaltSuperimposed, MCP_HaltSuperimposed – Stopping a movement (superimposed and interruptible)

Block Symbols

Function Description

The MC_HaltSuperimposed and MCP_HaltSuperimposed blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

Block MC_HaltSuperimposed commands a halt to all superimposed motions of the axis. The underlying motion is not interrupted.

Inputs

- **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^2] (double)
- **Jerk**: Maximal allowed jerk [unit/s^3] (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**: Error code (error)

i ..... REX general error
MC_Home, MCP_Home – Homing

Function Description

The MC_Home and MCP_Home blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The MC_Home block commands the axis to perform the "search home" sequence. The details of this sequence are described in PLCopen and can be set by parameters of the block. The "Position" input is used to set the absolute position when reference signal is detected. This Function Bock completes at "StandStill".

Note 1: Parameter/input BufferMode is not supported. Mode is always Aborting. It is not limitation, because homing is typically done once in initialization sequence before some regular movement is proceeded.

Note 2: Homing procedure requires some of RM_Axis block input connected. Depending on homing mode, ActualPos, ActualTorque, LimP, LimZ, LimN can be required. It is expected that only one method is used. Therefore, there are no separate inputs for zero switch and encoder reference pulse (both must be connected to 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>Velocity</td>
<td>Maximal allowed velocity [unit/s]</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s^2]</td>
</tr>
<tr>
<td>TorqueLimit</td>
<td>Maximal allowed torque/force</td>
</tr>
<tr>
<td>TimeLimit</td>
<td>Maximal allowed time for the whole algorithm [s]</td>
</tr>
<tr>
<td>DistanceLimit</td>
<td>Maximal allowed distance for the whole algorithm [unit]</td>
</tr>
<tr>
<td>Position</td>
<td>Requested target position (absolute) [unit]</td>
</tr>
<tr>
<td>Direction</td>
<td>Direction of movement (cyclic axis or special case only)</td>
</tr>
<tr>
<td>HomingMode</td>
<td>Homing mode algorithm</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxis</td>
<td>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
</tr>
<tr>
<td></td>
<td>i . . . . REX general error</td>
</tr>
</tbody>
</table>
**MC_MoveAbsolute, MCP_MoveAbsolute** — Move to position (absolute coordinate)

### 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 acts 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 acts almost the 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>AAxis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> reference connections are allowed)</td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>bool</td>
</tr>
<tr>
<td>Position</td>
<td>Requested target position (absolute) [unit]</td>
<td>double</td>
</tr>
<tr>
<td>Velocity</td>
<td>Maximal allowed velocity [unit/s]</td>
<td>double</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s²]</td>
<td>double</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
<td>double</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
<td>double</td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>1 . . . . Aborting (start immediately)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 . . . . Buffered (start after finish of previous motion)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 . . . . Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 . . . . Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 . . . . Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6 . . . . Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)</td>
<td></td>
</tr>
<tr>
<td>Direction</td>
<td>Direction of movement (cyclic axis or special case only)</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>1 . . . . Positive</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 . . . . Shortest</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 . . . . Negative</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 . . . . Current</td>
<td></td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxis</td>
<td>Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> reference connections are allowed)</td>
<td></td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>bool</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
<td>bool</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td>i</td>
<td>REX general error</td>
<td></td>
</tr>
</tbody>
</table>
MC_MoveAdditive, MCP_MoveAdditive – Move to position (relative to previous motion)

Block Symbols

Function Description

The **MC_MoveAdditive** and **MCP_MoveAdditive** blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the **MCP** version of the block.

The **MC_MoveAdditive** block moves an axis to specified position as fast as possible. The final position is determined by adding the value of **Distance** parameter to final position of previous motion block which was controlling the axis. If no further action is pending, final velocity is zero (axis moves to position and stops) otherwise it depends on blending mode. For blending purposes, start and stop velocity of this block is maximum velocity with direction respecting current and final position. If start velocity of next pending block is in opposite direction, then blending velocity is always zero.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input **Jerk** is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.
**Inputs**

- **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 start point) [unit]  
  double

- **Velocity**  
  Maximal allowed velocity [unit/s]  
  double

- **Acceleration**  
  Maximal allowed acceleration [unit/s^2]  
  double

- **Deceleration**  
  Maximal allowed deceleration [unit/s^2]  
  double

- **Jerk**  
  Maximal allowed jerk [unit/s^3]  
  double

- **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 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**  
  Error code  
  error

  i . . . . . REX general error
Example
MC_MoveRelative, MCP_MoveRelative – Move to position (relative to execution point)

Block Symbols

Function Description

The MC_MoveRelative and MCP_MoveRelative blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP version of the block.

The MC_MoveRelative block moves an axis to specified position as fast as possible. The final position is determined by adding the value of Distance parameter to the actual position at the moment of triggering the Execute input. If no further action is pending, final velocity is zero (axis moves to position and stops) otherwise it depends on blending mode. For blending purposes, start and stop velocity of this block is maximum velocity with direction respecting current and final position. If start velocity of next pending block is in opposite direction, then blending velocity is always zero.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.
CHAPTER 16. MC_SINGLE – MOTION CONTROL - SINGLE AXIS BLOCKS

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

**Velocity**  Maximal allowed velocity [unit/s]  double

**Acceleration**  Maximal allowed acceleration [unit/s²]  double

**Deceleration**  Maximal allowed deceleration [unit/s²]  double

**Jerk**  Maximal allowed jerk [unit/s³]  double

**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 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**  Error code  error

i . . . . . REX general error
Example

```
Example
```

![Diagram of MC_MoveRelative block 1 and block 2 with parameters]

- **Execute**
  - uAxis: Execute
  - yAxis: Execute

- **Distance**
  - Commanded distance

- **Velocity**
  - Commanded velocity

- **Acceleration**
  - Commanded acceleration

- **Deceleration**
  - Commanded deceleration

- **Jerk**
  - Commanded jerk

- **Buffer Mode**
  - Commanded buffer mode

- **Done**
  - Status: Done

- **Active**
  - Status: Active

- **Error**
  - Error ID

![Graphs of velocity and position over time]

- **Commanded velocity**
  - Velocity profile over time

- **Commanded position**
  - Position profile over time
CHAPTER 16. MC_SINGLE – MOTION CONTROL - SINGLE AXIS BLOCKS

MC_MoveSuperimposed, MCP_MoveSuperimposed – Superimposed move

Block Symbols

Function Description

The MC_MoveSuperimposed and MCP_MoveSuperimposed blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The MC_MoveSuperimposed block moves an axis to specified position as fast as possible (with respect to set limitations). Final position is specified by input parameter Distance. In case that the axis is already in motion at the moment of execution of the MC_MoveSuperimposed block, the generated values of position, velocity and acceleration are added to the values provided by the previous motion block. If there is no previous motion, the block behaves in the same way as the MC_MoveRelative command.

Note: There is no BufferMode parameter which is irrelevant in the superimposed mode. If there is already an superimposed motion running at the moment of execution, the new block is started immediately (analogous to aborting mode).

Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)</td>
<td>Axis reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>bool</td>
</tr>
<tr>
<td>Distance</td>
<td>Requested target distance (relative to execution point) [unit]</td>
<td>double</td>
</tr>
<tr>
<td>VelocityDiff</td>
<td>Maximal allowed velocity [unit/s]</td>
<td>double</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s²]</td>
<td>double</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
<td>double</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
<td>double</td>
</tr>
</tbody>
</table>
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 Error code error

i ..... REX general error
Example

MC_MoveSuperimposed − block 2

MC_MoveRelative − block 1

Execute2

 Execute1

Axis

Execute

Distance

VelocityDiff

Acceleration

Deceleration

Jerk

yAxis

Done

CommandAborted

Busy

Active

Error

ErrorID

Execute

Distance

Velocity

Acceleration

Deceleration

Jerk

yAxis

Done

CommandAborted

Busy

Active

Error

ErrorID

Commanded velocity

velocity

velocity2

velocity1

Commanded position

position

Time [s]

0 5 10 15

0

0.5

1

0 5 10 15

0

0.5

1

0 5 10 15

0

0.5

1

0 5 10 15

0

0.5

1

0 5 10 15

0

0.5

1

0 5 10 15

0

0.5

1

0 5 10 15

0

0.5

1

0 5 10 15

0

0.5

1

0 5 10 15

0

0.5

1
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
CHAPTER 16. MC_SINGLE – MOTION CONTROL - SINGLE AXIS BLOCKS

block behaves in the same way as MC_MoveAbsolute.

**Inputs**

- **uAxis**: Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis reference connections are allowed)
- **Execute**: The block is activated on rising edge
- **Position**: Requested target position (absolute) [unit]
- **Velocity**: Maximal allowed velocity [unit/s]
- **Acceleration**: Maximal allowed acceleration [unit/s²]
- **Deceleration**: Maximal allowed deceleration [unit/s²]
- **Jerk**: Maximal allowed jerk [unit/s³]
- **BufferMode**: Buffering mode
  - 1 : Aborting (start immediately)
  - 2 : Buffered (start after finish of previous motion)
  - 3 : Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
  - 4 : Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
  - 5 : Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)
  - 6 : Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)
- **Direction**: Direction of movement (cyclic axis or special case only)
  - 1 : Positive
  - 2 : Shortest
  - 3 : Negative
  - 4 : Current
- **EndVelocity**: End velocity

**Outputs**

- **yAxis**: Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis reference connections are allowed)
- **InEndVelocity**: Algorithm finished
- **CommandAborted**: Algorithm was aborted
- **Busy**: Algorithm not finished yet
- **Active**: The block is controlling the axis
- **Error**: Error occurred
- **ErrorID**: Error code
  - i : REX general error
**CHAPTER 16. MC_SINGLE – MOTION CONTROL - SINGLE AXIS BLOCKS**

**MC_MoveContinuousRelative, MCP_MoveContinuousRelative – Move to position (relative to previous motion)**

**Block Symbols**

**Function Description**

The **MC_MoveContinuousRelative** and **MCP_MoveContinuousRelative** blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the **MCP_** version of the block.

The **MC_MoveContinuousRelative** block moves an axis to specified position as fast as possible. The final position is determined by adding the value of **Distance** parameter to the actual position at the moment of triggering the **Execute** input. If no further action is pending, final velocity is specified by parameter **EndVelocity**. For blending purposes, start and stop velocity of this block is maximum velocity with direction respecting current and final position. If start velocity of next pending block is in opposite direction, then blending velocity is always zero.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input **Jerk** is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

Note 1: If the **EndVelocity** is set to zero value, the block behaves in the same way as **MC_MoveRelative**.
Note 2: If next motion command is executed before the final position is reached, the block behaves in the same way as MC_MoveRelative.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)</td>
<td>reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>bool</td>
</tr>
<tr>
<td>Distance</td>
<td>Requested target distance (relative to execution point) [unit]</td>
<td>double</td>
</tr>
<tr>
<td>Velocity</td>
<td>Maximal allowed velocity [unit/s]</td>
<td>double</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s²]</td>
<td>double</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
<td>double</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
<td>double</td>
</tr>
</tbody>
</table>
CHAPTER 16. MC_SINGLE – MOTION CONTROL - SINGLE AXIS BLOCKS

BufferMode Buffering mode

1 ….. Aborting (start immediately)
2 ….. Buffered (start after finish of previous motion)
3 ….. Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
4 ….. Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
5 ….. Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)
6 ….. Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)

EndVelocity End velocity

Outputs

yAxis Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)

InEndVelocity PLCopen Done (algorithm finished)
CommandAborted PLCopen CommandAborted (algorithm was aborted)
Busy PLCopen Busy (algorithm not finished yet)
Active PLCopen Active (the block is controlling the axis)
Error PLCopen Error (error occurred)
ErrorID Error code

i ….. REX general error
MC_MoveVelocity, MCP_MoveVelocity — Move with constant velocity

Block Symbols

Function Description

The MC_MoveVelocity and MCP_MoveVelocity blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The MC_MoveVelocity block changes axis velocity to specified value as fast as possible and keeps the specified velocity until the command is aborted by another block or event.

Note: parameter Direction enumerate also shortest_way although for this block it is not valid value.

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>Velocity</td>
<td>Maximal allowed velocity [unit/s]</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s²]</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
</tr>
<tr>
<td>Direction</td>
<td>Direction of movement (cyclic axis or special case only)</td>
</tr>
</tbody>
</table>

1. . . . . Positive
2. . . . . Shortest
3. . . . . Negative
4. . . . . Current
**BufferMode** Buffering mode

1 .... Aborting (start immediately)
2 .... Buffered (start after finish of previous motion)
3 .... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
4 .... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
5 .... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)
6 .... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)

**Outputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxis</td>
<td>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis reference connections are allowed)</td>
</tr>
<tr>
<td>InVelocity</td>
<td>Requested velocity reached</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
</tr>
</tbody>
</table>

i .... REX general error
Example
MC_PositionProfile, MCP_PositionProfile – Position profile

Block Symbols

Function Description

The MC_PositionProfile and MCP_PositionProfile blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The MC_PositionProfile block commands a time-position locked motion profile. Block implements two possibilities for definition of time-position function:

1. sequence of values: the user defines a sequence of time-position pairs. In each time interval, the values of position are interpolated. Times sequence is in array times, position sequence is in array values. Time sequence must be increasing and must start with zero or zero must be between the first and last point. Execution always starts from zero time, so if the sequence start with negative time, part of the profile is not executed (could be used for debugging or time shift). For MC_VelocityProfile and MC_AccelerationProfile interpolation is linear, but for MC_PositionProfile, 3rd order polynomial is used in order to avoid steps in velocity.

2. spline: time sequence is the same as in previous case. Each interval is interpolate by 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 \textbf{values} must be set as vector in all cases, e.g. text string must not include semicolon.

Note 3: incorrect parameter \textbf{cSeg} (higher then real size of arrays \textbf{times} and/or \textbf{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 \texttt{MC\_VelocityProfile} and \texttt{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 \texttt{-707} (step). It is recommended to use \texttt{BufferMode=BlendingNext} to eliminate the problem with start velocity.

\textbf{Inputs}

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{uAxis}</td>
<td>Axis reference (only \texttt{RM_Axis.axisRef-uAxis} or \texttt{yAxis-uAxis} connections are allowed)</td>
<td></td>
</tr>
<tr>
<td>\textbf{Execute}</td>
<td>The block is activated on rising edge</td>
<td>\textbf{bool}</td>
</tr>
<tr>
<td>\textbf{TimeScale}</td>
<td>Overall scale factor in time</td>
<td>\textbf{double}</td>
</tr>
<tr>
<td>\textbf{PositionScale}</td>
<td>Overall scale factor in value</td>
<td>\textbf{double}</td>
</tr>
<tr>
<td>\textbf{Offset}</td>
<td>Overall profile offset in value</td>
<td></td>
</tr>
<tr>
<td>\textbf{BufferMode}</td>
<td>Buffering mode</td>
<td>\textbf{long}</td>
</tr>
</tbody>
</table>

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)

\textbf{Outputs}

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{yAxis}</td>
<td>Axis reference (only \texttt{RM_Axis.axisRef-uAxis} or \texttt{yAxis-uAxis} connections are allowed)</td>
<td></td>
</tr>
<tr>
<td>\textbf{Done}</td>
<td>Algorithm finished</td>
<td>\textbf{bool}</td>
</tr>
<tr>
<td>\textbf{CommandAborted}</td>
<td>Algorithm was aborted</td>
<td>\textbf{bool}</td>
</tr>
<tr>
<td>\textbf{Busy}</td>
<td>Algorithm not finished yet</td>
<td>\textbf{bool}</td>
</tr>
<tr>
<td>\textbf{Active}</td>
<td>The block is controlling the axis</td>
<td>\textbf{bool}</td>
</tr>
<tr>
<td>\textbf{Error}</td>
<td>Error occurred</td>
<td>\textbf{bool}</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>alg</td>
<td>Algorithm for interpolation</td>
<td>○2  long</td>
</tr>
<tr>
<td></td>
<td>1 ..... Sequence of time/value pairs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ..... Sequence of equidistant values</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 ..... Spline</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 ..... Equidistant spline</td>
<td></td>
</tr>
<tr>
<td>cSeg</td>
<td>Number of profile segments</td>
<td>○3  long</td>
</tr>
<tr>
<td>times</td>
<td>Times when segments are switched</td>
<td>○[0 30] double</td>
</tr>
<tr>
<td>values</td>
<td>Values or interpolating polynomial coefficients (a0, a1, a2, ...)</td>
<td>○[0 100 100 50] double</td>
</tr>
</tbody>
</table>

Error ID: Error code
- i ..... REX general error
Example

```
Execute

TimeScale 1.0
posScale 1.0
offset 0
mode1 2
MC_PositionProfile
uAxis Execute
TimeScale PositionScale Offset BufferMode
yAxis Done
CommandAborted Busy Active Error ErrorID

execute

Axis

0 0.5 1 1.5 2 2.5 3

acceleration

0 0.5 1 1.5 2 2.5 3

velocity

0 0.5 1 1.5 2 2.5 3

position

0 0.5 1 1.5 2 2.5 3
```
**MC_Power** – Axis activation (power on/off)

**Block Symbol**                **Licence: MOTION CONTROL**

```
+-------------------------------------------------
| uAxis  | yAxis  | Status |
| Busy   | Active | Error  |
| Enable | ErrorID|
+-------------------------------------------------
  MC_Power
```

**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**

- **uAxis**: Axis reference (only **RM_Axis.axisRef** or **yAxis** connections are allowed)
- **Enable**: Block function is enabled, bool

**Outputs**

- **yAxis**: Axis reference (only **RM_Axis.axisRef** or **yAxis** connections are allowed)
- **Status**: Effective state of the power stage, bool
- **Busy**: Algorithm not finished yet, bool
- **Active**: The block is controlling the axis, bool
- **Error**: Error occurred, bool
- **ErrorID**: Error code
  - 1: REX general error
MC_ReadActualPosition – Read actual position

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

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Block function is enabled</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>Output value is valid</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
</tr>
<tr>
<td></td>
<td>i ..... REX general error</td>
</tr>
<tr>
<td>Position</td>
<td>Actual absolute position</td>
</tr>
</tbody>
</table>
MC_ReadAxisError – Read axis error

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 duplicate information about an error which is also accessible on the ErrorID output of the RM_Axis block.

Inputs

- **uAxis**: Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)
- **Enable**: Block function is enabled

Outputs

- **yAxis**: Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)
- **Valid**: Output value is valid
- **Busy**: Algorithm not finished yet
- **Error**: Error occurred
- **ErrorID**: Error code
  
i ..... REX general error
- **AxisErrorID**: Error code read from axis
  
i ..... REX general error
MC_ReadBoolParameter – Read axis parameter (bool)

Block Symbol

Function Description

The block `MC_ReadBoolParameter` displays actual value of various signals related to the connected axis on its Value output. The user chooses from a set of accessible logical variables by setting the ParameterNumber input. The output value is valid only while the block is activated by the logical Enable input.

The block displays the parameters and outputs of `RM_Axis` block and is implemented for sake of compatibility with the PLCopen specification.

Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> reference connections are allowed)</td>
<td></td>
</tr>
<tr>
<td>Enable</td>
<td>Block function is enabled</td>
<td>bool</td>
</tr>
<tr>
<td>ParameterNumber</td>
<td>Parameter ID</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>4 ..... Enable sw positive limit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 ..... Enable sw negative limit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6 ..... Enable position lag monitoring</td>
<td></td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxis</td>
<td>Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> reference connections are allowed)</td>
<td></td>
</tr>
<tr>
<td>Valid</td>
<td>Output value is valid</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td></td>
<td>i ..... REX general error</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Parameter value</td>
<td>bool</td>
</tr>
</tbody>
</table>
MC_ReadParameter – Read axis parameter

Block Symbol

<table>
<thead>
<tr>
<th>uAxis</th>
<th>yAxis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Busy</td>
</tr>
<tr>
<td>ParameterNumber</td>
<td>Error</td>
</tr>
</tbody>
</table>

MC_ReadParameter

Function Description

The block MC_ReadParameter displays actual value of various system variables of the connected axis on its Value output. The user chooses from a set of accessible variables by setting the ParameterNumber input. The output value is valid only while the block is activated by the logical Enable input.

The block displays the parameters and outputs of RM_Axis block and is implemented for sake of compatibility with the PLCOpen specification.

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Block function is enabled</td>
</tr>
<tr>
<td>ParameterNumber</td>
<td>Parameter ID</td>
</tr>
</tbody>
</table>

| 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxis</td>
<td>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis reference connections are allowed)</td>
<td>bool</td>
</tr>
<tr>
<td>Valid</td>
<td>Output value is valid</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td></td>
<td>i .... REX general error</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Parameter value</td>
<td>double</td>
</tr>
</tbody>
</table>
MC_ReadStatus – Read axis status

Function Description

The block MC_ReadStatus indicates the state of the connected axis on its logical output signals. The values of the states are valid only while the `Enable` input is set to nonzero value. This state is indicated by `Valid` output.

Inputs

- **uAxis**  
  Axis reference (only `RM_Axis.axisRef-uAxis` or `yAxis-uAxis` reference connections are allowed)  
- **Enable**  
  Block function is enabled

Outputs

- **yAxis**  
  Axis reference (only `RM_Axis.axisRef-uAxis` or `yAxis-uAxis` reference connections are allowed)  
- **Valid**  
  Output value is valid  
- **Busy**  
  Algorithm not finished yet  
- **Error**  
  Error occurred  
- **ErrorID**  
  Error code
  - i ..... REX general error  
- **ErrorStop**  
  Axis is in the ErrorStop state  
- **Disabled**  
  Axis is in the Disabled state  
- **Stopping**  
  Axis is in the Stopping state  
- **StandStill**  
  Axis is in the StandStill state  
- **DiscreteMotion**  
  Axis is in the DiscreteMotion state
ContinuousMotion  Axis is in the ContinuousMotion state  bool
SynchronizedMotion  Axis is in the SynchronizedMotion state  bool
Homing  Axis is in the Homing state  bool
ConstantVelocity  Axis is moving with constant velocity  bool
Accelerating  Axis is accelerating  bool
Decelerating  Axis is decelerating  bool
**MC_Reset — Reset axis errors**

**Block Symbol**

![Block Symbol](image)

**Function Description**

The **MC_Reset** block makes the transition from the state ErrorStop to StandStill by resetting all internal axis-related errors.

**Inputs**

- **uAxis**: Axis reference (only `RM_Axis.axisRef`-`uAxis` or `yAxis`-`uAxis` reference connections are allowed)
- **Execute**: The block is activated on rising edge

**Outputs**

- **yAxis**: Axis reference (only `RM_Axis.axisRef`-`uAxis` or `yAxis`-`uAxis` reference connections are allowed)
- **Done**: Algorithm finished
- **Busy**: Algorithm not finished yet
- **Error**: Error occurred
- **ErrorID**: Error code
  - `i` .... REX general error

**Licence**: MOTION CONTROL
MC_SetOverride, MCP_SetOverride — Set override factors

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 reacalculation of movement’s path if a block like MC_MoveAbsolute commands the axis. This recalculation needs lot of CPU time and also numerical problem could appear. For this reasons, a deadband (parameter diff) is established. The movement’s path recalculation is proceeded only if one of the factors is changed more then the deadband.

Note: all factor must be positive. Factor greater then 1.0 are possible, but often lead to overshooting of axis limits and failure of movement (with errorID=-700 - invalid parameter; if factor is set before start of block) or error stop of axis (with errorID=-701 - out of range; if factor is changed within movement and actual value overshoot limit).

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Block function is enabled</td>
</tr>
<tr>
<td>VelFactor</td>
<td>Velocity multiplication factor</td>
</tr>
<tr>
<td>AccFactor</td>
<td>Acceleration/deceleration multiplication factor</td>
</tr>
</tbody>
</table>

 Licence: MOTION CONTROL
JerkFactor  Jerk multiplication factor  double

Outputs

  yAxis  Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)  reference
  Enabled  Block function is enabled  bool
  Busy  Algorithm not finished yet  bool
  Error  Error occurred  bool
  ErrorID  Error code
  i .....  REX general error  error

Parameter

  diff  Deadband (difference for recalculation)  $\downarrow 0.0 \uparrow 1.0 \circ 0.1$  double
MC_Stop, MCP_Stop – Stopping a movement

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=\texttt{true} and Execute=\texttt{false} the axis goes to state StandStill.

Note 1: parameter/input BufferMode is not supported. Mode is always Aborting.

Note 2: Failing stop-command could be dangerous. This block does not generate invalid-parameter-error but tries to stop the axis anyway (e.g. uses parameters from RM_Axis or generates error-stop-sequence).

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s^2]</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s^3]</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxis</td>
<td>Axis reference (only RM_Axis.axisRef-yAxis or yAxis-yAxis reference connections are allowed)</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
</tbody>
</table>
Active   The block is controlling the axis  
Error    Error occurred  
ErrorID  Error code  
          i ..... REX general error
MC_TorqueControl, MCP_TorqueControl – Torque/force 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

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>Torque</td>
<td>Maximal allowed torque/force</td>
</tr>
<tr>
<td>TorqueRamp</td>
<td>Maximal allowed torque/force ramp</td>
</tr>
<tr>
<td>Velocity</td>
<td>Maximal allowed velocity [unit/s]</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s^2]</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s^2]</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s^3]</td>
</tr>
</tbody>
</table>
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  Error code  error
i .... REX general error

Parameter

kma  Torque/force to acceleration ratio  double
Example
CHAPTER 16. MC_SINGLE – MOTION CONTROL - SINGLE AXIS BLOCKS

MC_VelocityProfile, MCP_VelocityProfile – Velocity profile

Block Symbols

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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Axis reference (only <code>RM_Axis.axisRef</code> or <code>yAxis</code> connections are allowed)</td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>bool</td>
</tr>
<tr>
<td>TimeScale</td>
<td>Overall scale factor in time</td>
<td>double</td>
</tr>
<tr>
<td>VelocityScale</td>
<td>Overall scale factor in value</td>
<td>double</td>
</tr>
<tr>
<td>Offset</td>
<td>Overall profile offset in value</td>
<td>double</td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>1 . . . . . Aborting (start immediately)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 . . . . . Buffered (start after finish of previous motion)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 . . . . . Blending low (start after finishing the previous motion,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>previous motion finishes with the lowest velocity of both</td>
<td></td>
</tr>
<tr>
<td></td>
<td>commands)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 . . . . . Blending high (start after finishing the previous motion,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>previous motion finishes with the lowest velocity of both</td>
<td></td>
</tr>
<tr>
<td></td>
<td>commands)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 . . . . . Blending previous (start after finishing the previous</td>
<td></td>
</tr>
<tr>
<td></td>
<td>motion, previous motion finishes with its final velocity)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6 . . . . . Blending next (start after finishing the previous motion,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>previous motion finishes with the starting velocity of the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>next block)</td>
<td></td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxis</td>
<td>Axis reference (only <code>RM_Axis.axisRef</code> or <code>yAxis</code> connections are allowed)</td>
<td></td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>bool</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
<td>bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td></td>
<td>i . . . . . REX general error</td>
<td></td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>alg</td>
<td>Algorithm for interpolation</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>times</td>
<td>Times when segments are switched</td>
<td>double</td>
</tr>
<tr>
<td>values</td>
<td>Values or interpolating polynomial coefficients (a0, a1, a2, ...)</td>
<td>double</td>
</tr>
<tr>
<td>cSeg</td>
<td>Number of profile segments</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Example
**MC_WriteBoolParameter — Write axis parameter (bool)**

**Block Symbol**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Axis reference (only <code>RM_Axis.axisRef</code> or <code>yAxis-uAxis</code> reference connections are allowed)</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>ParameterNumber</td>
<td>Parameter ID</td>
</tr>
<tr>
<td>Value</td>
<td>Parameter value</td>
</tr>
<tr>
<td>yAxis</td>
<td>Axis reference (only <code>RM_Axis.axisRef</code> or <code>yAxis-uAxis</code> reference connections are allowed)</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
</tr>
</tbody>
</table>

**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` or `yAxis-uAxis` reference connections are allowed)
- **Execute**: The block is activated on rising edge
- **ParameterNumber**: Parameter ID
  - 4...: Enable sw positive limit
  - 5...: Enable sw negative limit
  - 6...: Enable position lag monitoring
- **Value**: Parameter value

**Outputs**

- **yAxis**: Axis reference (only `RM_Axis.axisRef` or `yAxis-uAxis` reference connections are allowed)
- **Done**: Algorithm finished
- **Busy**: Algorithm not finished yet
- **Error**: Error occurred
- **ErrorID**: Error code
  - i...: REX 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

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Axis reference (only RM_Axis.axisRef – uAxis or yAxis – uAxis connections are allowed)</td>
<td>bool</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>bool</td>
</tr>
<tr>
<td>ParameterNumber</td>
<td>Parameter ID</td>
<td>long</td>
</tr>
<tr>
<td>Value</td>
<td>Parameter value</td>
<td>double</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ParameterNumber</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Positive sw limit switch</td>
</tr>
<tr>
<td>3</td>
<td>Negative sw limit switch</td>
</tr>
<tr>
<td>7</td>
<td>Maximal position lag</td>
</tr>
<tr>
<td>8</td>
<td>Maximal velocity (system)</td>
</tr>
<tr>
<td>9</td>
<td>Maximal velocity (appl)</td>
</tr>
<tr>
<td>13</td>
<td>Maximal acceleration (appl.)</td>
</tr>
<tr>
<td>15</td>
<td>Maximal deceleration (appl.)</td>
</tr>
<tr>
<td>16</td>
<td>Maximal jerk</td>
</tr>
<tr>
<td>1001</td>
<td>Maximal torque/force</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxis</td>
<td>Axis reference (only RM_Axis.axisRef – uAxis or yAxis – uAxis connections are allowed)</td>
<td>bool</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
</tbody>
</table>
ErrorID  Error code  error
i       REX general error
**RM_AxisOut – Axis output**

**Block Symbol**

![Block Symbol](image)

**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 **REX** 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 RexView tool where the blocks are sorted by execution order.

Note 2: almost all blocks do not work with torque so commanded torque is 0. Commanded acceleration and torque should be used as feed-forward value for position/velocity controller so this value does not make any problem.

**Inputs**

- **uAxis**
  - Axis reference that must be connected to **axisRef** of the **RM_Axis** block

  (direct or indirect throw output **yAxis** of some other block)

**Input**

- **uAxis**
  - Axis reference (only **RM_Axis.axisRef-uAxis** or **yAxis-uAxis** reference connections are allowed)
<table>
<thead>
<tr>
<th>Outputs</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos0</td>
<td>Current commanded position [unit]</td>
<td>double</td>
</tr>
<tr>
<td>vel0</td>
<td>Current commanded velocity [unit/s]</td>
<td>double</td>
</tr>
<tr>
<td>acc0</td>
<td>Current commanded acceleration [unit/s$^2$]</td>
<td>double</td>
</tr>
<tr>
<td>trq0</td>
<td>Current commanded torque/force (if generated)</td>
<td>double</td>
</tr>
<tr>
<td>pos1</td>
<td>Next step commanded position [unit]</td>
<td>double</td>
</tr>
<tr>
<td>vel1</td>
<td>Next step commanded velocity [unit/s]</td>
<td>double</td>
</tr>
<tr>
<td>acc1</td>
<td>Next step commanded acceleration/deceleration [unit/s$^2$]</td>
<td>double</td>
</tr>
<tr>
<td>trq1</td>
<td>Next step commanded torque/force (if generated)</td>
<td>double</td>
</tr>
<tr>
<td>pos2</td>
<td>2nd next step commanded position [unit]</td>
<td>double</td>
</tr>
<tr>
<td>vel2</td>
<td>2nd next step commanded velocity [unit/s]</td>
<td>double</td>
</tr>
<tr>
<td>acc2</td>
<td>2nd next step commanded acceleration/deceleration [unit/s$^2$]</td>
<td>double</td>
</tr>
<tr>
<td>trq2</td>
<td>2nd next step commanded torque/force (if generated)</td>
<td>double</td>
</tr>
<tr>
<td>iState</td>
<td>State of the axis</td>
<td>long</td>
</tr>
<tr>
<td>0</td>
<td>Disabled</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Stand still</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Homing</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Discrete motion</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Continuous motion</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Synchronized motion</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Coordinated motion</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Stopping</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Error stop</td>
<td></td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td>0</td>
<td>REX general error</td>
<td></td>
</tr>
<tr>
<td>iTick</td>
<td>Current tick</td>
<td>long</td>
</tr>
</tbody>
</table>
RM_AxisSpline — Commanded values interpolation

Function Description

There are lot of motion control blocks which implement complicated algorithms so they require bigger sampling period (typical update rate is from 10 to 200ms). On the other side, the motor driver usually requires small sampling period for smooth/waveless movement. The RM_AxisSpline block solves this problem of multirate execution of motion planning and motion control levels. The block can run in different task than other motion control blocks with highest sampling period possible. It interpolates commanded position, velocity and torque and generates smooth curve which is more suited for motor driver controllers.

There are two possibilities of connection to RM_Axis block: connect all necessary values (outputs of the block RM_AxisOut) as input of interpolating block or use only axis reference and read the state directly. This block uses axis reference. For correct synchronization between two tasks, the block RM_Axis must be executed first followed by all axis related motion control blocks and finally by block RM_AxisOut at the end.

Note 1: For interpolation of position signal, 3rd order polynomial $p(t)$ is used, where $p_s(0) = pos0, p_s(t_0) = pos1, \frac{dp_s(t)}{dt} t=0 = vel0, \frac{dp_s(t)}{dt} t=t_S = vel1$. 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 REX 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 RexView 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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>Commanded interpolated position [unit]</td>
<td>double</td>
</tr>
<tr>
<td>vel</td>
<td>Commanded interpolated velocity [unit/s]</td>
<td>double</td>
</tr>
<tr>
<td>trq</td>
<td>Commanded interpolated torque/force</td>
<td>double</td>
</tr>
<tr>
<td>iState</td>
<td>Interpolator state/error</td>
<td>long</td>
</tr>
</tbody>
</table>

- **pos**
  - **Commanded interpolated position** [unit] (double)

- **vel**
  - **Commanded interpolated velocity** [unit/s] (double)

- **trq**
  - **Commanded interpolated torque/force** (double)

- **iState**
  - **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

Function Description

The RM_Track block includes few useful functions.

If the input TRACK is active (not zero), the block tries to track requested position (input pos) with respect to the limits for velocity, acceleration/deceleration and jerk. The block expects that requested position is changed in each step and therefore recalculates the path in each step. This is difference to MC_MoveAbsolute block, which does not allow to change target position while the movement is not finished. This mode is useful if position is generated out of the motion control subsystem, even thought the MC_PositionProfile block is better if whole path is known.

If the input 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

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>posvel</td>
<td>Requested target position or velocity [unit]</td>
</tr>
<tr>
<td>TRACKP</td>
<td>Position tracking mode</td>
</tr>
<tr>
<td>TRACKV</td>
<td>Velocity tracking mode</td>
</tr>
<tr>
<td>JOGP</td>
<td>Moving positive direction mode</td>
</tr>
<tr>
<td>JOGN</td>
<td></td>
</tr>
</tbody>
</table>
CHAP1ER 16. MC_SINGLE – MOTION CONTROL - SINGLE AXIS BLOCKS

JOGN  Moving negative direction mode  bool

Parameters

pv  Maximal allowed velocity [unit/s]  double
pa  Maximal allowed acceleration [unit/s²]  double
pd  Maximal allowed deceleration [unit/s²]  double
pj  Maximal allowed jerk [unit/s³]  double
iLen  Length of buffer for estimation  ⊗10 long

Outputs

yAxis  Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference 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  Error code  error
i  . . . . . . REX general error
Chapter 17

MC_MULTI – Motion control - multi axis blocks

Contents

- MC_CamIn, MCP_CamIn – Engage the cam ........................................ 446
- MC_CamOut – Disengage the cam ..................................................... 450
- MCP_CamTableSelect – Cam definition ............................................ 452
- MC_CombineAxes, MCP_CombineAxes – Combine the motion of 2 axes into a third axis ................................................................. 454
- MC_GearIn, MCP_GearIn – Engage the master/slave velocity ratio ............................................................... 457
- MC_GearInPos, MCP_GearInPos – Engage the master/slave velocity ratio in defined position ......................................................... 460
- MC_GearOut – Disengage the master/slave velocity ratio .......... 465
- MC_PhasingAbsolute, MCP_PhasingAbsolute – Phase shift in synchronized motion (absolute coordinates) ......................... 467
- MC_PhasingRelative, MCP_PhasingRelative – Phase shift in synchronized motion (relative coordinates) ......................... 470

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

Function Description

The MC_CamIn and MCP_CamIn blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_version of the block.

The MC_CamIn block switches on a mode in which the slave axis is commanded to position which corresponds to the position of master axis transformed with with a function defined by the MCP_CamTableSelect block (connected to CamTableID input). Denoting the transformation as \( \text{Cam}(x) \), master axis position \( \text{PosM} \) and slave axis position \( \text{PosS} \), we obtain (for absolute relationship, without phasing): \( \text{PosS} = \text{Cam}(\text{PosM} - \text{MasterOffset})/\text{MasterScaling} \times \text{SlaveScaling} + \text{SlaveOffset} \). This form of synchronized motion of the slave axis is called electronic cam.

The cam mode is switched off by executing other motion block on slave axis with 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 \texttt{RampIn} parameter 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.

\textbf{Inputs}

\begin{itemize}
\item \texttt{uMaster} \quad Master axis reference \quad \textbf{reference}
\item \texttt{uSlave} \quad Slave axis reference \quad \textbf{reference}
\item \texttt{CamTableID} \quad Cam table reference (connect to MCP.CamTableSelect.CamTableID) \quad \textbf{reference}
\end{itemize}

\begin{itemize}
\item \texttt{Execute} \quad The block is activated on rising edge \quad \textbf{bool}
\item \texttt{MasterOffset} \quad Offset in cam table on master side [unit] \quad \textbf{double}
\item \texttt{SlaveOffset} \quad Offset in cam table on slave side [unit] \quad \textbf{double}
\item \texttt{MasterScaling} \quad Overall scaling factor in cam table on master side \quad \textbf{double}
\item \texttt{SlaveScaling} \quad Overall scaling factor in cam table on slave side \quad \textbf{double}
\item \texttt{StartMode} \quad Select relative or absolute cam table \quad \textbf{long}

\begin{itemize}
\item 1 \ldots Master relative
\item 2 \ldots Slave relative
\item 3 \ldots Both relative
\item 4 \ldots Both absolute
\end{itemize}
\item \texttt{BufferMode} \quad Buffering mode \quad \textbf{long}

\begin{itemize}
\item 1 \ldots Aborting (start immediately)
\item 2 \ldots Buffered (start after finish of previous motion)
\item 3 \ldots Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
\item 4 \ldots Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
\item 5 \ldots Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)
\item 6 \ldots Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)
\end{itemize}
\item \texttt{RampIn} \quad RampIn factor (0 = RampIn mode not used); average additive velocity (absolute value) during ramp-in process \quad \textbf{double}

\textbf{Outputs}

\begin{itemize}
\item \texttt{yMaster} \quad Master axis reference \quad \textbf{reference}
\item \texttt{ySlave} \quad Slave axis reference \quad \textbf{reference}
\item \texttt{InSync} \quad Slave axis reached the cam profile \quad \textbf{bool}
\item \texttt{CommandAborted} \quad Algorithm was aborted \quad \textbf{bool}
\item \texttt{Busy} \quad Algorithm not finished yet \quad \textbf{bool}
\item \texttt{Active} \quad The block is controlling the axis \quad \textbf{bool}
\item \texttt{Error} \quad Error occurred \quad \textbf{bool}
\item \texttt{ErrorID} \quad Error code \quad \textbf{error}

\begin{itemize}
\item i \ldots REX general error
\end{itemize}
**EndOfProfile**  Indicate end of cam profile (not periodic cam only)  
**SyncDistance**  Position deviation of the slave axis from synchronized position
MC_CamOut – Disengage the cam

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

- uSlave: Slave axis reference
- Execute: The block is activated on rising edge

Outputs

- ySlave: Slave axis reference
- Done: Algorithm finished
- Busy: Algorithm not finished yet
- Error: Error occurred
- ErrorID: Error code
  
  1: REX general error
Example

Execute

Active

Acceleration

Velocity

Position
CHAPTER 17. MC_MULTI – MOTION CONTROL - MULTI AXIS BLOCKS

MCP_CamTableSelect – Cam definition

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

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uMaster</td>
<td>Master axis reference</td>
<td>reference</td>
</tr>
<tr>
<td>uSlave</td>
<td>Slave axis reference</td>
<td>reference</td>
</tr>
</tbody>
</table>
**Execute**  The block is activated on rising edge  bool

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yMaster</td>
<td>Master axis reference</td>
<td>reference</td>
</tr>
<tr>
<td>ySlave</td>
<td>Slave axis reference</td>
<td>reference</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
<td>error</td>
</tr>
</tbody>
</table>

**CamTableID**  Cam table reference (connect to MC_CamIn.CamTableID)  reference

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>alg</td>
<td>Algorithm for interpolation</td>
<td>o2 long</td>
</tr>
<tr>
<td></td>
<td>1 . . . . . . Sequence of time/value pairs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 . . . . . . Sequence of equidistant values</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 . . . . . . Spline</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 . . . . . . Equidistant spline</td>
<td></td>
</tr>
<tr>
<td>cSeg</td>
<td>Number of profile segments</td>
<td>o3 long</td>
</tr>
<tr>
<td>Periodic</td>
<td>Indicate periodic cam profile</td>
<td>o on  bool</td>
</tr>
<tr>
<td>camname</td>
<td>Filename of special editor data file (filename is generated by system if parameter is empty)</td>
<td>string</td>
</tr>
<tr>
<td>mvalues</td>
<td>Master positions where segments are switched</td>
<td>o [0 30] double</td>
</tr>
<tr>
<td>svalues</td>
<td>Slave positions or interpolating polynomial coefficients (a0, a1, a2, ...)</td>
<td>o [0 100 100 0] double</td>
</tr>
</tbody>
</table>
MC_CombineAxes, MCP_CombineAxes – Combine the motion of 2 axes into a third axis

Function Description

The MC_CombineAxes block combines a motion of two master axes into a slave axis command. The slave axis indicates synchronized motion state. Following relationship holds:

\[
\text{SlavePosition} = \text{Master1Position} \cdot \frac{\text{GearRatioNumeratorM1}}{\text{GearRatioDenominatorM1}} + \text{Master2Position} \cdot \frac{\text{GearRatioNumeratorM2}}{\text{GearRatioDenominatorM2}}
\]

Negative number can be set in GearRatio... parameter to obtain the resulting slave movement in form of difference of master axes positions.

Inputs

- **uMaster1**: First master axis reference (reference)
- **uMaster2**: Second master axis reference (reference)
- **uSlave**: Slave axis reference (reference)
- **Execute**: The block is activated on rising edge (bool)
- **GearRatioNumeratorM1**: Numerator for the gear factor for master axis 1 (long)
- **GearRatioDenominatorM1**: Denominator for the gear factor for master axis 1 (long)
- **GearRatioNumeratorM2**: Numerator for the gear factor for master axis 2 (long)
- **GearRatioDenominatorM2**: Denominator for the gear factor for master axis 2 (long)
**BufferMode** Buffering mode  

1. Aborting (start immediately)  
2. Buffered (start after finish of previous motion)  
3. Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)  
4. Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)  
5. Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)  
6. Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)  

**RampIn** RampIn factor ($0 = \text{RampIn mode not used}$)  

<table>
<thead>
<tr>
<th>BufferMode</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aborting (start immediately)</td>
<td>long</td>
</tr>
<tr>
<td>2</td>
<td>Buffered (start after finish of previous motion)</td>
<td>long</td>
</tr>
<tr>
<td>3</td>
<td>Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
<td>long</td>
</tr>
<tr>
<td>4</td>
<td>Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
<td>long</td>
</tr>
<tr>
<td>5</td>
<td>Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)</td>
<td>long</td>
</tr>
<tr>
<td>6</td>
<td>Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)</td>
<td>long</td>
</tr>
</tbody>
</table>

**Outputs**  

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>yMaster1</strong></td>
<td>First master axis reference</td>
<td>reference</td>
</tr>
<tr>
<td><strong>yMaster2</strong></td>
<td>Second master axis reference</td>
<td>reference</td>
</tr>
<tr>
<td><strong>ySlave</strong></td>
<td>Slave axis reference</td>
<td>reference</td>
</tr>
<tr>
<td><strong>InSync</strong></td>
<td>Slave axis reached the cam profile</td>
<td>bool</td>
</tr>
<tr>
<td><strong>CommandAborted</strong></td>
<td>Algorithm was aborted</td>
<td>bool</td>
</tr>
<tr>
<td><strong>Busy</strong></td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td><strong>Active</strong></td>
<td>The block is controlling the axis</td>
<td>bool</td>
</tr>
<tr>
<td><strong>Error</strong></td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td><strong>ErrorID</strong></td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td><strong>SyncDistance</strong></td>
<td>Position deviation of the slave axis from synchronized position</td>
<td>double</td>
</tr>
</tbody>
</table>

i . . . . . REX general error
Example
**MC_GearIn, MCP_GearIn** – Engage the master/slave velocity ratio

**Block Symbols**

![Block Symbols Diagram]

**Function Description**

The **MC_GearIn** and **MCP_GearIn** blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the **MCP_** version of the block.

The **MC_GearIn** block commands the slave axis motion in such a way that a preset ratio between master and slave velocities is maintained. Considering the velocity of master axis $V_{elM}$ and velocity of slave axis $V_{elS}$, following relation holds (without phasing): $V_{elS} = V_{elM} \times \frac{\text{RatioNumerator}}{\text{RatioDenominator}}$. Position and acceleration is commanded to be consistent with velocity; position/distance ratio is also locked. This mode of synchronized motion is called electronic gear.

The gear mode is switched off by executing other motion block on slave axis with mode aborting or by executing a **MC_GearIn** block.

Similarly to the **MC_CamIn** block, ramp-in mode is activated if initial velocity of slave axis is different from master axis and gearing ratio. Parameters **Acceleration**, **Deceleration**, **Jerk** are used during ramp-in mode.

**Inputs**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uMaster</td>
<td>Master axis reference</td>
</tr>
<tr>
<td>uSlave</td>
<td>Slave axis reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>RatioNumerator</td>
<td>Gear ratio Numerator</td>
</tr>
<tr>
<td><strong>RatioDenominator</strong></td>
<td>Gear ratio Denominator</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td><strong>Acceleration</strong></td>
<td>Maximal allowed acceleration [unit/s²]</td>
</tr>
<tr>
<td><strong>Deceleration</strong></td>
<td>Maximal allowed deceleration [unit/s²]</td>
</tr>
<tr>
<td><strong>Jerk</strong></td>
<td>Maximal allowed jerk [unit/s³]</td>
</tr>
<tr>
<td><strong>BufferMode</strong></td>
<td>Buffering mode</td>
</tr>
<tr>
<td>1 ......</td>
<td>Aborting (start immediately)</td>
</tr>
<tr>
<td>2 ......</td>
<td>Buffered (start after finish of previous motion)</td>
</tr>
<tr>
<td>3 ......</td>
<td>Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
</tr>
<tr>
<td>4 ......</td>
<td>Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)</td>
</tr>
<tr>
<td>5 ......</td>
<td>Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)</td>
</tr>
<tr>
<td>6 ......</td>
<td>Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th><strong>yMaster</strong></th>
<th>Master axis reference</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ySlave</strong></td>
<td>Slave axis reference</td>
<td>reference</td>
</tr>
<tr>
<td><strong>InGear</strong></td>
<td>Slave axis reached gearing ratio</td>
<td>bool</td>
</tr>
<tr>
<td><strong>CommandAborted</strong></td>
<td>Algorithm was aborted</td>
<td>bool</td>
</tr>
<tr>
<td><strong>Busy</strong></td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td><strong>Active</strong></td>
<td>The block is controlling the axis</td>
<td>bool</td>
</tr>
<tr>
<td><strong>Error</strong></td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td><strong>ErrorID</strong></td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td>i ......</td>
<td>REX general error</td>
<td></td>
</tr>
</tbody>
</table>
Example

- **Velocity**
  - Master: 30
  - Slave: 30

- **Position**
  - Master: 100
  - Slave: 100

- **Mode**
  - Master: 2
  - Slave: 2

- **Jerk**
  - Master: 0
  - Slave: 0

- **Acceleration**
  - Master: 25
  - Slave: 50

- **Deceleration**
  - Master: 10
  - Slave: 20

- **Block Execution**
  - Block 1: GearIn
  - Block 2: MoveAbsolute

- **Active States**
  - Block 1: GearIn
  - Block 2: MoveAbsolute

- **Graphs**
  - Execute
  - Active
  - Acceleration
  - Velocity
  - Position

- **Legend**
  - Master
  - Slave
  - GearIn
  - MoveAbsolute

- **Error Handling**
  - CommandAborted
  - Busy
  - Active
  - Error
  - ErrorID
MC_GearInPos, MCP_GearInPos – Engage the master/slave velocity ratio in defined position

**Block Symbols**

<table>
<thead>
<tr>
<th>MC_GearInPos</th>
<th>MCP_GearInPos</th>
</tr>
</thead>
<tbody>
<tr>
<td>yMaster</td>
<td>yMaster</td>
</tr>
<tr>
<td>ySlave</td>
<td>ySlave</td>
</tr>
<tr>
<td>Execute</td>
<td>InSync</td>
</tr>
<tr>
<td>StartSync</td>
<td>CommandAborted</td>
</tr>
<tr>
<td>Velocity</td>
<td>Active</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Error</td>
</tr>
<tr>
<td>Deceleration</td>
<td>ErrorID</td>
</tr>
<tr>
<td>Jerk</td>
<td>SyncDistance</td>
</tr>
<tr>
<td>BufferMode</td>
<td></td>
</tr>
<tr>
<td>SyncMode</td>
<td></td>
</tr>
</tbody>
</table>

Function Description

The **MC_GearInPos** and **MCP_GearInPos** blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the **MCP** version of the block.

The functional block **MC_GearInPos** engages a synchronized motion of master and slave axes in such a way that the ratio of velocities of both axes is maintained at a constant value. Compared to **MC_GearIn**, also the master to slave position ratio is determined in a given reference point, i.e. following relation holds:

\[
\frac{\text{SlavePosition} - \text{SlaveSyncPosition}}{\text{MasterPosition} - \text{MasterSyncPosition}} = \frac{\text{RatioNumerator}}{\text{RatioDenominator}}.
\]

In case that the slave position does not fulfill this condition of synchronicity at the moment of block activation (i.e. in an instant of positive edge of Execute input and after execution of previous commands in buffered mode), synchronization procedure is
started and indicated by output StartSync. During this procedure, proper slave trajectory which results in smooth synchronization of both axes is generated with respect to actual master motion and slave limits for Velocity, Acceleration, Deceleration and Jerk (these limits are not applied from the moment of successful synchronization). Parameter setting 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 \(\text{MasterSyncPosition} \pm \text{MasterStartDistance}\).

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 \(\text{SlaveSyncPosition}\) approximately at the moment when the master position enters the range for synchronization given by \(\text{MasterSyncPosition}\) and \(\text{MasterStartDistance}\). Violation of this rule can lead to unpredictable behaviour of the slave axis during the synchronization or to an overrun of the specified limits for slave axis. However, the motion of both axes is usually well defined and predictable in standard applications and correct synchronization can be performed easily by proper configuration of motion commands and functional block parameters.

Inputs

- \text{uMaster} \quad \text{Master axis reference}
- \text{uSlave} \quad \text{Slave axis reference}
- \text{Execute} \quad \text{The block is activated on rising edge}
- \text{RatioNumerator} \quad \text{Gear ratio Numerator}
- \text{RatioDenominator} \quad \text{Gear ratio Denominator}
- \text{MasterSyncPosition} \quad \text{Master position for synchronization}
- \text{SlaveSyncPosition} \quad \text{Slave position for synchronization}
- \text{MasterStartDistance} \quad \text{Master distance for starting gear in procedure}
- \text{Velocity} \quad \text{Maximal allowed velocity [unit/s]}
- \text{Acceleration} \quad \text{Maximal allowed acceleration [unit/s²]}
- \text{Deceleration} \quad \text{Maximal allowed deceleration [unit/s²]}
- \text{Jerk} \quad \text{Maximal allowed jerk [unit/s³]}
BufferMode Buffering mode
1 ..... Aborting (start immediately)
2 ..... Buffered (start after finish of previous motion)
3 ..... Blending low (start after finishing the previous motion,
previous motion finishes with the lowest velocity of both
commands)
4 ..... Blending high (start after finishing the previous motion,
previous motion finishes with the lowest velocity of both
commands)
5 ..... Blending previous (start after finishing the previous
motion, previous motion finishes with its final velocity)
6 ..... Blending next (start after finishing the previous motion,
previous motion finishes with the starting velocity of the
next block)

SyncMode Synchronization mode (cyclic axes only)
1 ..... CatchUp
2 ..... Shortest
3 ..... SlowDown

Outputs

yMaster Master axis reference
ySlave Slave axis reference
StartSync Commanded gearing starts
InSync Slave axis reached the cam profile
CommandAborted Algorithm was aborted
Busy Algorithm not finished yet
Active The block is controlling the axis
Error Error occurred
ErrorID Error code
i ..... REX general error
SyncDistance Position deviation of the slave axis from synchronized position
**MC_GearOut – Disengage the master/slave velocity ratio**

**Block Symbol**

<table>
<thead>
<tr>
<th>uSlave</th>
<th>ySlave</th>
<th>Execute</th>
<th>Busy</th>
<th>Error</th>
<th>ErrorID</th>
</tr>
</thead>
</table>

**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**

- **uSlave** Slave axis reference  
- **Execute** The block is activated on rising edge

**Outputs**

- **ySlave** Slave axis reference  
- **Done** Algorithm finished  
- **Busy** Algorithm not finished yet  
- **Error** Error occurred  
- **ErrorID** Error code  
  
  *i ..... REX general error*
Example

```
velocity2 = 30
ratio_numerator1 = 2
ratio_denominator1 = 1
position2 = 100
mode2 = 2
mode1 = 2
jerk3 = 0
jerk1 = 0
execute3
execute2
execute1
direction2 = 1
deceleration3 = 150
deceleration2 = 20
deceleration1 = 10
```

```
MC_Stop_block 4
```

```
MC_MoveAbsolute − block 2
```

```
MC_GearOut − block 3
```

```
MC_GearIn − block 1
```
**MC_PhasingAbsolute, MCP_PhasingAbsolute – Phase shift in synchronized motion (absolute coordinates)**

**Function Description**

*The MC_PhasingAbsolute and MCP_PhasingAbsolute blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_version of the block.*

The MC_PhasingAbsolute block introduces an additional phase shift in master-slave relation defined by an electronic cam (MC_CamIn) or electronic gear (MC_GearIn). The functionality of this command is very similar to MC_MoveSuperimposed (additive motion from 0 to PhaseShift position with respect to maximum velocity acceleration and jerk). The only difference is that the additive position/velocity/acceleration is added to master axis reference position in the functional dependence defined by a cam or gear ratio for the computation of slave motion instead of its direct summation with master axis movement. The absolute value of final phase shift is specified by PhaseShift parameter.

Note: The motion command is analogous to rotation of a mechanical cam by angle PhaseShift.

**Inputs**

- **uMaster** Master axis reference
- **uSlave** Slave axis reference
- **Execute** The block is activated on rising edge
- **PhaseShift** Requested phase shift (distance on master axis) for cam

**Licence:** MOTION CONTROL
CHAPTER 17. MC_MULTI – MOTION CONTROL - MULTI AXIS BLOCKS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Velocity</td>
<td>Maximal allowed velocity [unit/s]</td>
<td>double</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s²]</td>
<td>double</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
<td>double</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
<td>double</td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>1 . . . . . . Aborting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 . . . . . . Buffered</td>
<td></td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yMaster</td>
<td>Master axis reference</td>
<td>reference</td>
</tr>
<tr>
<td>ySlave</td>
<td>Slave axis reference</td>
<td>reference</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>bool</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
<td>bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td></td>
<td>i . . . . . . REX general error</td>
<td></td>
</tr>
</tbody>
</table>
Example

- Block 1 execute (GearIn)
- Block 2 execute (MoveAbsolute)
- Block 3 execute (PhasingAbsolute)

- Block 1 active (GearIn)
- Block 2 active (MoveAbsolute)
- Block 3 active (PhasingAbsolute)

- Acceleration axis 1 - master
- Acceleration axis 2 - slave

- Velocity axis 1 - master
- Velocity axis 2 - slave

- Position axis slave without phasing
- Position axis slave with phasing

- Position axis 1 - master
- Position axis 2 - slave
MC_PhasingRelative, MCP_PhasingRelative – Phase shift in synchronized motion (relative coordinates)

Function Description

The MC_PhasingRelative and MCP_PhasingRelative blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_version of the block.

The MC_PhasingRelative introduces an additional phase shift in master-slave relation defined by an electronic cam (MC_CamIn) or electronic gear (MC_GearIn). The functionality of this command is very similar to MC_MoveSuperimposed (additive motion from 0 to PhaseShift position with respect to maximum velocity acceleration and jerk). The only difference is that the additive position/velocity/acceleration is added to master axis reference position in the functional dependence defined by a cam or gear ratio for the computation of slave motion instead of its direct summation with master axis movement. The relative value of final phase shift with respect to previous value is specified by PhaseShift parameter. Note: The motion command is analogous to rotation of a mechanical cam by angle PhaseShift

Inputs

- **uMaster** Master axis reference
- **uSlave** Slave axis reference
- **Execute** The block is activated on rising edge
- **PhaseShift** Requested phase shift (distance on master axis) for cam

Licence: MOTION CONTROL
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Velocity</td>
<td>Maximal allowed velocity [unit/s]</td>
<td>double</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s²]</td>
<td>double</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
<td>double</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
<td>double</td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>1 ...... Aborting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ...... Buffered</td>
<td></td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yMaster</td>
<td>Master axis reference</td>
<td>reference</td>
</tr>
<tr>
<td>ySlave</td>
<td>Slave axis reference</td>
<td>reference</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>bool</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
<td>bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td></td>
<td>i ...... REX general error</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 18

MC_COORD – Motion control - coordinated movement blocks
MC_MoveLinearAbsolute – Linear move to position (absolute coordinates) .............................................. 509
MC_MoveLinearRelative – Linear move to position (relative to execution point) ............................................ 513
MC_MoveCircularAbsolute – Circular move to position (absolute coordinates) .............................................. 517
MC_MoveCircularRelative – Circular move to position (relative to execution point) ............................................ 521
MC_MoveDirectAbsolute – Direct move to position (absolute coordinates) .................................................. 525
MC_MoveDirectRelative – Direct move to position (relative to execution point) .................................................. 528
MC_MovePath – General spatial trajectory generation ................................................................. 531
MC_GroupSetOverride – Set group override factors ................................................................. 533
<table>
<thead>
<tr>
<th></th>
<th>Aborting</th>
<th>Buffered without Blending</th>
<th>Blending</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Trajectory of TCP</strong></td>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
<td><img src="image3.png" alt="Diagram" /></td>
</tr>
<tr>
<td><strong>Speed of TCP</strong></td>
<td><img src="image4.png" alt="Graph" /></td>
<td><img src="image5.png" alt="Graph" /></td>
<td><img src="image6.png" alt="Graph" /></td>
</tr>
</tbody>
</table>
RM_AxesGroup – Axes group for coordinated motion control

Block Symbol

Function Description

Note 1: Applicable for all non-administrative (moving) function blocks.
Note 2: In the states GroupErrorStop or GroupStopping, all Function Blocks can be called, although they will not be executed, except MC_GroupReset for GroupErrorStop and any occurring Error – they will generate the transition to GroupStandby or GroupErrorStop respectively.
Note 3: MC_GroupStop.DONE AND NOT MC_GroupStop.EXECUTE
Note 4: Transition is applicable if last axis is removed from the group
Note 5: Transition is applicable while group is not empty.
Note 6: MC_GroupDisable and MC_UngroupAllAxes can be issued in all states and will change the state to GroupDisabled.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>McsCount</td>
<td>Number of axis in MCS</td>
<td>↓1 ↑6 ⊙6 long</td>
</tr>
<tr>
<td>Acscount</td>
<td>Number of axis in ACS</td>
<td>↓1 ↑16 ⊙6 long</td>
</tr>
<tr>
<td>PosCount</td>
<td>Number of position axis</td>
<td>↓1 ↑6 ⊙3 long</td>
</tr>
<tr>
<td>Velocity</td>
<td>Maximal allowed velocity [unit/s]</td>
<td>double</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s²]</td>
<td>double</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
<td>double</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>refGroup</td>
<td>Axes group reference</td>
<td>reference</td>
</tr>
<tr>
<td>refPos</td>
<td>Position, velocity and acceleration vector</td>
<td>reference</td>
</tr>
<tr>
<td>iState</td>
<td>Group status</td>
<td>long</td>
</tr>
</tbody>
</table>

0 ...... Disabled
1 ...... Standby
2 ...... Homing
6 ...... Moving
7 ...... Stopping
8 ...... Error stop
ErrorID Error code  error
i ...... REX general error

The State Diagram of AxesGroup

[Diagram of state transitions and actions]

Note 1 and MC_GroupEnable

Note 2

MC_GroupHalt

Note 1

MC_GroupStop

Note 2

MC_GroupReset

Note 4
MC_AddAxisToGroup
MC_RemoveAxisFromGroup
MC_UngroupAllAxes

Note 5
MC_AssAxisToGroup
MC_RemoveAxisFromGroup

RM_AxesGroup

axes_group_reference

uVec

y1

y2

y3

y4

y5

y6

y7

y8

VTOR1

VTOR2
## CHAPTER 18. MC_COORD – MOTION CONTROL - COORDINATED MOVEMENT BLOCKS

Adding particular axis to axesgroup
Implementation of particular single axis
Implementation of axes group

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vel4</td>
<td>[actual_velocity4]</td>
</tr>
<tr>
<td>vel3</td>
<td>[actual_velocity3]</td>
</tr>
<tr>
<td>vel2</td>
<td>[actual_velocity2]</td>
</tr>
<tr>
<td>vel1</td>
<td>[actual_velocity1]</td>
</tr>
<tr>
<td>pos4</td>
<td>[actual_position4]</td>
</tr>
<tr>
<td>pos3</td>
<td>[actual_position3]</td>
</tr>
<tr>
<td>pos2</td>
<td>[actual_position2]</td>
</tr>
<tr>
<td>pos1</td>
<td>[actual_position1]</td>
</tr>
<tr>
<td>commanded_vel4</td>
<td>[commanded_velocity4]</td>
</tr>
<tr>
<td>commanded_vel3</td>
<td>[commanded_velocity3]</td>
</tr>
<tr>
<td>commanded_vel2</td>
<td>[commanded_velocity2]</td>
</tr>
<tr>
<td>commanded_vel1</td>
<td>[commanded_velocity1]</td>
</tr>
<tr>
<td>commanded_tor4</td>
<td>[commanded_torque4]</td>
</tr>
<tr>
<td>commanded_tor3</td>
<td>[commanded_torque3]</td>
</tr>
<tr>
<td>commanded_tor2</td>
<td>[commanded_torque2]</td>
</tr>
<tr>
<td>commanded_tor1</td>
<td>[commanded_torque1]</td>
</tr>
<tr>
<td>commanded_pos4</td>
<td>[commanded_position4]</td>
</tr>
<tr>
<td>commanded_pos3</td>
<td>[commanded_position3]</td>
</tr>
<tr>
<td>commanded_pos2</td>
<td>[commanded_position2]</td>
</tr>
<tr>
<td>commanded_pos1</td>
<td>[commanded_position1]</td>
</tr>
</tbody>
</table>

## Mc_Power4
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Enable</td>
</tr>
<tr>
<td>yAxis</td>
<td>Status</td>
</tr>
<tr>
<td>Busy</td>
<td>Active</td>
</tr>
<tr>
<td>Error</td>
<td>ErrorID</td>
</tr>
</tbody>
</table>

## MC_Power3
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Enable</td>
</tr>
<tr>
<td>yAxis</td>
<td>Status</td>
</tr>
<tr>
<td>Busy</td>
<td>Active</td>
</tr>
<tr>
<td>Error</td>
<td>ErrorID</td>
</tr>
</tbody>
</table>

## MC_Power2
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Enable</td>
</tr>
<tr>
<td>yAxis</td>
<td>Status</td>
</tr>
<tr>
<td>Busy</td>
<td>Active</td>
</tr>
<tr>
<td>Error</td>
<td>ErrorID</td>
</tr>
</tbody>
</table>

## MC_Power1
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Enable</td>
</tr>
<tr>
<td>yAxis</td>
<td>Status</td>
</tr>
<tr>
<td>Busy</td>
<td>Active</td>
</tr>
<tr>
<td>Error</td>
<td>ErrorID</td>
</tr>
</tbody>
</table>

## MC_GroupEnable_1234
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Execute</td>
</tr>
<tr>
<td>yAxis</td>
<td>Status</td>
</tr>
<tr>
<td>Busy</td>
<td>Active</td>
</tr>
<tr>
<td>Error</td>
<td>ErrorID</td>
</tr>
</tbody>
</table>

## MC_AddAxisToGroup_O4
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Execute</td>
</tr>
<tr>
<td>yAxis</td>
<td>Status</td>
</tr>
<tr>
<td>Busy</td>
<td>Active</td>
</tr>
<tr>
<td>Error</td>
<td>ErrorID</td>
</tr>
</tbody>
</table>

## MCP_SetKinTransform_Agebot
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Execute</td>
</tr>
<tr>
<td>yAxis</td>
<td>Status</td>
</tr>
<tr>
<td>Busy</td>
<td>Active</td>
</tr>
<tr>
<td>Error</td>
<td>ErrorID</td>
</tr>
</tbody>
</table>
RM_Feed – * MC Feeder ??

Block Symbol

License: COORDINATED MOTION CONTROL

Function Description

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

Inputs

- **uAxesGroup** Axes group reference
- **Execute** The block is activated on rising edge

Parameters

- **Filename** 0
- **VelFactor** 0 \( \downarrow 0.01 \uparrow 100.0 \circ 1.0 \) double
- **Relative** 0 bool
- **CoordSystem** 0
- **BufferMode** 0 \( \downarrow 1 \uparrow 3 \circ 2 \) long
- **TransitionMode** 0 \( \downarrow 0 \uparrow 15 \circ 1 \) long
- **TransitionParameter** 0

Outputs

- **yAxesGroup** Axes group reference
- **Done** Algorithm finished
- **CommandAborted** Algorithm was aborted
- **Busy** Algorithm not finished yet
- **Active** The block is controlling the axis
- **Error** Error occurred
- **ErrorID** Error code
- **Aux** 0 double
RM_Gcode – * CNC motion control

Block Symbol

Licence: COORDINATED MOTION CONTROL

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

Inputs
- uAxesGroup Axes group reference
- Execute The block is activated on rising edge
- BlockSkip MILAN

Parameters
- BaseDir Directory of the G-code files
- MainFile Source file number
- CoordSystem 0
- BufferMode Buffering mode
- TransitionMode Transition mode in blending mode

```plaintext
<table>
<thead>
<tr>
<th>BufferMode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aborting</td>
</tr>
<tr>
<td>2</td>
<td>Buffered</td>
</tr>
<tr>
<td>3</td>
<td>Blending low</td>
</tr>
<tr>
<td>4</td>
<td>Blending high</td>
</tr>
<tr>
<td>5</td>
<td>Blending previous</td>
</tr>
<tr>
<td>6</td>
<td>Blending next</td>
</tr>
<tr>
<td>7</td>
<td>Blending previous</td>
</tr>
<tr>
<td>8</td>
<td>Blending next</td>
</tr>
<tr>
<td>9</td>
<td>Blending previous</td>
</tr>
<tr>
<td>10</td>
<td>Blending next</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TransitionMode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TMNone</td>
</tr>
<tr>
<td>2</td>
<td>TMStartVelocity</td>
</tr>
<tr>
<td>3</td>
<td>TMConstantVelocity</td>
</tr>
<tr>
<td>4</td>
<td>TMCornerDistance</td>
</tr>
<tr>
<td>5</td>
<td>TMMaxCornerDeviation</td>
</tr>
<tr>
<td>6</td>
<td>Smooth</td>
</tr>
</tbody>
</table>
```
**TransitionParameter**  Parametr for transition (depends on transition mode)  double

**workOffsets**  Sets with initial coordinate  double  
⊙[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

**toolOffsets**  Sets of tool offset  double  
⊙[0 0 0]

**cutterOffsets**  Tool radii  double  
⊙[0 0 0]

**Outputs**

**yAxesGroup**  Axes group reference  reference

**Done**  Algorithm finished  bool

**CommandAborted**  Algorithm was aborted  bool

**Busy**  Algorithm not finished yet  bool

**Active**  The block is controlling the axis  bool

**Error**  Error occurred  bool

**ErrorID**  Error code  error

i .....  REX general error

**Cooling**  Cooling  bool

**MoveType**  Command execution  long

**ExecutingLine**  Current line of G-code  long

**SpindleSpeed**  Spindle speed  long
MC_AddAxisToGroup – Adds one axis to a group

Block Symbol

| uAxesGroup | yAxesGroup |
| yAxesGroup | yAxesGroup |
| uAxis      | yAxis      |
| Execute    | Error      |
| IdentInGroup | ErrorID |
| MC_AddAxisToGroup |

Function Description

The function block MC_AddAxisToGroup adds one uAxis to the group in a structure uAxesGroup. Axes Group is implemented by the function block RM_AxesGroup. The input uAxis must be defined by the function block RM_Axis from the MC_SINGLE library.

Note 1: Every IdentInGroup is unique and can be used only for one time otherwise the error is set.

Inputs

| uAxesGroup | Axes group reference |
| uAxis      | Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed) |
| Execute    | The block is activated on rising edge |
| IdentInGroup | The order of axes in the group (0 = first unassigned) |

Outputs

| yAxesGroup | Axes group reference |
| yAxis      | Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed) |
| Done      | Algorithm finished |
| Busy      | Algorithm not finished yet |
| Error      | Error occurred |
| ErrorID    | Error code |
|           | i .... REX general error |
MC_UngroupAllAxes – Removes all axes from the group

Block Symbol

Function Description

The function block MC_UngroupAllAxes removes all axes from the group uAxesGroup. After finalization the state is changed to "GroupDisabled".

Note 1: If the function block is execute in the group state "GroupDisabled", "Group-StandBy" or "GroupErrorStop" the error is set and the block is not execute.

Inputs

- **uAxesGroup** Axes group reference
- **Execute** The block is activated on rising edge

Outputs

- **yAxesGroup** Axes group reference
- **Done** Algorithm finished
- **Busy** Algorithm not finished yet
- **Error** Error occurred
- **ErrorID** Error code
  - i ..... REX general error
  - error

Licence: COORDINATED MOTION CONTROL
MC_GroupEnable – Changes the state of a group to GroupEnable

Function Description

The function block MC_GroupEnable changes the state for the group uAxesGroup from "GroupDisabled" to "GroupStandby".

Inputs

- **uAxesGroup**: Axes group reference
- **Execute**: The block is activated on rising edge

Outputs

- **yAxesGroup**: Axes group reference
- **Done**: Algorithm finished
- **Busy**: Algorithm not finished yet
- **Error**: Error occurred
- **ErrorID**: Error code
  - i ..... REX general error

Licence: COORDINATED MOTION CONTROL
**MC_GroupDisable** – Changes the state of a group to GroupDisabled

**Block Symbol**

- **uAxesGroup**: Axes group reference
- **Execute**: The block is activated on rising edge

**Outputs**

- **yAxesGroup**: Axes group reference
- **Done**: Algorithm finished
- **Busy**: Algorithm not finished yet
- **Error**: Error occurred
- **ErrorID**: Error code
  - i ..... REX general error

**Inputs**

- **uAxesGroup**: Axes group reference
- **Execute**: The block is activated on rising edge

**Function Description**

The function block **MC_GroupDisable** changes the state for the group **uAxesGroup** to "GroupDisabled". If the axes are not standing still while issuing this command the state of the group is changed to "Stopping". It is mean stopping with the maximal allowed deceleration. When stopping is done the state of the group is changed to "GroupDisabled".

**Inputs**

- **uAxesGroup**: Axes group reference
- **Execute**: The block is activated on rising edge

**Outputs**

- **yAxesGroup**: Axes group reference
- **Done**: Algorithm finished
- **Busy**: Algorithm not finished yet
- **Error**: Error occurred
- **ErrorID**: Error code
  - i ..... REX general error

**Licence: COORDINATED MOTION CONTROL**
MC_SetCartesianTransform – Sets Cartesian transformation

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 |
| TransX     | X-component of translation vector | double |
| TransY     | Y-component of translation vector | double |
| TransZ     | Z-component of translation vector | double |
| RotAngle1  | Rotation angle component | double |
| RotAngle2  | Rotation angle component | double |
| RotAngle3  | Rotation angle component | double |
| Relative   | Mode of position inputs | bool |

Outputs

| yAxesGroup | Axes group reference  | reference |
| Done       | Algorithm finished   | bool |
| Busy       | Algorithm not finished yet | bool |
| Error      | Error occurred       | bool |
| ErrorID    | Error code           | error |
| i ....     | REX general error    |
MC_ReadCartesianTransform – Reads the parameter of the cartesian transformation

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

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Axes group reference</td>
<td>reference</td>
</tr>
<tr>
<td>Enable</td>
<td>Block function is enabled</td>
<td>bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxesGroup</td>
<td>Axes group reference</td>
<td>reference</td>
</tr>
<tr>
<td>Valid</td>
<td>Output value is valid</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>TransX</td>
<td>X-component of translation vector</td>
<td>double</td>
</tr>
<tr>
<td>TransY</td>
<td>Y-component of translation vector</td>
<td>double</td>
</tr>
<tr>
<td>TransZ</td>
<td>Z-component of translation vector</td>
<td>double</td>
</tr>
<tr>
<td>RotAngle1</td>
<td>Rotation angle component</td>
<td>double</td>
</tr>
<tr>
<td>RotAngle2</td>
<td>Rotation angle component</td>
<td>double</td>
</tr>
<tr>
<td>RotAngle3</td>
<td>Rotation angle component</td>
<td>double</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
<td>error</td>
</tr>
</tbody>
</table>

_..._ i ..... REX general error
MC_GroupSetPosition, MCP_GroupSetPosition – Sets the position of all axes in a group

Function Description

The MC_GroupSetPosition and MCP_GroupSetPosition blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The function block MC_GroupSetPosition sets the position of all axes in the group uAxesGroup without moving the axes. The new coordinates are described by the input Position. With the coordinate system input CoordSystem the according coordinate system is selected. The function block MC_GroupSetPosition shifts position of the addressed coordinate system and affect the higher level coordinate systems (so if ACS selected, MCS and PCS are affected).

Inputs

- **uAxesGroup** Axes group reference
- **Execute** The block is activated on rising edge
- **Position** Array of coordinates (positions and orientations)
- **Relative** Mode of position inputs
  - off ... absolute
  - on ..... relative
- **CoordSystem** Reference to the coordinate system used
  - 1 ..... ACS
  - 2 ..... MCS
  - 3 ..... PCS

Outputs

- **yAxesGroup** Axes group reference
- **Done** Algorithm finished
- **Busy** Algorithm not finished yet
<table>
<thead>
<tr>
<th><strong>CommandAborted</strong></th>
<th>Algorithm was aborted</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Error</strong></td>
<td>Error occurred</td>
</tr>
<tr>
<td><strong>ErrorID</strong></td>
<td>Error code</td>
</tr>
<tr>
<td></td>
<td>i . . . . . . REX general error</td>
</tr>
</tbody>
</table>
MC_GroupReadActualPosition – **Read actual position in the selected coordinate system**

**Function Description**

The function block `MC_GroupReadActualPosition` returns the actual position in the selected coordinate system of an axes group. The position is valid only if the output `Valid` is true which is achieved by setting the input `Enable` on true.

**Inputs**

- `uAxesGroup` Axes group reference
- `Enable` Block function is enabled
- `CoordSystem` Reference to the coordinate system used
  - 1 . . . ACS
  - 2 . . . . MCS
  - 3 . . . . PCS

**Outputs**

- `yAxesGroup` Axes group reference
- `Valid` Output value is valid
- `Busy` Algorithm not finished yet
- `Error` Error occurred
- `ErrorID` Error code
  - i . . . REX general error
- `Position` xxx
MC_GroupReadActualVelocity – Read actual velocity in the selected coordinate system

Block Symbol

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>yAxesGroup</td>
<td>Valid</td>
</tr>
<tr>
<td>Enable</td>
<td>Error</td>
<td>ErrorID</td>
</tr>
<tr>
<td>CoordSystem</td>
<td>Velocity</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**: Error code [error]
  
  1 ..... REX general error
- **Velocity**: xxx [reference]
**MC_GroupReadActualAcceleration** – Read actual acceleration in the selected coordinate system

**Block Symbol**

<table>
<thead>
<tr>
<th>MC_GroupReadActualAcceleration</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
</tr>
<tr>
<td>Enable</td>
</tr>
<tr>
<td>CoordSystem</td>
</tr>
<tr>
<td>yAxesGroup</td>
</tr>
</tbody>
</table>

**Function Description**

The function block **MC_GroupReadActualAcceleration** returns the actual velocity in the selected coordinate system of an axes group. The position is valid only if the output **Valid** is true which is achieved by setting the input **Enable** on true.

**Inputs**

- **uAxesGroup** Axes group reference
- **Enable** Block function is enabled
- **CoordSystem** Reference to the coordinate system used
  1. ACS
  2. MCS
  3. PCS

**Outputs**

- **yAxesGroup** Axes group reference
- **Valid** Output value is valid
- **Busy** Algorithm not finished yet
- **Error** Error occurred
- **ErrorID** Error code
  1. REX general error
- **Acceleration** Acceleration

---

**License:** COORDINATED MOTION CONTROL
MC_GroupStop – Stopping a group movement

Function Description

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

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Axes group reference</td>
<td>reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>bool</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
<td>double</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
<td>double</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxesGroup</td>
<td>Axes group reference</td>
<td>reference</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>bool</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
<td>bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td></td>
<td>i ...... REX general error</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 18. MC_COORD – MOTION CONTROL - COORDINATED MOVEMENT BLOCKS

Execute - MC_MoveLinearAbsolute

Done - MC_MoveLinearAbsolute

Error - MC_MoveLinearAbsolute

Execute - MC_GroupStop

Done - MC_GroupStop

Velocity AxesGroup

Position AxesGroup

Time [s]

Position [rad]

Velocity [rad/s]
**MC_GroupHalt** — Stopping a group movement (interruptible)

**Block Symbol**

![Block Symbol Diagram](image)

**Function Description**

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

**Inputs**

- **uAxesGroup**: Axes group reference
- **Execute**: The block is activated on rising edge
- **Deceleration**: Maximal allowed deceleration [unit/s²]
- **Jerk**: Maximal allowed jerk [unit/s³]

**Outputs**

- **yAxesGroup**: Axes group reference
- **Done**: Algorithm finished
- **CommandAborted**: Algorithm was aborted
- **Busy**: Algorithm not finished yet
- **Active**: The block is controlling the axis
- **Error**: Error occurred
- **ErrorID**: Error code
  
  i . . . . . REX general error
MC_MoveLinearAbsolute - Function Block 1

- uAxesGroup
- Execute
- Position
- Velocity
- Acceleration
- CoordSystem
- BufferMode
- TransitionMode
- TransitionParameter
- yAxesGroup
- Done
- CommandAborted
- Busy
- Active
- Error
- ErrorID

MC_GroupHalt

- uAxesGroup
- Execute
- Deceleration
- Jerk
- yAxesGroup
- Done
- CommandAborted
- Busy
- Active
- Error
- ErrorID

- velocity2
- 0.4
- velocity1
- 0.4
- transition_parameter2
- 1
- transition_parameter1
- 1
- transition_mode2
- 1
- transition_mode1
- reference_to_axesgroup

RTOV1

RTOV2
Execute − MC_MoveLinearAbsolute — Function Block 1

Done − MC_MoveLinearAbsolute — Function Block 1

Execute − MC_MoveLinearAbsolute — Function Block 2

Done − MC_MoveLinearAbsolute — Function Block 2

Execute − MC_GroupHalt

Execute − MC_GroupHalt

Velocity AxesGroup

Position AxesGroup
MC_GroupInterrupt, MCP_GroupInterrupt – Read a group interrupt

Block Symbols

Function Description

The MC_GroupInterrupt and MCP_GroupInterrupt blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP version of the block.

The function block MC_GroupInterrupt interrupts the on-going motion and stops the group from moving, however does not abort the interrupted motion (meaning that at the interrupted FB the output CommandAborted will not be Set, Busy is still high and Active is reset). It stores all relevant track or path information internally at the moment it becomes active. The uAxesGroup stays in the original state even if the velocity zero is reached and the Done output is set.

Note 1: This function block is complementary to the function block MC_GroupContinue which execution the uAxesGroup state is reset to the original state (before MC_GroupInterrupt execution)

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Axes group reference</td>
<td>reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>bool</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
<td>double</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
<td>double</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxesGroup</td>
<td>Axes group reference</td>
<td>reference</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
<td>bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------</td>
</tr>
<tr>
<td>i</td>
<td>REX general error</td>
<td></td>
</tr>
</tbody>
</table>
MC_GroupContinue – Continuation of interrupted movement

Block Symbol

<table>
<thead>
<tr>
<th>yAxesGroup</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td>Busy</td>
<td></td>
</tr>
<tr>
<td>CommandAborted</td>
<td></td>
</tr>
<tr>
<td>Error</td>
<td></td>
</tr>
<tr>
<td>ErrorID</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MC_GroupContinue</td>
</tr>
</tbody>
</table>

Licence: COORDINATED MOTION CONTROL

Function Description

The function block MC_GroupContinue transfers the program back to the situation at issuing MC_GroupInterrupt. It uses internally the data set as stored at issuing MC_GroupInterrupt, and at the end (output Done set) transfer the control on the group back to the original FB doing the movements on the axes group, meaning also that at the originally interrupted FB the output Busy is still high and the output Active is set again.

Inputs

- **uAxesGroup**: Axes group reference
- **Execute**: The block is activated on rising edge

Outputs

- **yAxesGroup**: Axes group reference
- **Done**: Algorithm finished
- **Busy**: Algorithm not finished yet
- **CommandAborted**: Algorithm was aborted
- **Error**: Error occurred
- **ErrorID**: Error code
  - i: REX general error
MC_GroupReadStatus – Read a group status

Function Description

The function block MC_GroupReadStatus returns the status of the uAxesGroup. The status is valid only if the output Valid is true which is achieved by setting the input Enable on true.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Axes group reference</td>
<td>reference</td>
</tr>
<tr>
<td>Enable</td>
<td>Block function is enabled</td>
<td>bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxesGroup</td>
<td>Axes group reference</td>
<td>reference</td>
</tr>
<tr>
<td>Valid</td>
<td>Output value is valid</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>GroupMoving</td>
<td>State GroupMoving</td>
<td>bool</td>
</tr>
<tr>
<td>GroupHoming</td>
<td>State GroupHoming</td>
<td>bool</td>
</tr>
<tr>
<td>GroupErrorStop</td>
<td>State ErrorStop</td>
<td>bool</td>
</tr>
<tr>
<td>GroupStandby</td>
<td>State Standby</td>
<td>bool</td>
</tr>
<tr>
<td>GroupStopping</td>
<td>State Stopping</td>
<td>bool</td>
</tr>
<tr>
<td>GroupDisabled</td>
<td>State Disabled</td>
<td>bool</td>
</tr>
<tr>
<td>ConstantVelocity</td>
<td>Constant velocity motion</td>
<td>bool</td>
</tr>
<tr>
<td>Accelerating</td>
<td>Accelerating</td>
<td>bool</td>
</tr>
<tr>
<td>Decelerating</td>
<td>Decelerating</td>
<td>bool</td>
</tr>
<tr>
<td>InPosition</td>
<td>Symptom achieve the desired position</td>
<td>bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>------------------</td>
</tr>
<tr>
<td>i</td>
<td>.....</td>
<td>REX general error</td>
</tr>
</tbody>
</table>
MC_GroupReadError – Read a group error

**Function Description**

The function block `MC_GroupReadError` describes general error on the `uAxesGroup` which is not relating to the function blocks. If the output `GroupErrorID` is equal to 0 there is no error on the axes group. The actual error code `GroupErrorID` is valid only if the output `Valid` is true which is achieved by setting the input `Enable` on true.

Note 1: This function block is implemented because of compatibility with the PLCopen norm. The same error value is on the output `ErrorID` of the function block `RM_AxesGroup`.

**Inputs**

- `uAxesGroup` Axes group reference
- `Enable` Block function is enabled

**Outputs**

- `yAxesGroup` Axes group reference
- `Valid` Output value is valid
- `Busy` Algorithm not finished yet
- `Error` Error occurred
- `ErrorID` Error code
  - `i` ..... REX general error
- `GroupErrorID` Error code
  - `i` ..... REX general error
MC_GroupReset – Reset axes errors

Function Description

The function block `MC_GroupReset` makes the transition from the state "GroupErrorStop" to "GroupStandBy" by resetting all internal group-related errors. This function block also resets all axes in this group like the function block `MC_Reset` from the MC_SINGLE library.

Inputs

- **uAxesGroup**: Axes group reference
- **Execute**: The block is activated on rising edge

Outputs

- **yAxesGroup**: Axes group reference
- **Done**: Algorithm finished
- **Busy**: Algorithm not finished yet
- **Error**: Error occurred
- **ErrorID**: Error code
  
i ..... REX general error
MC_MoveLinearAbsolute – Linear move to position (absolute coordinates)

Block Symbol

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uAxesGroup</code></td>
<td>Axes group reference</td>
</tr>
<tr>
<td><code>Execute</code></td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td><code>Position</code></td>
<td>Array of coordinates (positions and orientations)</td>
</tr>
<tr>
<td><code>Velocity</code></td>
<td>Maximal allowed velocity [unit/s]</td>
</tr>
<tr>
<td><code>Acceleration</code></td>
<td>Maximal allowed acceleration [unit/s²]</td>
</tr>
<tr>
<td><code>Jerk</code></td>
<td>Maximal allowed jerk [unit/s³]</td>
</tr>
<tr>
<td><code>CoordSystem</code></td>
<td>Reference to the coordinate system used</td>
</tr>
<tr>
<td><code>BufferMode</code></td>
<td></td>
</tr>
<tr>
<td><code>TransitionMode</code></td>
<td></td>
</tr>
<tr>
<td><code>TransitionParameter</code></td>
<td></td>
</tr>
<tr>
<td><code>yAxesGroup</code></td>
<td></td>
</tr>
<tr>
<td><code>Done</code></td>
<td></td>
</tr>
<tr>
<td><code>CommandAborted</code></td>
<td></td>
</tr>
<tr>
<td><code>Busy</code></td>
<td></td>
</tr>
<tr>
<td><code>Active</code></td>
<td></td>
</tr>
<tr>
<td><code>Error</code></td>
<td></td>
</tr>
<tr>
<td><code>ErrorID</code></td>
<td></td>
</tr>
</tbody>
</table>

Function Description

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

Inputs

- `uAxesGroup` Axes group reference
- `Execute` The block is activated on rising edge
- `Position` Array of coordinates (positions and orientations)
- `Velocity` Maximal allowed velocity [unit/s]
- `Acceleration` Maximal allowed acceleration [unit/s²]
- `Jerk` Maximal allowed jerk [unit/s³]
- `CoordSystem` Reference to the coordinate system used
  - 1 . . . . ACS
  - 2 . . . . MCS
  - 3 . . . . PCS

Licence: COORDINATED MOTION CONTROL
BufferMode Buffering mode  
1 . . . . . Aborting  
2 . . . . . Buffered  
3 . . . . . Blending low  
4 . . . . . Blending high  
5 . . . . . Blending previous  
6 . . . . . Blending next  
TransitionMode Transition mode in blending mode  
1 . . . . . TMNone  
2 . . . . . TMStartVelocity  
3 . . . . . TMConstantVelocity  
4 . . . . . TMCornerDistance  
5 . . . . . TMMaxCornerDeviation  
11 . . . . Smooth  
TransitionParameter Parametr for transition (depends on transition mode)  

Outputs  

yAxesGroup Axes group reference  
Done Algorithm finished  
CommandAborted Algorithm was aborted  
Busy Algorithm not finished yet  
Active The block is controlling the axis  
Error Error occurred  
ErrorID Error code  
i . . . . REX general error
Sequence of two complete motions (Done>Execute)
**Chapter 18. MC_COORD – MOTION CONTROL - COORDINATED MOVEMENT BLOCKS**

- **Execute** - MC_MoveLinearAbsolute — Function Block 1
- **Active** - MC_MoveLinearAbsolute — Function Block 1
- **Done** - MC_MoveLinearAbsolute — Function Block 1

- **Execute** - MC_MoveLinearAbsolute — Function Block 2
- **Active** - MC_MoveLinearAbsolute — Function Block 2
- **Done** - MC_MoveLinearAbsolute — Function Block 2

**Velocity AxesGroup**

**Position AxesGroup**

- Commanded position x,y
- CAS [s]
- Position [rad]
MC_MoveLinearRelative – Linear move to position (relative to execution point)

**Function Description**

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

**Inputs**

- **uAxesGroup** Axes group reference
- **Execute** The block is activated on rising edge
- **Distance** Array of coordinates (relative distances and orientations)
- **Velocity** Maximal allowed velocity [unit/s]
- **Acceleration** Maximal allowed acceleration [unit/s^2]
- **Jerk** Maximal allowed jerk [unit/s^3]
- **CoordSystem** Reference to the coordinate system used
  - 1 . . . ACS
  - 2 . . . MCS
  - 3 . . . PCS
- **BufferMode** Buffering mode
  - 1 . . . Aborting
  - 2 . . . Buffered
  - 3 . . . Blending low
  - 4 . . . Blending high
  - 5 . . . Blending previous
  - 6 . . . Blending next

**Licence:** COORDINATED MOTION CONTROL
TransitionMode Transition mode in blending mode
1 ...... TMNone
2 ...... TMStartVelocity
3 ...... TMConstantVelocity
4 ...... TMCornerDistance
5 ...... TMMaxCornerDeviation
11 .... Smooth

TransitionParameter Parametr for transition (depends on transition mode)

Outputs

yAxesGroup Axes group reference
Done Algorithm finished
CommandAborted Algorithm was aborted
Busy Algorithm not finished yet
Active The block is controlling the axis
Error Error occurred
ErrorID Error code

i ...... REX general error
Sequence of two complete motions (Done>Execute)
MC_MoveCircularAbsolute – Circular move to position (absolute coordinates)

Function Description

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

Inputs

- **uAxesGroup** Axes group reference
- **Execute** The block is activated on rising edge

Diagram:

- **CircMode = BORDER**
  - Starting point
  - Border point
  - End point

- **CircMode = CENTER**
  - Center point
  - End point

- **CircMode = RADIUS**
  - Start point
  - Spearhead point
  - Length = Radius of the circle

Diagram labels:

- X-axis
- Y-axis

Diagram notes:

- CircMode = BORDER
- CircMode = CENTER
- CircMode = RADIUS
**CHAPTER 18. MC_COORD – MOTION CONTROL - COORDINATED MOVEMENT BLOCKS**

**CircMode**
Specifies the meaning of the input signals AuxPoint and CircDirection

```
1       BORDER
2       CENTER
3       RADIUS
```

**AuxPoint**
Next coordinates to define circle (depend on CircMode)

**EndPoint**
Target axes coordinates position

**PathChoice**
Choice of path

```
1       Clockwise
2       CounterClockwise
```

**Velocity**
Maximal allowed velocity [unit/s] double

**Acceleration**
Maximal allowed acceleration [unit/s²] double

**Jerk**
Maximal allowed jerk [unit/s³] double

**CoordSystem**
Reference to the coordinate system used

```
1       ACS
2       MCS
3       PCS
```

**BufferMode**
Buffering mode

```
1       Aborting
2       Buffered
3       Blending low
4       Blending high
5       Blending previous
6       Blending next
```

**TransitionMode**
Transition mode in blending mode

```
1       TMNone
2       TMStartVelocity
3       TMConstantVelocity
4       TMCornerDistance
5       TMMaxCornerDeviation
11      Smooth
```

**TransitionParameter**
Parametr for transition (depends on transition mode)

double

**Outputs**

**yAxesGroup**
Axes group reference

**Done**
Algorithm finished

**CommandAborted**
Algorithm was aborted

**Busy**
Algorithm not finished yet

**Active**
The block is controlling the axis

**Error**
Error occurred

**ErrorID**
Error code

```
i       REX general error
```
MC_MoveCircularAbsolute - Example
MC_MoveCircularRelative – Circular move to position (relative to execution point)

**Block Symbol**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Axes group reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>CircMode</td>
<td>Done</td>
</tr>
<tr>
<td>AuxPoint</td>
<td></td>
</tr>
<tr>
<td>EndPoint</td>
<td>CommandAborted</td>
</tr>
<tr>
<td>PathChoice</td>
<td></td>
</tr>
<tr>
<td>Velocity</td>
<td>Busy</td>
</tr>
<tr>
<td>Acceleration</td>
<td></td>
</tr>
<tr>
<td>Jerk</td>
<td>Active</td>
</tr>
<tr>
<td>CoordSystem</td>
<td></td>
</tr>
<tr>
<td>BufferMode</td>
<td>Error</td>
</tr>
<tr>
<td>TransitionMode</td>
<td></td>
</tr>
<tr>
<td>TransitionParameter</td>
<td>ErrorID</td>
</tr>
</tbody>
</table>

**Function Description**

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

**Inputs**

- **uAxesGroup** Axes group reference
- **Execute** The block is activated on rising edge

**Diagram**

- **CircMode = BORDER**
- **CircMode = CENTER**
- **CircMode = RADIUS**

**Legend**

- **Starting point**
- **Border point**
- **Center point**
- **Spearhead point**
- **Length = Radius of the circle**
## CHAPTER 18. MC_COORD – MOTION CONTROL - COORDINATED MOVEMENT BLOCKS

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CircMode</td>
<td>Specifies the meaning of the input signals AuxPoint and CircDirection</td>
<td>long</td>
</tr>
<tr>
<td>AuxPoint</td>
<td>Next coordinates to define circle (depend on CircMode)</td>
<td>reference</td>
</tr>
<tr>
<td>EndPoint</td>
<td>Target axes coordinates position</td>
<td>reference</td>
</tr>
<tr>
<td>PathChoice</td>
<td>Choice of path</td>
<td>long</td>
</tr>
<tr>
<td>Velocity</td>
<td>Maximal allowed velocity [unit/s]</td>
<td>double</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s²]</td>
<td>double</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
<td>double</td>
</tr>
<tr>
<td>CoordSystem</td>
<td>Reference to the coordinate system used</td>
<td>long</td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
<td>long</td>
</tr>
<tr>
<td>TransitionMode</td>
<td>Transition mode in blending mode</td>
<td>long</td>
</tr>
<tr>
<td>TransitionParameter</td>
<td>Parametr for transition (depends on transition mode)</td>
<td>double</td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>yAxesGroup</td>
<td>Axes group reference</td>
<td>reference</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>bool</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
<td>bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td></td>
<td>i ..... REX general error</td>
<td></td>
</tr>
</tbody>
</table>
MC_MoveCircularRelative - Example

- **Starting point 1**: AuxPoint1
- **EndPoint2**: EndPoint2
- **AuxPoint2**: AuxPoint2
- **AuxPoint1**: AuxPoint1
- **EndPoint1**: EndPoint1

- **Axes Group 1**
  - Velocity: 0.5
  - Transition Parameter: 1
  - Transition Mode: 1
  - Coord System: 2
  - Buffer Mode: 1
  - Acceleration: 0.5
  - RTOV1
    - uVec: u1, u2, u3, u4, u5, u6, u7, u8
    - yVec: RTOV3
  - PathChoice: 1
  - Cyclic Mode: CircMode1
  - Function Block 1

- **Axes Group 2**
  - Velocity: 0.5
  - Transition Parameter: 1
  - Transition Mode: 1
  - Coord System: 2
  - Buffer Mode: 1
  - Acceleration: 0.5
  - RTOV2
    - uVec: u1, u2, u3, u4, u5, u6, u7, u8
    - yVec: RTOV3
  - PathChoice: 1
  - Cyclic Mode: CircMode2
  - Function Block 2
MC_MoveDirectAbsolute – Direct move to position (absolute coordinates)

Block Symbol

<p align="center">uAxesGroup  yAxesGroup
Execute  Done
Position  CommandAborted
CoordSystem  Busy
BufferMode  Active
TransitionMode  Error
TransitionParameter  ErrorID
</p>

MC_MoveDirectAbsolute

Function Description

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

Inputs

| uAxesGroup | Axes group reference | reference |
| Execute | The block is activated on rising edge | bool |
| Position | Array of coordinates (positions and orientations) | reference |
| CoordSystem | Reference to the coordinate system used | long |
| BufferMode | Buffering mode | long |
| TransitionMode | Transition mode in blending mode | long |


| BufferMode | long |
| Aborting | 1 |
| Buffered | 2 |
| Blending low | 3 |
| Blending high | 4 |
| Blending previous | 5 |
| Blending next | 6 |

| TransitionMode | long |
| TMNone | 1 |
| TMStartVelocity | 2 |
| TMConstantVelocity | 3 |
| TMCornerDistance | 4 |
| TMMaxCornerDeviation | 5 |
| Smooth | 11 |
**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** Error code error
  
i ..... REX general error

---

**MC_MoveDirectAbsolute** - Example

Starting point

End point

---

Position

Position

CoordSystem 2

BufferMode 1

TransitionMode 1

TransitionParameter 1

RTOV1

orientation_of_effector

position x

position y

position z

-0.9

-0.6

-1

-1.5

yVec

yVec
MC_MoveDirectRelative – Direct move to position (relative to execution point)

Block Symbol

| uAxesGroup | Axes group reference |
| Execute | The block is activated on rising edge |
| Distance | Array of coordinates (relative distances and orientations) |
| CoordSystem | Reference to the coordinate system used |
| BufferMode | Buffering mode |
| TransitionMode | Transition mode in blending mode |
| TransitionParameter | Error ID |

Function Description

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

Inputs

| uAxesGroup | Axes group reference |
| Execute | The block is activated on rising edge |
| Distance | Array of coordinates (relative distances and orientations) |
| CoordSystem | Reference to the coordinate system used |
| BufferMode | Buffering mode |
| TransitionMode | Transition mode in blending mode |

| BufferMode | long |
| Aborting | 1 |
| Buffered | 2 |
| Blending low | 3 |
| Blending high | 4 |
| Blending previous | 5 |
| Blending next | 6 |

| TransitionMode | long |
| TMNone | 1 |
| TMStartVelocity | 2 |
| TMConstantVelocity | 3 |
| TMCornerDistance | 4 |
| TMMaxCornerDeviation | 5 |
| Smooth | 11 |
**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**  Error code  error

---

**MC_MoveDirectRelative - Example**

- Starting point
- Endpoint
- Distance
**MC_MovePath – General spatial trajectory generation**

**Block Symbol**

![Block Symbol](image)

**Function Description**

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

**Inputs**

- **uAxesGroup** Axes group reference
- **Execute** The block is activated on rising edge
- **TotalTime** Time [s] for whole move
- **RampTime** Time [s] for acceleration/deceleration
- **CoordSystem** Reference to the coordinate system used
  1 ..... ACS
  2 ..... MCS
  3 ..... PCS
- **BufferMode** Buffering mode
  1 ..... Aborting
  2 ..... Buffered
  3 ..... Blending low
  4 ..... Blending high
  5 ..... Blending previous
  6 ..... Blending next
- **RampIn** Reference to the coordinate system used
  1 ..... ACS
  2 ..... MCS
  3 ..... PCS
- **Done** End of motion
- **CommandAborted** Command is aborted
- **Busy** Block is busy
- **Active** Block is active
- **Error** Block is in error
- **ErrorID** Error code
CHAPTER 18. MC_COORD – MOTION CONTROL - COORDINATED MOVEMENT BLOCKS

**TransitionMode** Transition mode in blending mode  long
1 .... TMNone
2 .... TMStartVelocity
3 .... TMConstantVelocity
4 .... TMCornerDistance
5 .... TMMaxCornerDeviation
11 .... Smooth

**TransitionParameter** Parameter for transition (depends on transition mode)  double

**RampIn** RampIn factor (0 = RampIn mode not used)  double

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pc</td>
<td>Control-points matrix</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>⊙ [0.0 1.0 2.0; 0.0 1.0 1.0; 0.0 1.0 0.0]</td>
<td></td>
</tr>
<tr>
<td>pk</td>
<td>Knot-points vector</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>⊙ [0.0 0.0 0.0 0.0 0.5 1.0 1.0]</td>
<td></td>
</tr>
<tr>
<td>pw</td>
<td>Weighting vector</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>⊙ [1.0 1.0 1.0]</td>
<td></td>
</tr>
<tr>
<td>pv</td>
<td>Polynoms for feedrate definition</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>⊙ [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]</td>
<td></td>
</tr>
<tr>
<td>pt</td>
<td>Knot-points (time [s]) for feedrate</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>⊙ [0.0 1.0 10.0 11.0]</td>
<td></td>
</tr>
<tr>
<td>user</td>
<td>Only for special edit</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>⊙ [0.0 1.0 2.0 3.0]</td>
<td></td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxesGroup</td>
<td>Axes group reference</td>
<td>reference</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>bool</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
<td>bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td>i ....</td>
<td>REX general error</td>
<td></td>
</tr>
</tbody>
</table>
MC_GroupSetOverride – Set group override factors

Block Symbol

Licence: COORDINATED MOTION CONTROL

Function Description

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

Inputs

- **uAxesGroup**: Axes group reference
- **Enable**: Block function is enabled
- **VelFactor**: Velocity multiplication factor
- **AccFactor**: Acceleration/deceleration multiplication factor
- **JerkFactor**: Jerk multiplication factor

Parameter

- **diff**: Deadband (difference for recalculation), $\varnothing 0.05$, double

Outputs

- **yAxesGroup**: Axes group reference
- **Enabled**: Signal that the override factors are set successfully
- **Busy**: Algorithm not finished yet
- **Error**: Error occurred
- **ErrorID**: Error code
  
  i ..... REX general error
Appendix A

Licensing of individual function blocks

The function blocks of the REX Control System are divided into several licensing groups to provide maximum flexibility for individual projects.

The STANDARD function blocks are always available, the other groups require activation by a corresponding licence.

<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS_</td>
<td>STANDARD</td>
</tr>
<tr>
<td>ABSROT</td>
<td>STANDARD</td>
</tr>
<tr>
<td>ACD</td>
<td>OTHER</td>
</tr>
<tr>
<td>ADD</td>
<td>OTHER</td>
</tr>
<tr>
<td>ADDOCT</td>
<td>OTHER</td>
</tr>
<tr>
<td>AFLUSH</td>
<td>OTHER</td>
</tr>
<tr>
<td>ALB</td>
<td>OTHER</td>
</tr>
<tr>
<td>ALBI</td>
<td>OTHER</td>
</tr>
<tr>
<td>ALN</td>
<td>OTHER</td>
</tr>
<tr>
<td>ALNI</td>
<td>OTHER</td>
</tr>
<tr>
<td>AND_</td>
<td>OTHER</td>
</tr>
<tr>
<td>ANDOCT</td>
<td>OTHER</td>
</tr>
<tr>
<td>ANLS</td>
<td>OTHER</td>
</tr>
<tr>
<td>ARC</td>
<td>OTHER</td>
</tr>
<tr>
<td>ARLY</td>
<td>OTHER</td>
</tr>
<tr>
<td>ASW</td>
<td>STANDARD</td>
</tr>
<tr>
<td>ATMT</td>
<td>OTHER</td>
</tr>
<tr>
<td>AVG</td>
<td>OTHER</td>
</tr>
<tr>
<td>AVS</td>
<td>OTHER</td>
</tr>
<tr>
<td>BDHEXD</td>
<td>OTHER</td>
</tr>
</tbody>
</table>

The list continues on the next page...
APPENDIX A. LICENSING OF INDIVIDUAL FUNCTION BLOCKS

<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STANDARD</td>
</tr>
<tr>
<td>BDOCT</td>
<td>●</td>
</tr>
<tr>
<td>BINS</td>
<td>●</td>
</tr>
<tr>
<td>BIS</td>
<td>●</td>
</tr>
<tr>
<td>BITOP</td>
<td>●</td>
</tr>
<tr>
<td>BMHEXD</td>
<td>●</td>
</tr>
<tr>
<td>BMDCX</td>
<td>●</td>
</tr>
<tr>
<td>BPF</td>
<td>●</td>
</tr>
<tr>
<td>CDELSM</td>
<td>●</td>
</tr>
<tr>
<td>CMP</td>
<td>●</td>
</tr>
<tr>
<td>CNA</td>
<td>●</td>
</tr>
<tr>
<td>CNB</td>
<td>●</td>
</tr>
<tr>
<td>CNDR</td>
<td>●</td>
</tr>
<tr>
<td>CNE</td>
<td>●</td>
</tr>
<tr>
<td>CNI</td>
<td>●</td>
</tr>
<tr>
<td>CNR</td>
<td>●</td>
</tr>
<tr>
<td>CNS</td>
<td>●</td>
</tr>
<tr>
<td>CONCAT</td>
<td>●</td>
</tr>
<tr>
<td>COUNT</td>
<td>●</td>
</tr>
<tr>
<td>CSSM</td>
<td>●</td>
</tr>
<tr>
<td>DATE_</td>
<td>●</td>
</tr>
<tr>
<td>DATETIME</td>
<td>●</td>
</tr>
<tr>
<td>DDELSM</td>
<td>●</td>
</tr>
<tr>
<td>DEL</td>
<td>●</td>
</tr>
<tr>
<td>DELM</td>
<td>●</td>
</tr>
<tr>
<td>DER</td>
<td>●</td>
</tr>
<tr>
<td>DIF_</td>
<td>●</td>
</tr>
<tr>
<td>Display</td>
<td>●</td>
</tr>
<tr>
<td>DIV</td>
<td>●</td>
</tr>
<tr>
<td>DSSM</td>
<td>●</td>
</tr>
<tr>
<td>EAS</td>
<td>●</td>
</tr>
<tr>
<td>EATMT</td>
<td>●</td>
</tr>
<tr>
<td>EDGE_</td>
<td>●</td>
</tr>
<tr>
<td>EMD</td>
<td>●</td>
</tr>
<tr>
<td>EPC</td>
<td>●</td>
</tr>
<tr>
<td>EVAR</td>
<td>●</td>
</tr>
<tr>
<td>EXEC</td>
<td>●</td>
</tr>
<tr>
<td>FIND</td>
<td>●</td>
</tr>
<tr>
<td>FLCU</td>
<td>●</td>
</tr>
<tr>
<td>FNX</td>
<td>●</td>
</tr>
</tbody>
</table>

The list continues on the next page...
<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STANDARD</td>
</tr>
<tr>
<td>FNXY</td>
<td></td>
</tr>
<tr>
<td>FOPDT</td>
<td></td>
</tr>
<tr>
<td>FRID</td>
<td></td>
</tr>
<tr>
<td>From</td>
<td></td>
</tr>
<tr>
<td>GAIN</td>
<td></td>
</tr>
<tr>
<td>GETPA</td>
<td></td>
</tr>
<tr>
<td>GETPB</td>
<td></td>
</tr>
<tr>
<td>GETPI</td>
<td></td>
</tr>
<tr>
<td>GETPA</td>
<td></td>
</tr>
<tr>
<td>GETPS</td>
<td></td>
</tr>
<tr>
<td>Goto</td>
<td></td>
</tr>
<tr>
<td>GotoTagVisibility</td>
<td></td>
</tr>
<tr>
<td>GRADS</td>
<td></td>
</tr>
<tr>
<td>HMI</td>
<td></td>
</tr>
<tr>
<td>HTTP</td>
<td></td>
</tr>
<tr>
<td>I3PM</td>
<td></td>
</tr>
<tr>
<td>IADD</td>
<td></td>
</tr>
<tr>
<td>IDIV</td>
<td></td>
</tr>
<tr>
<td>IMOD</td>
<td></td>
</tr>
<tr>
<td>IMUL</td>
<td></td>
</tr>
<tr>
<td>INFO</td>
<td></td>
</tr>
<tr>
<td>INHEXD</td>
<td></td>
</tr>
<tr>
<td>INOCT</td>
<td></td>
</tr>
<tr>
<td>Inport</td>
<td></td>
</tr>
<tr>
<td>INQUAD</td>
<td></td>
</tr>
<tr>
<td>INSTD</td>
<td></td>
</tr>
<tr>
<td>INTE</td>
<td></td>
</tr>
<tr>
<td>INTSM</td>
<td></td>
</tr>
<tr>
<td>IODRV</td>
<td></td>
</tr>
<tr>
<td>IOTASK</td>
<td></td>
</tr>
<tr>
<td>ISSW</td>
<td></td>
</tr>
<tr>
<td>ISUB</td>
<td></td>
</tr>
<tr>
<td>ITOI</td>
<td></td>
</tr>
<tr>
<td>KDER</td>
<td></td>
</tr>
<tr>
<td>LC</td>
<td></td>
</tr>
<tr>
<td>LEN</td>
<td></td>
</tr>
<tr>
<td>LIN</td>
<td></td>
</tr>
<tr>
<td>LLC</td>
<td></td>
</tr>
<tr>
<td>LPBRK</td>
<td></td>
</tr>
</tbody>
</table>

*The list continues on the next page...*
## APPENDIX A. LICENSING OF INDIVIDUAL FUNCTION BLOCKS

<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPF</td>
<td>•</td>
</tr>
<tr>
<td>MC_AccelerationProfile</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_AddAxisToGroup</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_CamIn</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_CamOut</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_CombineAxes</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GearIn</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GearInPos</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GearOut</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GroupContinue</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GroupDisable</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GroupEnable</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GroupHalt</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GroupInterrupt</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GroupReadActualAcceleration</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GroupReadActualPosition</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GroupReadActualVelocity</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GroupReadError</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GroupReadStatus</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GroupReset</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GroupSetOverride</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GroupSetPosition</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_GroupStop</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_Halt</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_HaltSuperimposed</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_Home</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveAbsolute</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveAdditive</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveCircularAbsolute</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveCircularRelative</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveContinuousAbsolute</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveContinuousRelative</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveDirectAbsolute</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveDirectRelative</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveLinearAbsolute</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveLinearRelative</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MovePath</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MovePath_PH</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveRelative</td>
<td>MOTION CONTROL</td>
</tr>
</tbody>
</table>

The list continues on the next page...
<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC_MoveSuperimposed</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveVelocity</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_PhasingAbsolute</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_PhasingRelative</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_PositionProfile</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_Power</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_ReadActualPosition</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_ReadAxisError</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_ReadBoolParameter</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_ReadCartesianTransform</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_ReadParameter</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_ReadStatus</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_Reset</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_UngroupAllAxes</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MC_VelocityProfile</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_WriteBoolParameter</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_WriteParameter</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_AccelerationProfile</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_CamIn</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_CamTableSelect</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_CombineAxes</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_GearIn</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_GearInPos</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_GroupHalt</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_GroupInterrupt</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_GroupSetOverride</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_GroupSetPosition</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_GroupStop</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_Halt</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_HaltSuperimposed</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_Home</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_MoveAbsolute</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_MoveAdditive</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_MoveCircularAbsolute</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_MoveCircularRelative</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
</tbody>
</table>

*The list continues on the next page...*
<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCP_MoveContinuousAbsolute</td>
<td>STANDARD</td>
</tr>
<tr>
<td>MCP_MoveContinuousRelative</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_MoveDirectAbsolute</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_MoveDirectRelative</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_MoveLinearAbsolute</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_MoveLinearRelative</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_MovePath</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_MovePath_PH</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_MoveRelative</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_MoveSuperimposed</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_MoveVelocity</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_PhasingAbsolute</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_PhasingRelative</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_PositionProfile</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_SetCartesianTransform</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_SetKinTransform_Arm</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_SetOverride</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_Stop</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_TorqueControl</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_VelocityProfile</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCU</td>
<td>●</td>
</tr>
<tr>
<td>MDL</td>
<td>●</td>
</tr>
<tr>
<td>MDLI</td>
<td>●</td>
</tr>
<tr>
<td>MID</td>
<td>●</td>
</tr>
<tr>
<td>MINMAX</td>
<td>●</td>
</tr>
<tr>
<td>MODULE</td>
<td>●</td>
</tr>
<tr>
<td>MP</td>
<td>●</td>
</tr>
<tr>
<td>MUL</td>
<td>●</td>
</tr>
<tr>
<td>MVD</td>
<td>●</td>
</tr>
<tr>
<td>NOT_</td>
<td>●</td>
</tr>
<tr>
<td>NSCL</td>
<td>●</td>
</tr>
<tr>
<td>OR_</td>
<td>●</td>
</tr>
<tr>
<td>OROCT</td>
<td>●</td>
</tr>
<tr>
<td>OSCALL</td>
<td>●</td>
</tr>
<tr>
<td>OUTHEXD</td>
<td>●</td>
</tr>
<tr>
<td>OUTOCT</td>
<td>●</td>
</tr>
<tr>
<td>Outport</td>
<td>●</td>
</tr>
<tr>
<td>OUTQUAD</td>
<td>●</td>
</tr>
<tr>
<td>OUTRHEXD</td>
<td>ADVANCED</td>
</tr>
</tbody>
</table>

*The list continues on the next page...*
<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTROCT</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>OUTRQUAD</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>OUTRSTD</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>OUTSTD</td>
<td></td>
</tr>
<tr>
<td>PARA</td>
<td></td>
</tr>
<tr>
<td>PARB</td>
<td></td>
</tr>
<tr>
<td>PARI</td>
<td></td>
</tr>
<tr>
<td>PARR</td>
<td></td>
</tr>
<tr>
<td>PARS</td>
<td></td>
</tr>
<tr>
<td>PIDAT</td>
<td>AUTOTUNING</td>
</tr>
<tr>
<td>PIDE</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>PIDGS</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>PIDMA</td>
<td>AUTOTUNING</td>
</tr>
<tr>
<td>PIDU</td>
<td></td>
</tr>
<tr>
<td>PIDUI</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>PJROCT</td>
<td></td>
</tr>
<tr>
<td>PJSOCT</td>
<td></td>
</tr>
<tr>
<td>POL</td>
<td></td>
</tr>
<tr>
<td>POUT</td>
<td></td>
</tr>
<tr>
<td>PRBS</td>
<td></td>
</tr>
<tr>
<td>PRGM</td>
<td></td>
</tr>
<tr>
<td>PROJECT</td>
<td></td>
</tr>
<tr>
<td>PSMPC</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>PWM</td>
<td></td>
</tr>
<tr>
<td>QFC</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>QFD</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>QTASK</td>
<td></td>
</tr>
<tr>
<td>RDC</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>REC</td>
<td></td>
</tr>
<tr>
<td>REGEXP</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>REL</td>
<td></td>
</tr>
<tr>
<td>REPLACE</td>
<td></td>
</tr>
<tr>
<td>REXLANG</td>
<td>REXLANG</td>
</tr>
<tr>
<td>RLIM</td>
<td></td>
</tr>
<tr>
<td>RLY</td>
<td></td>
</tr>
<tr>
<td>RM_AxesGroup</td>
<td>COORDINATED MOTION CONTROL</td>
</tr>
<tr>
<td>RM_Axis</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>RM_AxisOut</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>RM_AxisSpline</td>
<td>MOTION CONTROL</td>
</tr>
</tbody>
</table>

*The list continues on the next page...*
### Function block name | Licensing group | Other
--- | --- | ---
RM_Feed | STANDARD | COORDINATED MOTION CONTROL
RM_Gcode | STANDARD | COORDINATED MOTION CONTROL
RM_GroupTrack | STANDARD | COORDINATED MOTION CONTROL
RM_Track | STANDARD | MOTION CONTROL
RS | • | 
RT01 | • | 
RT0S | • | 
RT0V | • | 
S10F2 | STANDARD | ADVANCED
SAIL | STANDARD | ADVANCED
SAT | • | 
SC2FA | STANDARD | AUTOTUNING
SCU | • | 
SCUV | • | 
SEL | • | 
SELHEXD | • | 
SELFCT | • | 
SELQUAD | • | 
SELSOCT | • | 
SELU | • | 
SETPA | • | 
SETPB | • | 
SETP1 | • | 
SETPR | • | 
SETPS | • | 
SGL | • | 
SGLI | • | 
SGS LP | • | ADVANCED
SHIFTOCT | • | 
SILD | • | 
SILD | • | 
SINT | • | 
SLEEP | • | 
SMHCC | • | ADVANCED
SMHCCA | • | AUTOTUNING
SMTP | • | ADVANCED
SOPDT | • | ADVANCED
SPIKE | • | ADVANCED
SQR | • | 

*The list continues on the next page...*
<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td>STANDARD</td>
<td>Other</td>
</tr>
<tr>
<td>SQRT_</td>
<td>●</td>
</tr>
<tr>
<td>SR</td>
<td>●</td>
</tr>
<tr>
<td>SRTF</td>
<td></td>
</tr>
<tr>
<td>SSw</td>
<td>●</td>
</tr>
<tr>
<td>STOR</td>
<td>●</td>
</tr>
<tr>
<td>SUB</td>
<td>●</td>
</tr>
<tr>
<td>SubSystem</td>
<td>●</td>
</tr>
<tr>
<td>SNR</td>
<td>●</td>
</tr>
<tr>
<td>SWU</td>
<td>●</td>
</tr>
<tr>
<td>SWVMR</td>
<td>●</td>
</tr>
<tr>
<td>TASK</td>
<td>●</td>
</tr>
<tr>
<td>TIME</td>
<td>●</td>
</tr>
<tr>
<td>TIMER_</td>
<td>●</td>
</tr>
<tr>
<td>TIODRV</td>
<td>●</td>
</tr>
<tr>
<td>TRND</td>
<td>●</td>
</tr>
<tr>
<td>TRNDLF</td>
<td>●</td>
</tr>
<tr>
<td>TRNDV</td>
<td>●</td>
</tr>
<tr>
<td>TRNDVLF</td>
<td>●</td>
</tr>
<tr>
<td>TSE</td>
<td>●</td>
</tr>
<tr>
<td>VDEL</td>
<td>●</td>
</tr>
<tr>
<td>VIN</td>
<td>●</td>
</tr>
<tr>
<td>VOUT</td>
<td>●</td>
</tr>
<tr>
<td>VTOR</td>
<td>●</td>
</tr>
<tr>
<td>WSCH</td>
<td>●</td>
</tr>
<tr>
<td>WWW</td>
<td>●</td>
</tr>
<tr>
<td>ZV41S</td>
<td>●</td>
</tr>
</tbody>
</table>

ADVANCED
Appendix B

Error codes of the REX Control 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 ....... REX 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 .... REX Module error - some driver or block is not installed or licensed
-109 .... REX I/O driver error
APPENDIX B. ERROR CODES OF THE REX CONTROL SYSTEM

-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 . . . REX configuration is being changed
-117 . . . Invalid target device
-118 . . . Access denied by REX 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 REX group
-124 . . . Password mismatch
-125 . . . Bad user name or password
-126 . . . Target device is not compatible

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 .... REX configuration has been changed
-411 .... REX executive is being terminated
-412 .... REX 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
APPENDIX B. ERROR CODES OF THE REX CONTROL SYSTEM

-426 . . . SSL handshake error
-427 . . . Certificate verification error
-428 . . . Reserved for SSL 2
-429 . . . Reserved for SSL 3
-430 . . . Reserved for SSL 4
-431 . . . Reserved for SSL 5
-432 . . . Relay rejected
-433 . . . STARTTLS rejected
-434 . . . Authentication method rejected
-435 . . . Authentication failed
-436 . . . Send operation failed
-437 . . . Receive operation failed
-438 . . . Communication command failed
-439 . . . Receiving buffer too small
-440 . . . Sending buffer too small
-441 . . . Invalid header
-442 . . . HTTP server responded with error
-443 . . . HTTP server responded with redirect
-444 . . . Operation would blok
-445 . . . Invalid operation
-446 . . . Communication closed
-447 . . . Connection cancelled

Numerical error codes

-500 . . . General numeric error
-501 . . . Division by zero
-502 . . . Numeric stack overflow
-503 . . . Invalid numeric instruction
-504 . . . Invalid numeric address
-505 . . . Invalid numeric type
-506 . . . Not initialized numeric value
-507 . . . Numeric argument overflow/underflow
-508 . . . Numeric range check error
-509 . . . Invalid subvector/submatrix range
-510 . . . Numeric value too close to zero

Archive system codes

-600 . . . Archive seek underflow
-601 . . . Archive semaphore fatal error
-602 . . . Archive cleared
-603 . . . Archive reconstructed from saved vars
-604 . . . Archive reconstructed from normal vars
-605 . . . Archive check summ error
-606 . . . Archive integrity error
-607 .... Archive sizes changed
-608 .... Maximum size of disk archive file exceeded

**Motion control codes**

-700 .... MC - Invalid parameter
-701 .... MC - Out of range
-702 .... MC - Position not reachable
-703 .... MC - Invalid axis state
-704 .... MC - Torque limit exceeded
-705 .... MC - Time limit exceeded
-706 .... MC - Distance limit exceeded
-707 .... MC - Step change in position or velocity
-708 .... MC - Base axis error or invalid state
-709 .... MC - Stopped by 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
Bibliography


Index

TODO
  SRTF DLOG, 34
  strfind?, 359
  X, 105, 111, 112, 114, 115, 144, 148, 331–333, 336

ABS_, 63, 535
absolute
  position sensor, 101
absolute value, 63
ABSROT, 101, 535
ACD, 275, 535
ADD, 64, 65, 535
addition, 64
  extended, 72
  integer, 81
ADDOCT, 64, 65, 97, 535
AFLUSH, 285, 535
alarm
  Boolean value, 271
  numerical value, 273
ALB, 271, 535
ALBI, 271, 535
algebraic loop, 28
ALN, 273, 535
ALNI, 273, 535
analog input
  safety, 129
AND_, 234, 235, 535
ANDOCT, 234, 235, 535
ANLS, 150, 535
application
  of the REX Control System, 20
ARC, 18, 21, 272, 274, 276, 279, 281, 285, 535
architecture
  open, 29
  archiv, 270
  archive, 18
  backed-up memory, 270
  configuration, 18
  disk, 270
  RAM memory, 270
archiving
  delta criterion, 275
ARLY, 163, 535
ASW, 103, 535
ATMT, 12, 236, 244, 304, 313, 317, 535
automaton
  finite-state, 236, 244
average
  moving, 105
AVG, 105, 535
AVS, 12, 106, 535

band
  frequency transmission, 107
band-pass filter, 107
bandwidth, 120
BDHEXD, 239, 244, 535
BDOCT, 239, 244, 536
Bessel filter, 120
binary number
  transformation, 251
binary sequence
  generator, 152, 154
BINS, 152, 536
BIS, 152, 154, 155, 536
BITOP, 240, 536
block
  description, 13
  description format, 13

553
INDEX

blocks
- analog signal processing, 12
- data archiving, 12
- generators, 12
- input-output, 11
- logic control, 12
- parameter-related, 12
- regulation, 12
- special, 13

BMHEXD, 241, 244, 536
BMOCX, 241, 244, 536
Boolean complementation, 253
BPF, 107, 536
Butterworth filter, 120
CDELSSM, 322, 536
circuit
- flip-flop Reset-Set, 256
- flip-flop Set-Reset, 257
CMP, 108, 536
CNA, 338, 536
CNB, 66, 536
CNDR, 109, 536
CNE, 67, 536
CNI, 68, 536
CNR, 69, 536
CNS, 288, 536
coefficient
- relative damping, 107, 120
comparator, 108
compatibility
- REX and Simulink, 28
compensator
- lead, 171
- lead-lag, 172

nonlinearity, 109
simple nonlinearity, 122
compiler
- RexComp, 20, 28
compression, 275
CONCAT, 289, 536
conditioner
- nonlinear, 109
configuration
- archives, 20
- computation task, 20
- input-output drivers, 20
- modules, 20
- REX Control System, 20
constant
- Boolean, 66
- integer, 68
- logic, 66
- real, 69
control
- motion, 13
- sequential, 236, 244
controller
- fuzzy logic, 164
- PID, 188
- PID with gain scheduling, 180
- PID with input-defined parameters, 191
- PID with relay autotuner, 175
- PID with static error, 178
- step, 211, 214
- with frequency autotuner, 205
correction
- real to integer, 94
COUNT, 15, 242, 536
counter
- controlled, 242
CSSM, 325, 536
data
- remote connection, 351
data storing, 277, 280
data types, 14
DATE_, 262, 263, 536
DATE_TIME, 262, 263, 266, 536
DEDELSSM, 327, 536
dead time, 331, 335
DEL, 111, 536
delay
  transport, 112
  variable, 144
  with initialization, 111
DELM, 112, 536
delta criterion, 275
demultiplexer
  bitwise, 239
denominator, 73
DER, 113, 536
derivation, 113, 118
detection
  edge, 247
deviation
  standard, 115
DIF_, 70, 536
difference, 70
Display, 42, 536
DIV, 71, 536
division
  extended, 73
  integer, 87, 88
  remainder, 88
  two signals, 71
DLL library, 29
driver
  .rio file extension, 25
  configuration data, 25
  input-output, 11
  input/output, 25
  input/output with tasks, 38
  REX Control System, 25
  user manual, 27
drivers
  REX system, 11
DSSM, 329, 536
EAS, 72, 536
EATMT, 244, 536
edge detection, 247
EDGE_, 136, 247, 536
element
  three state, 232
EMD, 73, 536
EPC, 35, 344, 536
error
  fatal, 31
EVAR, 115, 536
EXEC, 18, 20, 25–27, 29, 31, 32, 36–39, 536
executive
  configuration, 11, 17
  real-time, 20
  RexCore program, 11
external program, 344
feedback loop, 28
filter
  band-pass, 107
  Bessel, 120
  Butterworth, 120
  low-pass, 120
  moving average, 105
  nonlinear, 140
  spike, 140
filtering, 113, 118
  digital, 31
FIND, 290, 536
finite-state machine, 236, 244
first order system, 335
FLCU, 12, 164, 536
flip-flop circuit
  Reset-Set, 256
  Set-Reset, 257
FNX, 74, 536
FNXY, 76, 537
FOPDT, 172, 335
FOPDT, 331, 537
frequency transmission band, 107
FRID, 167, 537
From, 43, 46, 47, 537
function
  operating system, 35
  single variable, 74
two variables, 76
fuzzy logic, 164

GAIN, 78, 537
gain, 78
generator
    binary sequence, 152, 154
    piecewise linear function, 150
    signal, 158
    time function, 194
GETPA, 302, 537
GETPB, 304, 537
GETPI, 304, 537
GETPS, 306, 537
Goto, 43–45, 47, 537
GotoTagVisibility, 46, 47, 537
GRADS, 79, 537
HMI, 22, 537
HTTP, 347, 537
hysteresis, 108
I3PM, 169, 537
IADD, 81, 537
identification
    three parameter model, 169
IDIV, 87, 537
IMOD, 88, 537
IMUL, 85, 537
INFO, 24, 537
INHEXD, 51, 537
INOCT, 51, 537
INQUAD, 51, 537
INSTD, 43, 51, 52, 537
INTE, 116, 139, 537
integer
    division, 87
integer number
    transformation, 251
integer signal
    switching, 249
integrator
    controlled, 116
    simple, 139
interpolation
    linear, 89
INTSM, 248, 250, 537
IODRV, 21, 25, 43, 45, 537
IOTASK, 27, 34, 38, 302, 304, 311, 313, 323, 325, 537
ISSW, 249, 537
ISUB, 83, 537
ITOI, 251, 537
KDER, 118, 537
LC, 171, 537
least squares method, 113
LEN, 291, 537
LIN, 89, 537
linear interpolation, 89
linear function
    generator, 150
LLC, 172, 335, 537
logical sum, 254
loop
    algebraic, 28
    feedback, 28
low-pass filter, 120
LPBRK, 11, 28, 103, 537
LPF, 120, 538
LSM, 113
maximum, 121
MC_AccelerationProfile, 380, 412, 413, 432, 433, 538
MC_AddAxisToGroup, 482, 538
MC_CamIn, 446, 453, 457, 467, 470, 538
MC_CamOut, 446, 450, 538
MC_CombineAxes, 454, 538
MC_GearIn, 457, 460, 465, 467, 470, 538
MC_GearInPos, 460, 538
MC_GearOut, 465, 538
MC_GroupContinue, 502, 504, 538
MC_GroupDisable, 485, 538
MC_GroupEnable, 484, 538
MC_GroupHalt, 497, 538
INDEX

MC_GroupInterrupt, 502, 504, 538
MC_GroupReadActualAcceleration, 493, 538
MC_GroupReadActualPosition, 491, 538
MC_GroupReadActualVelocity, 492, 538
MC_GroupReadError, 507, 538
MC_GroupReadStatus, 505, 538
MC_GroupReset, 508, 538
MC_GroupSetOverride, 533, 538
MC_GroupSetPosition, 489, 538
MC_GroupStop, 494, 538
MC_Halt, 384, 538
MC_HaltSuperimposed, 385, 538
MC_Home, 386, 417, 538
MC_MoveAbsolute, 388, 401, 402, 425, 443, 461, 538
MC_MoveAdditive, 389, 392, 538
MC_MoveCircularAbsolute, 517, 538
MC_MoveCircularRelative, 521, 538
MC_MoveContinuousAbsolute, 401, 538
MC_MoveContinuousRelative, 404, 538
MC_MoveDirectAbsolute, 525, 538
MC_MoveDirectRelative, 528, 538
MC_MoveLinearAbsolute, 509, 538
MC_MoveLinearRelative, 513, 538
MC_MovePath, 531, 538
MC_MovePath_PH, 538
MC_MoveRelative, 388, 389, 395, 398, 404, 405, 538
MC_MoveSuperimposed, 389, 398, 467, 470, 539
MC_MoveVelocity, 373, 408, 443, 539
MC_PhasingAbsolute, 467, 539
MC_PhasingRelative, 470, 539
MC_PositionProfile, 380, 381, 412, 432, 433, 443, 452, 539
MC_Power, 416, 539
MC_ReadActualPosition, 417, 539
MC_ReadAxisError, 418, 539
MC_ReadBoolParameter, 419, 539
MC_ReadCartesianTransform, 488, 539
MC_ReadParameter, 420, 539
MC_ReadStatus, 422, 539
MC_Reset, 424, 508, 539
MC_SetCartesianTransform, 486, 539
MC_SetOverride, 425, 539
MC_SetPosition, 386
MC_Stop, 384, 427, 539
MC_TorqueControl, 429, 539
MC_UngroupAllAxes, 483, 539
MC_VelocityProfile, 380, 381, 412, 413, 432, 539
MC_WriteBoolParameter, 436, 539
MC_WriteParameter, 437, 539
MCP_AccelerationProfile, 380, 539
MCP_CamIn, 446, 539
MCP_CamTableSelect, 446, 447, 452, 539
MCP_CombineAxes, 454, 539
MCP_GearIn, 457, 539
MCP_GearInPos, 460, 539
MCP_GroupHalt, 539
MCP_GroupInterrupt, 502, 539
MCP_GroupSetOverride, 539
MCP_GroupSetPosition, 489, 539
MCP_GroupStop, 539
MCP_Halt, 384, 539
MCP_HaltSuperimposed, 385, 539
MCP_Home, 386, 539
MCP_MoveAbsolute, 388, 539
MCP_MoveAdditive, 392, 539
MCP_MoveCircularAbsolute, 539
MCP_MoveCircularRelative, 539
MCP_MoveContinuousAbsolute, 401, 540
MCP_MoveContinuousRelative, 404, 540
MCP_MoveDirectAbsolute, 540
MCP_MoveDirectRelative, 540
MCP_MoveLinearAbsolute, 540
MCP_MoveLinearRelative, 540
MCP_MovePath, 540
MCP_MovePath_PH, 540
MCP_MoveRelative, 395, 540
MCP_MoveSuperimposed, 398, 540
MCP_MoveVelocity, 408, 540
MCP_PhasingAbsolute, 467, 540
MCP_PhasingRelative, 470, 540
MCP_PositionProfile, 412, 540
MCP_SetCartesianTransform, 540
MCP_SetKinTransform_Arm, 540
MCP_SetOverride, 425, 540
MCP_Stop, 427, 540
MCP_TorqueControl, 429, 540
MCP_VelocityProfile, 432, 540
MCU, 173, 231, 540
MDL, 332, 333, 540
MDLI, 333, 540
mean value, 115
MID, 292, 540
minimum, 121
MINMAX, 121, 540
model
  first order, 331
  FOPDT, 172, 335
  process, 332, 333
  second order, 335
  state space
    continuous, 325
    continuous with time delay, 322
    discrete, 329
    discrete with time delay, 327
modulation
  pulse width, 200
MODULE, 21, 25, 29, 540
module, 29
  extending the REX Control System, 29
  extension, 25
motion control, 13, 106
moving average, 105
MP, 155, 540
MPC, 196
MUL, 90, 540
multiplexer
  bitwise, 241
multiplication
  by a constant, 78
  extended, 73
  two signals, 90
MVD, 334, 540
negation, 253
nonlinear transformation
  simple, 122
NOT_, 253, 540
NSCL, 122, 540
OPC server, 354
operating system, 35
operation
  binary, 93
  bitwise, 240
operator
  relational, 93
optimization
  gradient based, 79
OR_, 254, 255, 540
order
  driver execution, 25
  driver initialization, 25
  module execution, 29
  module initialization, 29
  of task execution, 36
  of task initialization, 36
OROCT, 254, 255, 540
OSCALL, 35, 346, 540
OUTH杏D, 52, 53, 540
OUTOCT, 52, 53, 540
Outport, 48, 50, 540
output
  pulse, 193
  three state, 232
output saturation, 203
OUTQUAD, 52, 53, 540
OUTRSTD, 55, 541
OUTSTD, 45, 51, 52, 55, 541
overhead
  control system core, 20
PARA, 307, 541
parameter
  tick, 20
  input-defined, 308
  remote, 302, 304, 311, 313
  remote acquisiption, 302, 304
PARB, 308, 541
PARI, 308, 541
PARR, 308, 541
INDEX

PARS, 310, 541
path
  full, 33
period
  of quick task execution, 31
  of task execution, 36
PID
  autotuning, 182
  controller, 188
  with gain scheduling, 180
  with input-defined parameters, 191
  with moment autotuner, 182
  with relay autotuner, 175
  with static error, 178
PIDAT, 12, 175, 541
PIDE, 178, 541
PIDGS, 12, 180, 541
PIDMA, 12, 182, 354, 541
PIDU, 175, 178, 180, 182, 188, 191, 231, 354, 374, 541
PIDUI, 191, 541
PJROCT, 293, 541
PJSOCT, 294, 541
POL, 91, 541
polynomial
  evaluation, 91
position sensor
  absolute, 101
POUT, 193, 541
PRBS, 156, 541
prediction, 113
predictive control, 196
PRGM, 194, 541
priority
  dependancy on the operating system,
    21
  logic, 21
  logical, 25, 31
  of tasks, 36
process
  model, 332
  model with variable parameters, 333
product
  Boolean, 234, 235
  logical, 234, 235
program
  RexView, Halt/Run button, 33
  external, 344
  RexDraw, 25
  RexView, 26
  RexView, Enable checkbox, 33
  RexView, Reset button, 33
programmable block, 356
programme
  RexView, 18
  weekly, 267
programmer, 194
PROJECT, 30, 541
project
  main file, 20, 25
  protocol
  UDP/IP, 351
PSMPC, 12, 196, 541
puls, 193
pulse
  manually generated, 155
  pulse counting
    bidirectional, 242
  pulse output, 193
  pulse width modulation, 200
PWM, 181, 186, 190, 192, 200, 220, 224, 225, 541
QFC, 56, 57, 362, 541
QFD, 53, 55–57, 362, 541
QTASK, 20, 21, 27, 31, 34, 36, 323, 325, 541
quotient, 71
  integer, 87
rate limiter, 125
rate monotonic scheduling, 21
RDC, 13, 351, 541
RDFT, 123
real-time
  executive, 17, 20
REC, 92, 541
reciprocal value, 92
REGEXP, 295, 541
REL, 93, 541
relative damping coefficient, 107, 120
relay
    advance, 163
    with hysteresis, 202
remote
    data connection, 351
    parameter, 302, 304
REPLACE, 296, 541
RexComp
compiler, 28
RexComp compiler, 20
REXLANG, 13, 356, 541
RLIM, 125, 541
RLY, 163, 202, 541
RM_AxesGroup, 476, 482, 507, 541
RM_Axis, 340, 373, 374, 386, 416–420, 427,
    439, 441, 482, 541
RM_AxisOut, 439, 441, 541
RM_AxisSpline, 441, 541
RM_Feed, 479, 542
RM_Gcode, 480, 542
RM_GroupTrack, 542
RM_Track, 443, 542
root
    square, 96
RS, 256, 542
RTOI, 94, 542
RTOS, 297, 542
RTOV, 339, 345, 542
S1OF2, 126, 542
safety analog input, 129
safety selector, 126
SAI, 126, 128, 129, 542
sample and hold, 138
SAT, 203, 542
SC2FA, 205, 542
schedule
    weekly, 267
SCU, 181, 186, 190, 192, 211, 542
SCUV, 181, 186, 188, 190, 192, 214, 542
SEL, 132, 542
selector
    active controller, 218
    analog signal, 126, 132
    safety, 126
    with ramp, 143
SELHEXD, 134, 542
SELOCT, 134, 542
SELQUAD, 132, 134, 542
SELSOCT, 298, 542
SELU, 218, 340, 542
sensor
    absolute position, 101
    sequence
        pseudo-random binary, 156
    sequential control, 236, 244
SETPA, 311, 542
SETPB, 313, 542
SETPA, 313, 542
SETPB, 313, 542
SETPR, 313, 317, 542
SETPS, 315, 542
SETPS, 315, 542
setpoint, 194
setpoint, 194
SETPSC, 313, 542
setpoint, 194
SETPSC, 313, 542
setpoint, 194
SETPSC, 313, 542
signal generator, 158
SILD, 318, 320, 542
Simulation
    parameters, 32
    real-time, 32
Simulink, 32
SINT, 116, 139, 542
SLEEP, 11, 32, 542
SMHCC, 220, 542
SMHCCA, 224, 542
SMTP, 349, 542
SOPDT, 335, 542
SPIKE, 129–131, 140, 542
SQR, 95, 542
SQR])] 96, 543
square root, 96
square value, 95
SR, 257, 543
SRTF, 33, 543
INDEX

SSW, 142, 340, 354, 543
stack
  size, 25
standard deviation, 115
starting unit, 106
state machine, 236, 244
state space model, 325, 329
  with time delay, 322, 327
step controller
  with position feedback, 211
  with velocity output, 214
STOR, 300, 543
SUB, 65, 97, 543
SubSystem, 50, 543
subsystem
  archiving, 269
  execution, 33
subtraction
  extended, 72
  two signals, 97
sum, 64
  Boolean, 255
  logical, 254, 255
switch
  integer signals, 249
  simple, 142
  unit, 231
  with automatic selection of input, 103
SWR, 143, 340, 543
SWU, 218, 231, 543
SWVMR, 340, 543
system
  first order, 172, 331
  second order, 335
TASK, 20, 21, 27, 31, 34, 36, 323, 325, 543
task
  driver-triggered, 27
  execution, 33
  execution period, 36
  priority, 36
  quick, 31
  quick, execution period, 31
  standard, 36
TIME, 263, 266, 543
time delay, 112, 331, 335
  variable, 144
timer
  system, 27
  weekly, 267
TIMER_, 258, 543
TIDDRV, 21, 27, 38, 543
trajectory
  time-optimal, 106
transformation
  binary number, 251
  integer number, 251
transport delay, 112
trend
  recording, 277, 280
TRND, 15, 277, 280, 354, 543
TRNDLF, 282, 543
TRNDV, 280, 543
TRNDVLF, 284, 543
TSE, 211, 214, 232, 543
type
  input, 14
  output, 14
  parameter, 14
types
  of variables, 14
user programmable block, 356
value
  default, 14
  maximal, 14
  mean, 115
  minimal, 14
  reciprocal, 92
  substitute, 71, 73, 74, 76, 87, 88, 92, 96
valve
  motor driven, 334
  servo, 334
variance, 115
VDEL, 144, 543
VIN, 53, 55, 57, 58, 362, 543
VOUT, 56, 59, 362, 543
VTOR, 123, 341, 345, 543

weekly
   schedule, 267

WSCH, 267, 543

WWW, 40, 543

ZV4IS, 145, 543