## Contents

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

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

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

<table>
<thead>
<tr>
<th>Block Type</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTRQUAD, OUTROCT, OUTRHEXD</td>
<td>Multi-output blocks with verification</td>
<td>57</td>
</tr>
<tr>
<td>OUTRSTD</td>
<td>Output block with verification</td>
<td>58</td>
</tr>
<tr>
<td>QFC</td>
<td>Quality flags coding</td>
<td>59</td>
</tr>
<tr>
<td>QFD</td>
<td>Quality flags decoding</td>
<td>60</td>
</tr>
<tr>
<td>VIN</td>
<td>Validation of the input signal</td>
<td>61</td>
</tr>
<tr>
<td>VOUT</td>
<td>Validation of the output signal</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td><strong>4 MATH</strong> – Math blocks</td>
<td>63</td>
</tr>
<tr>
<td>ABS</td>
<td>Absolute value</td>
<td>65</td>
</tr>
<tr>
<td>ADD</td>
<td>Addition of two signals</td>
<td>66</td>
</tr>
<tr>
<td>ADDQUAD, ADDOCT, ADDHEXD</td>
<td>Multi-input addition</td>
<td>67</td>
</tr>
<tr>
<td>CNB</td>
<td>Boolean (logic) constant</td>
<td>68</td>
</tr>
<tr>
<td>CNE</td>
<td>Enumeration constant</td>
<td>69</td>
</tr>
<tr>
<td>CNI</td>
<td>Integer constant</td>
<td>70</td>
</tr>
<tr>
<td>CNR</td>
<td>Real constant</td>
<td>71</td>
</tr>
<tr>
<td>DIF</td>
<td>Difference</td>
<td>72</td>
</tr>
<tr>
<td>DIV</td>
<td>Division of two signals</td>
<td>73</td>
</tr>
<tr>
<td>EAS</td>
<td>Extended addition and subtraction</td>
<td>74</td>
</tr>
<tr>
<td>EMD</td>
<td>Extended multiplication and division</td>
<td>75</td>
</tr>
<tr>
<td>FNX</td>
<td>Evaluation of single-variable function</td>
<td>76</td>
</tr>
<tr>
<td>FNXY</td>
<td>Evaluation of two-variables function</td>
<td>78</td>
</tr>
<tr>
<td>GAIN</td>
<td>Multiplication by a constant</td>
<td>80</td>
</tr>
<tr>
<td>GRADS</td>
<td>Gradient search optimization</td>
<td>81</td>
</tr>
<tr>
<td>IADD</td>
<td>Integer addition</td>
<td>83</td>
</tr>
<tr>
<td>ISUB</td>
<td>Integer subtraction</td>
<td>84</td>
</tr>
<tr>
<td>IMUL</td>
<td>Integer multiplication</td>
<td>85</td>
</tr>
<tr>
<td>IDIV</td>
<td>Integer division</td>
<td>87</td>
</tr>
<tr>
<td>IMOD</td>
<td>Remainder after integer division</td>
<td>88</td>
</tr>
<tr>
<td>LIN</td>
<td>Linear interpolation</td>
<td>89</td>
</tr>
<tr>
<td>MUL</td>
<td>Multiplication of two signals</td>
<td>90</td>
</tr>
<tr>
<td>POL</td>
<td>Polynomial evaluation</td>
<td>91</td>
</tr>
<tr>
<td>REC</td>
<td>Reciprocal value</td>
<td>92</td>
</tr>
<tr>
<td>REL</td>
<td>Relational operator</td>
<td>93</td>
</tr>
<tr>
<td>RTOI</td>
<td>Real to integer number conversion</td>
<td>94</td>
</tr>
<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>
<tr>
<td></td>
<td><strong>5 ANALOG</strong> – Analog signal processing</td>
<td>99</td>
</tr>
<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>
</tbody>
</table>
## CONTENTS

<table>
<thead>
<tr>
<th>Block</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</td>
<td>Analog signal selectors</td>
<td>133</td>
</tr>
<tr>
<td>SHIFTOCT</td>
<td>Data shift register</td>
<td>135</td>
</tr>
<tr>
<td>SHLD</td>
<td>Sample and hold</td>
<td>137</td>
</tr>
<tr>
<td>SINT</td>
<td>Simple integrator</td>
<td>138</td>
</tr>
<tr>
<td>SPIKE</td>
<td>Spike filter</td>
<td>139</td>
</tr>
<tr>
<td>SSW</td>
<td>Simple switch</td>
<td>141</td>
</tr>
<tr>
<td>SWR</td>
<td>Selector with ramp</td>
<td>142</td>
</tr>
<tr>
<td>VDEL</td>
<td>Variable time delay</td>
<td>143</td>
</tr>
<tr>
<td>ZV4IS</td>
<td>Zero vibration input shaper</td>
<td>144</td>
</tr>
</tbody>
</table>

### 6 GEN – Signal generators

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANLS</td>
<td>Controlled generator of piecewise linear function</td>
<td>149</td>
</tr>
<tr>
<td>BINS</td>
<td>Controlled binary sequence generator</td>
<td>150</td>
</tr>
<tr>
<td>BIS</td>
<td>Binary sequence generator</td>
<td>152</td>
</tr>
<tr>
<td>MP</td>
<td>Manual pulse generator</td>
<td>154</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>157</td>
</tr>
</tbody>
</table>

### 7 REG – Function blocks for control

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARLY</td>
<td>Advance relay</td>
<td>161</td>
</tr>
<tr>
<td>FLCU</td>
<td>Fuzzy logic controller unit</td>
<td>163</td>
</tr>
<tr>
<td>FRID</td>
<td>Frequency response identification</td>
<td>164</td>
</tr>
<tr>
<td>I3PM</td>
<td>Identification of a three parameter model</td>
<td>166</td>
</tr>
<tr>
<td>LC</td>
<td>Lead compensator</td>
<td>168</td>
</tr>
<tr>
<td>LLC</td>
<td>Lead-lag compensator</td>
<td>170</td>
</tr>
<tr>
<td>MCU</td>
<td>Manual control unit</td>
<td>171</td>
</tr>
<tr>
<td>PIDAT</td>
<td>PID controller with relay autotuner</td>
<td>172</td>
</tr>
<tr>
<td>PIDE</td>
<td>PID controller with defined static error</td>
<td>174</td>
</tr>
</tbody>
</table>
### CONTENTS

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIDGS</td>
<td>PID controller with gain scheduling</td>
<td>179</td>
</tr>
<tr>
<td>PIDMA</td>
<td>PID controller with moment autotuner</td>
<td>181</td>
</tr>
<tr>
<td>PIDU</td>
<td>PID controller unit</td>
<td>187</td>
</tr>
<tr>
<td>PIDUI</td>
<td>PID controller unit with variable parameters</td>
<td>190</td>
</tr>
<tr>
<td>POUT</td>
<td>Pulse output</td>
<td>192</td>
</tr>
<tr>
<td>PRGM</td>
<td>Setpoint programmer</td>
<td>193</td>
</tr>
<tr>
<td>PSMPC</td>
<td>Pulse-step model predictive controller</td>
<td>195</td>
</tr>
<tr>
<td>PWM</td>
<td>Pulse width modulation</td>
<td>199</td>
</tr>
<tr>
<td>RLY</td>
<td>Relay with hysteresis</td>
<td>201</td>
</tr>
<tr>
<td>SAT</td>
<td>Saturation with variable limits</td>
<td>202</td>
</tr>
<tr>
<td>SC2FA</td>
<td>State controller for 2nd order system with frequency autotuner</td>
<td>204</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>217</td>
</tr>
<tr>
<td>SMHCC</td>
<td>Sliding mode heating/cooling controller</td>
<td>218</td>
</tr>
<tr>
<td>SMHCCA</td>
<td>Sliding mode heating/cooling controller with autotuner</td>
<td>222</td>
</tr>
<tr>
<td>SWU</td>
<td>Switch unit</td>
<td>229</td>
</tr>
<tr>
<td>TSE</td>
<td>Three-state element</td>
<td>230</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, ANDQUAD, ANDOCT, ANDHEXD</td>
<td>Logical product of two signals</td>
<td>232</td>
</tr>
<tr>
<td>ATMT</td>
<td>Finite-state automaton</td>
<td>234</td>
</tr>
<tr>
<td>BD OCT, BD HEXD</td>
<td>Bitwise demultiplexers</td>
<td>236</td>
</tr>
<tr>
<td>BITOP</td>
<td>Bitwise operation</td>
<td>237</td>
</tr>
<tr>
<td>BMC OCT, BM HEXD</td>
<td>Bitwise multiplexers</td>
<td>238</td>
</tr>
<tr>
<td>COUNT</td>
<td>Controlled counter</td>
<td>239</td>
</tr>
<tr>
<td>EATMT</td>
<td>Extended finite-state automaton</td>
<td>241</td>
</tr>
<tr>
<td>EDGE, EDGE _</td>
<td>Falling/rising edge detection in a binary signal</td>
<td>244</td>
</tr>
<tr>
<td>INTSM</td>
<td>Integer number bit shift and mask</td>
<td>245</td>
</tr>
<tr>
<td>ISSW</td>
<td>Simple switch for integer signals</td>
<td>246</td>
</tr>
<tr>
<td>INTSM</td>
<td>Integer number bit shift and mask</td>
<td>247</td>
</tr>
<tr>
<td>ITOI</td>
<td>Transformation of integer and binary numbers</td>
<td>248</td>
</tr>
<tr>
<td>NOT, NOT _</td>
<td>Boolean complementation</td>
<td>249</td>
</tr>
<tr>
<td>OR, ORQUAD, OROCT, ORHEXD</td>
<td>Logical sum of two signals</td>
<td>250</td>
</tr>
<tr>
<td>RS</td>
<td>Reset-set flip-flop circuit</td>
<td>252</td>
</tr>
<tr>
<td>SR</td>
<td>Set-reset flip-flop circuit</td>
<td>253</td>
</tr>
<tr>
<td>TIMER, TIMER _</td>
<td>Multipurpose timer</td>
<td>254</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, DATE _</td>
<td>Current date</td>
<td>258</td>
</tr>
<tr>
<td>DATETIME</td>
<td>Get, set and convert time</td>
<td>259</td>
</tr>
<tr>
<td>TIME</td>
<td>Current time</td>
<td>261</td>
</tr>
</tbody>
</table>
CONTENTS

WSCH – Weekly schedule ........................................... 262

10 ARC – Data archiving ........................................... 265
  10.1 Functionality of the archiving subsystem ............... 266
  10.2 Generating alarms and events ......................... 267
      ALB, ALBI – Alarms for Boolean value ................. 267
      ALN, ALNI – Alarms for numerical value ............. 269
  10.3 Trends recording ........................................... 271
      ACD – Archive compression using Delta criterion .... 271
      TRND – Real-time trend recording ................... 273
      TRNDV – Real-time trend recording with vector input 276
  10.4 Archive management ....................................... 277
      AFLUSH – Forced archive flushing .................. 277

11 STRING – Blocks for string operations ................... 279
  CNS – String constant ......................................... 280
  CONCAT – Concat string by pattern .......................... 281
  FIND – Find a Substring ...................................... 282
  ITOS – Integer number to string conversion ................ 283
  LEN – String length ........................................... 284
  MID – Substring Extraction ................................... 285
  PJROCT – Parse JSON string (real output) ................ 286
  PJSOCT – Parse JSON string (string output) .............. 287
  REGEXP – Regular expresion parser .......................... 288
  REPLACE – Replace substring .................................. 291
  RTOS – Real Number to String Conversion .................. 292
  SELSOCT – String selector ..................................... 293
  STOR – String to real number conversion ................... 294

12 PARAM – Blocks for parameter handling .................. 295
  GETPA – Block for remote array parameter acquirement .... 296
  GETPR, GETPI, GETPB – Blocks for remote parameter acquirement 298
  GetsPS – * Block for remote string parameter acquirement 300
  PARA – Block with input-defined array parameter .......... 301
  PAR, PARI, PARB – Blocks with input-defined parameter .... 302
  PARS – * Block with input-defined string parameter ........ 304
  SETPA – Block for remote array parameter setting .......... 305
  SETPR, SETPI, SETPB – Blocks for remote parameter setting 307
  SETPS – * Block for remote string parameter setting ....... 309
  SGSLP – Set, get, save and load parameters ............... 310
  SILO – Save input value, load output value ............... 314
  SILOS – Save input string, load output string ............. 316
13 MODEL – Dynamic systems simulation

CDELSSM – Continuous state space model of a linear system with time delay .................................................. 320
CSSM – Continuous state space model of a linear system ................................................................. 323
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 .............................................................................................. 329
MDL – Process model .............................................................................................................................. 330
MDLI – Process model with input-defined parameters ............................................................................. 331
MDV – Motorized valve drive .................................................................................................................... 332
SOPDT – Second order plus dead-time model ........................................................................................ 333

14 MATRIX – Blocks for matrix and vector operations

CNA – Array (vector/matrix) constant ....................................................................................................... 338
MB_DASUM – Sum of the absolute values ................................................................................................ 339
MB_DAXPY – Performs y := a*x + y for vectors x,y .............................................................................. 340
MB_DCOPY – Copies vector x to vector y .................................................................................................. 342
MB-DDOT – Dot product of two vectors ................................................................................................... 344
MB_DGEMM – Performs C := alpha*op(A)*op(B) + beta*C, where op(X) = X or op(X) = X^T ........... 346
MB_DGEMV – Performs y := alpha*A^T*x + beta*y or y := alpha*A*x + beta*y ................................. 348
MB_DGER – Performs A := alpha*x*y^T + A ......................................................................................... 350
MB_DNRM2 – Euclidean norm of a vector ................................................................................................. 352
MB_DROT – Plain rotation of a vector ........................................................................................................ 353
MB_DSCAL – Scales a vector by a constant ............................................................................................... 355
MB_DSWAP – Interchanges two vectors ..................................................................................................... 356
MB_DTRMM – Performs B := alpha*op(A)*B or B := alpha*B*op(A), where op(X) = X or op(X) = X^T for triangular matrix A ........................................................................................................ 358
MB_DTRMV – Performs x := A*x or x := A^T*x for triangular matrix A ............................................... 360
MB_DTRSV – Solves one of the system of equations A*x = b or A^T*x = b for triangular matrix A ..... 363
ML_DGEBAK – Backward transformation to ML_DGEBAL of left or right eigenvectors ......................... 366
ML_DGEBAL – Balancing of a general real matrix ..................................................................................... 368
ML_DGEBRD – Reduces a general real matrix to bidiagonal form by an orthogonal transformation ...... 370
ML_DGECON – Estimates the reciprocal of the condition number of a general real matrix .................. 372
ML_DGEE – Computes the eigenvalues, the Schur form, and, optionally, the matrix of Schur vectors .... 374
ML_DGEEV – Computes the eigenvalues and, optionally, the left and/or right eigenvectors ................... 376
ML_DGEHRD – Reduces a real general matrix A to upper Hessenberg form ............................................. 378
ML_DGELQF – Computes an LQ factorization of a real M-by-N matrix A ............................................... 380
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ML_DGELSD</td>
<td>Computes the minimum-norm solution to a real linear least squares problem.</td>
<td>382</td>
</tr>
<tr>
<td>ML_DGEEQRF</td>
<td>Computes an QR factorization of a real M-by-N matrix A.</td>
<td>384</td>
</tr>
<tr>
<td>ML_DGESDD</td>
<td>Computes the singular value decomposition (SVD) of a real M-by-N matrix A.</td>
<td>386</td>
</tr>
<tr>
<td>ML_DLACPY</td>
<td>Copies all or part of one matrix to another matrix.</td>
<td>388</td>
</tr>
<tr>
<td>ML_DLANGE</td>
<td>Computes one of the matrix norms of a general matrix.</td>
<td>390</td>
</tr>
<tr>
<td>ML_DLASET</td>
<td>Initializes the off-diagonal elements and the diagonal elements of a matrix to given values.</td>
<td>392</td>
</tr>
<tr>
<td>ML_DTRSYL</td>
<td>Solves the real Sylvester matrix equation for quasi-triangular matrices A and B.</td>
<td>394</td>
</tr>
<tr>
<td>MX_AT</td>
<td>Get Matrix/Vector element.</td>
<td>396</td>
</tr>
<tr>
<td>MX_ATSET</td>
<td>Set Matrix/Vector element.</td>
<td>397</td>
</tr>
<tr>
<td>MX_CNADD</td>
<td>Add scalar to each Matrix/Vector element.</td>
<td>398</td>
</tr>
<tr>
<td>MX_CNMUL</td>
<td>Multiply a Matrix/Vector by a scalar.</td>
<td>399</td>
</tr>
<tr>
<td>MX_CTDQPA</td>
<td>Discretizes continuous model given by (A,B) to (Ad,Bd) using Padé approximations.</td>
<td>400</td>
</tr>
<tr>
<td>MX_DIM</td>
<td>Matrix/Vector dimensions.</td>
<td>402</td>
</tr>
<tr>
<td>MX_DIMSET</td>
<td>Set Matrix/Vector dimensions.</td>
<td>403</td>
</tr>
<tr>
<td>MX_DSAGET</td>
<td>Set subarray of A into B.</td>
<td>404</td>
</tr>
<tr>
<td>MX_DSAREF</td>
<td>Set reference to subarray of A into B.</td>
<td>406</td>
</tr>
<tr>
<td>MX_DSASET</td>
<td>Set A into subarray of B.</td>
<td>408</td>
</tr>
<tr>
<td>MX_DTRNSP</td>
<td>General matrix transposition: B := alpha*A^T.</td>
<td>410</td>
</tr>
<tr>
<td>MX_DTRNSQ</td>
<td>Square matrix in-place transposition: A := alpha*A^T.</td>
<td>412</td>
</tr>
<tr>
<td>MX_FILL</td>
<td>Fill real matrix or vector.</td>
<td>413</td>
</tr>
<tr>
<td>MX_MAT</td>
<td>Matrix data storage block.</td>
<td>414</td>
</tr>
<tr>
<td>MX_RAND</td>
<td>Randomly generated matrix or vector.</td>
<td>415</td>
</tr>
<tr>
<td>MX_REFCOPY</td>
<td>Copies input references of matrices A and B to their output references.</td>
<td>417</td>
</tr>
<tr>
<td>MX_VEC</td>
<td>Vector data storage block.</td>
<td>418</td>
</tr>
<tr>
<td>MX_WRITE</td>
<td>Write a Matrix/Vector to the console/system log.</td>
<td>419</td>
</tr>
<tr>
<td>RTOV</td>
<td>Vector multiplexer.</td>
<td>421</td>
</tr>
<tr>
<td>SWVMR</td>
<td>Vector/matrix/reference signal switch.</td>
<td>422</td>
</tr>
<tr>
<td>VTOR</td>
<td>Vector demultiplexer.</td>
<td>423</td>
</tr>
</tbody>
</table>

### 15 SPEC – Special blocks

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPC</td>
<td>External program call.</td>
<td>426</td>
</tr>
<tr>
<td>HTTP</td>
<td>HTTP GET or POST request (obsolete).</td>
<td>429</td>
</tr>
<tr>
<td>HTTP2</td>
<td>Block for generating HTTP GET or POST requests.</td>
<td>431</td>
</tr>
<tr>
<td>SMTP</td>
<td>Send e-mail message via SMTP.</td>
<td>433</td>
</tr>
<tr>
<td>RDC</td>
<td>Remote data connection.</td>
<td>435</td>
</tr>
<tr>
<td>REXLANG</td>
<td>User programmable block.</td>
<td>439</td>
</tr>
</tbody>
</table>
## CONTENTS

### 16 MQTT – Communication via MQTT protocol

- **MqttPublish** – Publish MQTT message ........................................ 458
- **MqttSubscribe** – Subscribe to MQTT topic .................................... 460

### 17 CanDrv – Communication via CAN bus

- **CanItem** – Secondary received CAN message ............................... 464
- **CanRecv** – Receive CAN message .................................................. 465
- **CanSend** – Send CAN message ...................................................... 467

### 18 MC_SINGLE – Motion control - single axis blocks

- **RM_Axis** – Motion control axis ..................................................... 472
- **MC_AccelerationProfile, MCP_AccelerationProfile** – Acceleration profile .................................................. 478
- **MC_Halt, MCP_Halt** – Stopping a movement (interruptible) ............. 482
- **MC_HaltSuperimposed, MCP_HaltSuperimposed** – Stopping a movement (superimposed and interruptible) ......................... 483
- **MC_Home, MCP_Home** – Homing .................................................... 484
- **MC_MoveAbsolute, MCP_MoveAbsolute** – Move to position (absolute coordinate) .................................................. 486
- **MC_MoveAdditive, MCP_MoveAdditive** – Move to position (relative to previous motion) .................................................. 489
- **MC_MoveRelative, MCP_MoveRelative** – Move to position (relative to execution point) .................................................. 492
- **MC_MoveSuperimposed, MCP_MoveSuperimposed** – Superimposed move .................................................. 495
- **MC_MoveContinuousAbsolute, MCP_MoveContinuousAbsolute** – Move to position (absolute coordinate) .................................................. 498
- **MC_MoveContinuousRelative, MCP_MoveContinuousRelative** – Move to position (relative to previous motion) .................................................. 501
- **MC_MoveVelocity, MCP_MoveVelocity** – Move with constant velocity .................................................. 505
- **MC_PositionProfile, MCP_PositionProfile** – Position profile ........ 509
- **MC_Power** – Axis activation (power on/off) .................................... 513
- **MC_ReadActualPosition** – Read actual position ............................ 514
- **MC_ReadAxisError** – Read axis error ............................................. 515
- **MC_ReadBoolParameter** – Read axis parameter (bool) ................. 516
- **MC_ReadParameter** – Read axis parameter ..................................... 517
- **MC_ReadStatus** – Read axis status ............................................. 519
- **MC_Reset** – Reset axis errors ..................................................... 521
- **MC_SetOverride, MCP_SetOverride** – Set override factors ............ 522
- **MC_Stop, MCP_Stop** – Stopping a movement .................................. 524
- **MC_TorqueControl, MCP_TorqueControl** – Torque/force control ..... 526
- **MC_VelocityProfile, MCP_VelocityProfile** – Velocity profile .......... 529
- **MC_WriteBoolParameter** – Write axis parameter (bool) .............. 533
- **MC_WriteParameter** – Write axis parameter .................................. 534
- **RM_AxisOut** – Axis output ..................................................... 535
## RM_AxisSpline – Commanded values interpolation
537

## RM_Track – Tracking and inching
539

### 19 MC_MULTI – Motion control - multi axis blocks
541
- **MC_CamIn, MCP_CamIn** – Engage the cam
542
- **MC_CamOut** – Disengage the cam
545
- **MCP_CamTableSelect** – Cam definition
547
- **MC_CombineAxes, MCP_CombineAxes** – Combine the motion of 2 axes into a third axis
549
- **MC_GearIn, MCP_GearIn** – Engage the master/slave velocity ratio in defined position
552
- **MC_GearInPos, MCP_GearInPos** – Engage the master/slave velocity ratio
555
- **MC_GearOut** – Disengage the master/slave velocity ratio
560
- **MC_PhasingAbsolute, MCP_PhasingAbsolute** – Phase shift in synchronized motion (absolute coordinates)
562
- **MC_PhasingRelative, MCP_PhasingRelative** – Phase shift in synchronized motion (relative coordinates)
565

### 20 MCCOORD – Motion control - coordinated movement blocks
567
- **RM_AxesGroup** – Axes group for coordinated motion control
570
- **RM_Feed** – * MC Feeder
573
- **RM_Gcode** – * CNC motion control
574
- **MC_AddAxisToGroup** – Adds one axis to a group
576
- **MC_UngroupAllAxes** – Removes all axes from the group
577
- **MC_GroupEnable** – Changes the state of a group to GroupEnable
578
- **MC_GroupDisable** – Changes the state of a group to GroupDisabled
579
- **MC_SetCartesianTransform** – Sets Cartesian transformation
580
- **MC_ReadCartesianTransform** – Reads the parameter of the cartesian transformation
582
- **MC_GroupSetPosition, MCP_GroupSetPosition** – Sets the position of all axes in a group
583
- **MC_GroupReadActualPosition** – Read actual position in the selected coordinate system
585
- **MC_GroupReadActualVelocity** – Read actual velocity in the selected coordinate system
586
- **MC_GroupReadActualAcceleration** – Read actual acceleration in the selected coordinate system
587
- **MC_GroupStop** – Stopping a group movement
588
- **MC_GroupHalt** – Stopping a group movement (interruptible)
591
- **MC_GroupInterrupt, MCP_GroupInterrupt** – Read a group interrupt
596
- **MC_GroupContinue** – Continuation of interrupted movement
597
- **MC_GroupReadStatus** – Read a group status
598
- **MC_GroupReadError** – Read a group error
599
- **MC_GroupReset** – Reset axes errors
600
MC_MoveLinearAbsolute — Linear move to position (absolute coordinates) 601
MC_MoveLinearRelative — Linear move to position (relative to execution point) 605
MC_MoveCircularAbsolute — Circular move to position (absolute coordinates) 609
MC_MoveCircularRelative — Circular move to position (relative to execution point) 613
MC_MoveDirectAbsolute — Direct move to position (absolute coordinates) 617
MC_MoveDirectRelative — Direct move to position (relative to execution point) 620
MC_MovePath — General spatial trajectory generation 623
MC_GroupSetOverride — Set group override factors 625

A Licensing options 627
B Licensing of individual function blocks 629
C Error codes of the REXYGEN system 639

Bibliography 645
Index 647

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

Introduction

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

1.1 How to use this manual

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

The content of individual chapters of this manual is following:

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

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

3 INOUT – Input and output blocks
   This sublibrary consists of the blocks used mainly for the REXYGEN system. These blocks provide the connection between the control tasks and input/output drivers.
4 **MATH** – Mathematic blocks
The blocks for arithmetic operations and basic math functions. Similar blocks can be found in native Simulink libraries, but only blocks from this library can be used for applications whose target platform is the REXYGEN control system.

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

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

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

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

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

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

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

14 **MATRIX** – Working with matrix and vector data
Function blocks for handling vector and matrix data in REXYGEN are included in this sublibrary.
1.2. THE FUNCTION BLOCK DESCRIPTION FORMAT

**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.

**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.

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

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

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

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

1.2 The function block description format

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

**Block Symbol** – displays the graphical symbol of the block

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

**Inputs** – detailed description of all inputs of the block

**Outputs** – detailed description of all outputs of the block

**Parameters** – detailed description of all parameters of the block

**Examples** – a simple example of the use of the block in the context of other blocks and optional graph with input and output signals for better understanding of the block function.
If the block function is obvious, the section Examples is omitted. In case of block with no input or no output the corresponding section is omitted as well.

The inputs, outputs and parameters description has a tabular form:

<table>
<thead>
<tr>
<th>&lt;name&gt;</th>
<th>nam</th>
<th>Detailed description of the input (output, parameter) &lt;name&gt;. &lt;type&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mathematical symbol nam on the right side of the first column is used in the equations in the Function Description section. It is listed only if it differs from the name more than typographically. If the variable value is limited to only enumerated values, the meaning of these values is explained in this column.</td>
</tr>
</tbody>
</table>

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

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

1.3 Conventions for variables, blocks and subsystems naming

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

- **bool** – for two-state logic variables, e.g. on/off, yes/no or true/false. The logic one (yes,
true, on) is referred to as on in this manual. Similarly the logic zero (no, false, off) is represented by off. Nevertheless, some tools may display these values as on for 1 and off for 0 for Matlab-Simulink compatibility reasons. The names of logic variables consist of uppercase letters, e.g. \textit{RUN}, \textit{YCN}, \textit{R1}, \textit{UP}, etc.

\textbf{\textit{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. \textit{ips}, \textit{l}, \textit{isig}, \textit{iE}, etc. But several exceptions to this rule exist, e.g. \textit{cnt} in the \textit{COUNT} block, \textit{btype}, \textit{ptypel}, \textit{pfac} and \textit{afac} in the \textit{TRND} block, etc.

\textbf{\textit{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. \textit{k}, \textit{hilim}, \textit{y}, \textit{ti}, \textit{tt}.

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

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

\section*{1.4 The signal quality corresponding with OPC}

Every signal (input, output, parameter) in the \textit{REXYGEN} system has the so-called \textit{quality flags} in addition to its own value of corresponding type (table 1.1). The quality flags in the \textit{REXYGEN} system correspond with the OPC (OLE for Process Control) specification \cite{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</td>
<td>128</td>
<td>BAD</td>
<td>0 0</td>
<td>S S S S</td>
<td>L L</td>
</tr>
<tr>
<td>0</td>
<td>64</td>
<td>UNCERTAIN</td>
<td>0 1</td>
<td>S S S S</td>
<td>L L</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
<td>not used in OPC</td>
<td>1 0</td>
<td>S S S S</td>
<td>L L</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>GOOD</td>
<td>1 1</td>
<td>S S S S</td>
<td>L L</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></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 \textbf{GOOD}, \textbf{UNCERTAIN} and \textbf{BAD}. The
four SSSS bits provide more detailed information about the signal. They have different meaning for each basic quality. The two least significant bits LL inform whether the value exceeded its limits or if it is constant. Additional details and the meaning of all bits can be found in [1], chapter 6.8.
Chapter 2

EXEC – Real-time executive configuration

Contents

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

Function Description

The ARC block is intended for archives configuration in the REXYGEN control system. The archives can be used for continuous recording of alarms, events and history trends directly on the target platform. The output Archives of the EXEC block must be connected to the prev input of the first archive. The following archives can be added by connecting the input prev with the preceding archive’s output next. Only one archive block can be connected to each next output, the output of the last archive remains unconnected.

The resulting archives sequence determines the order of allocation and initialization of individual archives in the REXYGEN system and also the index of the archive, which is used in the arc parameter of the archiving blocks (see chapter 10). The archives are numbered from 1 and the maximum number of archives is limited to 15 (archive no. 0 is the internal system log).

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

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

Input

prev Input for connecting with the next output of the preceding 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>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>atype</td>
<td>Archive type</td>
<td>1</td>
<td></td>
<td>long</td>
</tr>
<tr>
<td>1</td>
<td>. . . .</td>
<td>archive is allocated in the RAM memory (data is irreversibly lost after restarting the target device)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>. . . .</td>
<td>archive is allocated in backed-up memory, e.g. CMOS (data remains available after restarting the target device)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>. . . .</td>
<td>archive is allocated on a drive (data remains available in the file after restarting)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>asize</td>
<td>Size of the archive in bytes</td>
<td>256</td>
<td>102400</td>
<td>long</td>
</tr>
<tr>
<td>nmarks</td>
<td>Number of time stamps for speeding-up sequential seeking in the archive</td>
<td>2</td>
<td>720</td>
<td>long</td>
</tr>
<tr>
<td>ldymax</td>
<td>Maximum size of archive per day [bytes]</td>
<td>1000</td>
<td>2147480000</td>
<td>large</td>
</tr>
<tr>
<td>period</td>
<td>Period of writing data to disk [s]</td>
<td>60.0</td>
<td>double</td>
<td></td>
</tr>
</tbody>
</table>
EXEC – Real-time executive

Block Symbol

Function Description

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

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

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

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

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

**Outputs**

<table>
<thead>
<tr>
<th>Modules</th>
<th>Output for connecting the REXYGEN system expansion modules, see the <code>MODULE</code> function block description for details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drivers</td>
<td>Output for connecting the REXYGEN system input/output drivers, see the <code>I0DRV</code> and <code>T0DRV</code> function block descriptions for details</td>
</tr>
<tr>
<td>Archives</td>
<td>Output for archives configuration, see the <code>ARC</code> block</td>
</tr>
<tr>
<td>QTask</td>
<td>Output for connecting quick tasks with the highest priority and the shortest period, see the <code>QTASK</code> block</td>
</tr>
<tr>
<td>Level0</td>
<td>Computation level for inserting tasks (see the <code>TASK</code> block) with high priority pri0 and short period determined by the ntick0 parameter</td>
</tr>
<tr>
<td>Level1</td>
<td>Computation level for inserting tasks with medium priority pri1 and medium-length period determined by the ntick1 parameter</td>
</tr>
<tr>
<td>Level2</td>
<td>Computation level for inserting tasks with low priority pri2 and long period determined by the ntick2 parameter</td>
</tr>
<tr>
<td>Level3</td>
<td>Computation level for inserting tasks with the lowest priority pri3 and the longest period determined by the ntick3 parameter</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>target</th>
<th>Target device</th>
<th>○PC - Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>tick</td>
<td>The base period (tick) of the REXYGEN system core and also the quick task (QTASK) period (in seconds) ○0.05 double</td>
<td></td>
</tr>
<tr>
<td>ntick0</td>
<td>The multiplication tick*ntick0 determines the period of tasks connected to Level0 ↓1 ○10 long</td>
<td></td>
</tr>
<tr>
<td>ntick1</td>
<td>The multiplication tick*ntick1 determines the period of tasks connected to Level1 ↓ntick0+1 ○50 long</td>
<td></td>
</tr>
<tr>
<td>ntick2</td>
<td>The multiplication tick*ntick2 determines the period of tasks connected to Level2 ↓ntick1+1 ○100 long</td>
<td></td>
</tr>
<tr>
<td>ntick3</td>
<td>The multiplication tick*ntick3 determines the period of tasks connected to Level3 ↓ntick2+1 ○1200 long</td>
<td></td>
</tr>
<tr>
<td>pri0</td>
<td>Priority of all Level0 tasks ↓3 ↑31 ○5 long</td>
<td></td>
</tr>
<tr>
<td>pri1</td>
<td>Priority of all Level1 tasks ↓pri0+1 ↑31 ○9 long</td>
<td></td>
</tr>
<tr>
<td>pri2</td>
<td>Priority of all Level2 tasks ↓pri1+1 ↑31 ○13 long</td>
<td></td>
</tr>
<tr>
<td>pri3</td>
<td>Priority of all Level3 tasks ↓pri2+1 ↑31 ○18 long</td>
<td></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 REXGEN system currently provides three straightforward methods of how to create Human-Machine Interface:

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

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

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

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

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

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

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

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

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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IncludeHMI</td>
<td>Include HMI files in the project</td>
<td>on</td>
<td>bool</td>
</tr>
<tr>
<td>HmiDir</td>
<td>Output folder for HMI files</td>
<td>hmi</td>
<td>string</td>
</tr>
<tr>
<td>SourceDir</td>
<td>Source directory</td>
<td>hmisrc</td>
<td>string</td>
</tr>
<tr>
<td>GenerateWebWatch</td>
<td>Generate WebWatch HMI files</td>
<td>on</td>
<td>bool</td>
</tr>
<tr>
<td>GenerateRexHMI</td>
<td>Generate HMI from SVG and JS files</td>
<td>on</td>
<td>bool</td>
</tr>
<tr>
<td>RedirectToHMI</td>
<td>Web server will automatically redirect to HMI webpage if enabled</td>
<td>off</td>
<td>bool</td>
</tr>
<tr>
<td>Compression</td>
<td>Enables data compression in gzip format.</td>
<td></td>
<td>bool</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

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

#### Input

<table>
<thead>
<tr>
<th>prev</th>
<th>Input for connecting the driver with the <strong>Drivers</strong> output of the <strong>EXEC</strong> block or with the <strong>next</strong> output of the preceding driver</th>
</tr>
</thead>
</table>

#### Output

<table>
<thead>
<tr>
<th>next</th>
<th>Output for connecting to the <strong>prev</strong> input of the succeeding driver</th>
</tr>
</thead>
</table>

#### Parameters

<table>
<thead>
<tr>
<th>module</th>
<th>Name of the module, which includes the input/output driver (mandatory only if module name differs from <strong>classname</strong>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>classname</td>
<td>I/O driver class name; case sensitive!</td>
</tr>
<tr>
<td>cfgname</td>
<td>Name of the driver configuration file</td>
</tr>
<tr>
<td>factor</td>
<td>Multiple of the <strong>EXEC</strong> block’s <strong>tick</strong> parameter defining the driver’s task execution period</td>
</tr>
<tr>
<td>stack</td>
<td>Stack size of the driver’s task in bytes</td>
</tr>
<tr>
<td>pri</td>
<td>Logical priority of the driver’s task</td>
</tr>
<tr>
<td>timer</td>
<td>Driver is a source of time</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DrvClass</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>iodrv.rio</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>bool</td>
</tr>
</tbody>
</table>
IOTASK – Driver-triggered task of the REXYGEN system

Block Symbol

Licence: STANDARD

Function Description

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

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

Input

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>factor</td>
<td>Execution factor which can be used to determine the task execution period, see the user guide of the corresponding I/O driver</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 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)</td>
<td>string</td>
</tr>
</tbody>
</table>
LPBRK – Loop break

Block Symbol

Licence: STANDARD

Function Description

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

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

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

Input

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

Output

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

Block Symbol

<table>
<thead>
<tr>
<th>prev</th>
<th>next</th>
</tr>
</thead>
</table>

Licence: **STANDARD**

**Function Description**

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

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

**Input**

<table>
<thead>
<tr>
<th>prev</th>
<th>Input for connecting the module with the <em>Modules</em> output of the <strong>EXEC</strong> block or with the <em>next</em> output of the preceding module</th>
</tr>
</thead>
</table>

**Output**

<table>
<thead>
<tr>
<th>next</th>
<th>Output for connecting to the <em>prev</em> input of the succeeding module</th>
</tr>
</thead>
</table>

| long | |
PROJECT – Additional Project Settings

Block Symbol: PROJECT

Licence: STANDARD

Function Description

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

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

Parameters

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

Block Symbol

Licence: STANDARD

Function Description

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

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

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

Input

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>
</tr>
</thead>
<tbody>
<tr>
<td>factor</td>
<td>Multiple of the EXEC block’s tick parameter defining the quick task execution period</td>
</tr>
<tr>
<td>stack</td>
<td>Stack size [bytes]</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>
</tr>
</tbody>
</table>

factor long

stack long

filename string
SLEEP – Timing in Simulink

Block Symbol  Licence: STANDARD

Function Description

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

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

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

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ts</td>
<td>Simulation scheme execution period (in seconds)</td>
<td>≥ 0.1</td>
<td>double</td>
</tr>
</tbody>
</table>
SRTF – Set run-time flags

Block Symbol

Function Description

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

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

The run-time flags allow the following operations:

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

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

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

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

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

<table>
<thead>
<tr>
<th>EXDIS</th>
<th>Disable execution</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXOSH</td>
<td>One-shot execution</td>
<td>bool</td>
</tr>
<tr>
<td>DGEN</td>
<td>Enable diagnostics</td>
<td>bool</td>
</tr>
<tr>
<td>DGRES</td>
<td>Reset diagnostic data</td>
<td>bool</td>
</tr>
<tr>
<td>DLOG</td>
<td>Enable more verbose logging</td>
<td>bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>E</th>
<th>Error flag</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<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>

<table>
<thead>
<tr>
<th>iE</th>
<th>Error code (for E = on)</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>... No error</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>... The object specified by the bname parameter was not found</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>... REXYGEN system internal error (invalid pointers)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>... Flag could not be set (timeout)</td>
<td></td>
</tr>
</tbody>
</table>

Parameter

<table>
<thead>
<tr>
<th>bname</th>
<th>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:</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>Computational level, e.g. ^0 for Level0</td>
</tr>
<tr>
<td>&amp;</td>
<td>Input/Output Driver, e.g. &amp;&lt;driver_name&gt;</td>
</tr>
</tbody>
</table>

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

**Block Symbol**

![OSCALL block symbol]

**Licence:** STANDARD

**Function Description**

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

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

**Input**

<table>
<thead>
<tr>
<th>TRG</th>
<th>Trigger of the selected action</th>
<th>bool</th>
</tr>
</thead>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>E</th>
<th>Error flag</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>iE</td>
<td>Error code</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>i . . . .</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REXYGEN general error</td>
<td></td>
</tr>
</tbody>
</table>

**Parameter**

<table>
<thead>
<tr>
<th>action</th>
<th>System function to perform</th>
<th></th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reboot system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>System shutdown</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>System halt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Flush disc caches</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Lock system partition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Unlock system partition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Disable internal webserver</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Enable internal webserver</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 2. EXEC – REAL-TIME EXECUTIVE CONFIGURATION

TASK – Standard task of the REXYGEN system

Block Symbol

Function Description

The overall control algorithm of the REXYGEN system consists of individual tasks. These are included by using the TASK block. There can be one or more tasks in the control algorithm. The REXYGEN system contains four main computational levels represented by the Level0 to Level3 outputs of the EXEC block. The individual tasks are added to the given computational level \(<i>\) by connecting the prev input with the corresponding Level<i> output or with the next output of a TASK, which is already included in the given level \(<i>\). There can be only one task connected to the next output of the TASK block. The next output of the last task in the given level remains unconnected. This means that the tasks in one level create a unidirectional chain which defines the order of initialization and execution of the individual tasks of the given level in the REXYGEN system. The individual levels are ordered from Level0 to Level3 (the QTASK block precedes Level0).

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

Input

prev Input for connecting the task with the corresponding Level<i> 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: $\text{factor} \times \text{tick} \times \text{ntick}_i$

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

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

**stack**  
Stack size [bytes]

**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)
TIODRV – The REXYGEN system input/output driver with tasks

Block Symbol Licence: STANDARD

Function Description

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

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

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

Input

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module</td>
<td>Name of the module, which includes the input/output driver (mandatory only if module name differs from classname)</td>
</tr>
<tr>
<td>classname</td>
<td>Name of the driver class; case sensitive!</td>
</tr>
<tr>
<td>cfgname</td>
<td>Name of the driver configuration file</td>
</tr>
<tr>
<td>factor</td>
<td>Multiple of the EXEC block’s tick parameter defining the driver’s task execution period</td>
</tr>
</tbody>
</table>

Note: DrvClass and iodrv.rio are case sensitive.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack</td>
<td>Stack size of the driver’s task in bytes</td>
<td>( \downarrow 1024 \uparrow 10240 ) long</td>
</tr>
<tr>
<td>pri</td>
<td>Logical priority of the driver’s task</td>
<td>( \downarrow 1 \uparrow 31 ) long</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 and so it belongs to the EXEC category.

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

Parameters

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

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

Compression  Enables data compression in gzip format.
Chapter 3

INOUT – Input and output blocks

Contents

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

Block Symbol

Function Description

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

Input

\( u \) Input signal unknown

Parameters

<table>
<thead>
<tr>
<th>Format</th>
<th>Format of displayed value</th>
<th>unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best fit</td>
<td>○1 long</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

Licence: STANDARD

Function Description

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

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

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

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

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

<DRV>___<signal>

E.g. the first digital input of a Modbus I/O device might be referenced by MBM__DI1. Detailed information about signal naming can be found in the user manual of the corresponding I/O driver.
Since version 2.50.5 it is possible to use placeholders in names of I/O driver signals. This is useful inside subsystems where this placeholder is replaced by the value of subsystem parameter. E.g. the flag MBM_Di$id<$id> will refer to digital input 1, 2, 3 etc. depending on the parameter id of the subsystem the block is contained in. See the SubSystem function block for information on defining subsystem parameters.

Output

value Signal coming from I/O driver or Goto block. The type of output is 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 REXYGEN I/O driver, which should provide data through the block’s output.
Goto, OUTSTD – Signal source or output

Block Symbols

Licence: STANDARD

Function Description

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

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

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

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

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

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

<DRV>__<signal>
E.g. the first digital output of a Modbus I/O device might be referenced by `MBM__DO1`. Detailed information about signal naming can be found in the user manual of the corresponding I/O driver.

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

**Input**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GotoTag</td>
<td>Reference to a From block with the same GotoTag parameter, which should be connected with the Goto block or a reference to output signal of the REXYGEN control system driver, which should send the data from block input to the process.</td>
</tr>
<tr>
<td>TagVisibility</td>
<td>Visibility (availability) of the block within the .mdl file. Defines conditions under which the two corresponding Goto and From blocks are reciprocally available:</td>
</tr>
<tr>
<td></td>
<td>local the two blocks must be in the same subsystem</td>
</tr>
<tr>
<td></td>
<td>global blocks can be anywhere in the given task (.mdl file)</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
</tbody>
</table>
GotoTagVisibility – Visibility of the signal source

Block Symbol

Function Description

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

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

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

Parameter

GotoTag Reference to a Goto block with the GotoTag parameter, whose string visibility is defined by the position of this block (GotoTagVisibility)
CHAPTER 3. INOUT – INPUT AND OUTPUT BLOCKS

Inport, Outport – Input and output port

Block Symbols

License: STANDARD

Function Description

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

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

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

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

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

Input

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

Output

<p>| value | Value coming from the input pin or Outport | unknown |</p>
<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**

**Function Description**

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

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

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

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

Please refer to [3] for details on using subsystems and creating reusable components in REXYGEN.

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

**Inputs**

The ordering and names of the inputs are given by the numbers and names of the Import blocks contained within the subsystem. See REXYGEN Studio manual [3] for details.

**Outputs**
The ordering and names of the outputs are given by the numbers and names of the Outport blocks contained within the subsystem. See REXYGEN Studio manual [3] for details.

Parameters

The parameters of the subsystem are defined by the so-called subsystem mask. See REXYGEN Studio manual [3] for details.
INQUAD, INOCT, INHEXD – Multi-input blocks

Function Description

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

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

<DRV>__<signal>

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

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

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

Outputs

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

Function Description

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

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

<DRV>_<signal>

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

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

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

Inputs

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

Block Symbols

Licence: ADVANCED

Function Description

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

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

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

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

Inputs

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

Outputs

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

**Block Symbol**

![OUTRSTD Block Symbol]

**Licence:** ADVANCED

**Function Description**

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

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

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

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

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

**Input**

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

**Output**

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

Block Symbol

Function Description

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

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

Inputs

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

Output

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( iqf )</td>
<td>Bit combination of the ( iq, is ) and ( il ) input signals</td>
<td>long</td>
</tr>
</tbody>
</table>
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 REXYGEN system. Further details about quality flags can be found in chapter 1.4 of this manual. The RexLib function block library for Matlab-Simulink does not use any quality flags.

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

Input

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

Outputs

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

Function Description

The VIN block can be used for verification of the input signal quality in the REXYGEN system. Further details about quality flags can be found in chapter 1.4 of this manual. The RexLib function block library for Matlab-Simulink does not use any quality flags.

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

- For \( GU = \text{off} \) 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 \( QG = \text{on} \). 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 \text{unknown}

Outputs

- \( yg \) Validated output signal (\( yg = u \) for \( QG = \text{on} \) or \( yg = sv \) for \( QG = \text{off} \)) \text{unknown}
- \( QG \) 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}
  - \( off \ldots \) Uncertain quality unacceptable
  - \( on \ldots \) Uncertain quality acceptable
VOUT – Validation of the output signal

Block Symbol Licence: ADVANCED

Function Description

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

Inputs

- 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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS_</td>
<td>Absolute value</td>
<td>65</td>
</tr>
<tr>
<td>ADD</td>
<td>Addition of two signals</td>
<td>66</td>
</tr>
<tr>
<td>ADDQUAD, ADDOCT, ADDHEXD</td>
<td>Multi-input addition</td>
<td>67</td>
</tr>
<tr>
<td>CNB</td>
<td>Boolean (logic) constant</td>
<td>68</td>
</tr>
<tr>
<td>CNE</td>
<td>Enumeration constant</td>
<td>69</td>
</tr>
<tr>
<td>CNI</td>
<td>Integer constant</td>
<td>70</td>
</tr>
<tr>
<td>CNR</td>
<td>Real constant</td>
<td>71</td>
</tr>
<tr>
<td>DIF_</td>
<td>Difference</td>
<td>72</td>
</tr>
<tr>
<td>DIV</td>
<td>Division of two signals</td>
<td>73</td>
</tr>
<tr>
<td>EAS</td>
<td>Extended addition and subtraction</td>
<td>74</td>
</tr>
<tr>
<td>EMD</td>
<td>Extended multiplication and division</td>
<td>75</td>
</tr>
<tr>
<td>FNX</td>
<td>Evaluation of single-variable function</td>
<td>76</td>
</tr>
<tr>
<td>FNXY</td>
<td>Evaluation of two-variables function</td>
<td>78</td>
</tr>
<tr>
<td>GAIN</td>
<td>Multiplication by a constant</td>
<td>80</td>
</tr>
<tr>
<td>GRADS</td>
<td>Gradient search optimization</td>
<td>81</td>
</tr>
<tr>
<td>IADD</td>
<td>Integer addition</td>
<td>83</td>
</tr>
<tr>
<td>ISUB</td>
<td>Integer subtraction</td>
<td>84</td>
</tr>
<tr>
<td>IMUL</td>
<td>Integer multiplication</td>
<td>85</td>
</tr>
<tr>
<td>IDIV</td>
<td>Integer division</td>
<td>87</td>
</tr>
<tr>
<td>IMOD</td>
<td>Remainder after integer division</td>
<td>88</td>
</tr>
<tr>
<td>LIN</td>
<td>Linear interpolation</td>
<td>89</td>
</tr>
<tr>
<td>MUL</td>
<td>Multiplication of two signals</td>
<td>90</td>
</tr>
<tr>
<td>POL</td>
<td>Polynomial evaluation</td>
<td>91</td>
</tr>
<tr>
<td>REC</td>
<td>Reciprocal value</td>
<td>92</td>
</tr>
<tr>
<td>REL</td>
<td>Relational operator</td>
<td>93</td>
</tr>
<tr>
<td>RTOI</td>
<td>Real to integer number conversion</td>
<td>94</td>
</tr>
<tr>
<td>Block</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------</td>
<td>------</td>
</tr>
<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 \texttt{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 \texttt{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 \hspace{1cm} \text{double}

Outputs

- \( y \) Absolute value of the input signal \hspace{1cm} \text{double}
- \( \text{sgn} \) Indication of the input signal sign \hspace{1cm} \text{long}
ADD – Addition of two signals

Block Symbol

Licence: STANDARD

Function Description

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

\[ y = u_1 + u_2. \]

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

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>Sum of the input signals</td>
<td>double</td>
</tr>
</tbody>
</table>
ADDQUAD, ADDOCT, ADDHEXD – Multi-input addition

Block Symbols

Licence: STANDARD

Function Description

The ADDQUAD, ADDOCT and ADDHEXD blocks sum (or subtract) up to 16 input signals. The \( n_l \) parameter defines the inputs which are subtracted instead of adding. For an empty \( n_l \) parameter the block output is given by \( y = u_1 + u_2 + u_3 + u_4 + u_5 + u_6 + u_7 + \ldots + u_{16} \).

For e.g. \( n_l = 2,5,7 \), the block implements the function \( y = u_1 - u_2 + u_3 + u_4 - u_5 + u_6 - u_7 + \ldots + u_{16} \).

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

Inputs

\begin{itemize}
  \item \textit{u1..u16} Analog input signals \textbf{double}
\end{itemize}

Output

\begin{itemize}
  \item \textit{y} Resulting value \textbf{double}
\end{itemize}

Parameter

\begin{itemize}
  \item \textit{n1} 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. \textbf{long}
\end{itemize}
**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 Licence: STANDARD

Function Description

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

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

Parameters

- `yenum` Enumeration constant $\odot 1$: option A string
- `pupstr` Popup list definition $\odot 1$: option A $\mid 2$: option B $\mid 3$: option C string

Output

- `iy` Integer output of the block long
CNI – Integer constant

Block Symbol

Licence: STANDARD

Function Description
The CNI block stands for an integer constant.

Output
\( iy \) Integer output of the block long

Parameter
\( icn \) Integer constant ⊗1 long
CNR – Real constant

Block Symbol

Function Description
The CNR block stands for a real constant.

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

Parameter
- \( ycn \) Real constant \( \circ 1.0 \) \( \text{double} \)
CHAPTER 4. MATH – MATH BLOCKS

DIF_ – Difference

Block Symbol

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

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

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

Input

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

Block Symbol Licence: STANDARD

Function Description
The DIV block divides two analog input signals \( y = \frac{u1}{u2} \). In case \( u2 = 0 \), the output \( E \) is set to on and the output \( y \) is substituted by \( y = yerr \).

Inputs
- \( u1 \) First analog input of the block double
- \( u2 \) Second analog input of the block double

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

Parameter
- \( yerr \) Substitute value for an error case ⊙1.0 double
EAS – Extended addition and subtraction

Function Description
The EAS block sums input analog signals \( u_1, u_2, u_3, u_4 \) with corresponding weights \( a, b, c, d \). 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, \( \text{double} \)
- \( u_2 \): Second analog input of the block, \( \text{double} \)
- \( u_3 \): Third analog input of the block, \( \text{double} \)
- \( u_4 \): Fourth analog input of the block, \( \text{double} \)

Output
- \( y \): Analog output of the block, \( \text{double} \)

Parameters
- \( a \): Weighting coefficient of the \( u_1 \) input, \( \circ1.0 \) \( \text{double} \)
- \( b \): Weighting coefficient of the \( u_2 \) input, \( \circ1.0 \) \( \text{double} \)
- \( c \): Weighting coefficient of the \( u_3 \) input, \( \circ1.0 \) \( \text{double} \)
- \( d \): Weighting coefficient of the \( u_4 \) input, \( \circ1.0 \) \( \text{double} \)
- \( y_0 \): Additive constant (bias), \( \text{double} \)
**EMD – Extended multiplication and division**

**Block Symbol**

<table>
<thead>
<tr>
<th>u1</th>
<th>u2</th>
<th>u3</th>
<th>u4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Licence:** STANDARD

**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  
  - Type: double

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

- \( u_3 \): Third analog input of the block  
  - Type: double

- \( u_4 \): Fourth analog input of the block  
  - Type: double

**Outputs**

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

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

**Parameters**

- \( a \): Weighting coefficient of the \( u_1 \) input  
  - Value: \( 1.0 \)  
  - Type: double

- \( a_0 \): Additive constant for \( u_1 \) input  
  - Type: double

- \( b \): Weighting coefficient of the \( u_2 \) input  
  - Value: \( 1.0 \)  
  - Type: double

- \( b_0 \): Additive constant for \( u_2 \) input  
  - Type: double

- \( c \): Weighting coefficient of the \( u_3 \) input  
  - Value: \( 1.0 \)  
  - Type: double

- \( c_0 \): Additive constant for \( u_3 \) input  
  - Type: double

- \( d \): Weighting coefficient of the \( u_4 \) input  
  - Value: \( 1.0 \)  
  - Type: double

- \( d_0 \): Additive constant for \( u_4 \) input  
  - Type: double

- \( y_{err} \): Substitute value for an error case  
  - Value: \( 1.0 \)  
  - Type: double
FNX – Evaluation of single-variable function

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

\textit{Note:} All trigonometric functions process data in radians.

The error output is activated (\( E = \text{on} \)) 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)  \( \circ 1 \)  \( \text{long} \)
\( \text{yerr} \)  Substitute value for an error case  \( \text{double} \)
FNXY – Evaluation of two-variables function

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 \neq 0.0 )</td>
</tr>
<tr>
<td>3: pow</td>
<td>exponentiation of the inputs ( y = u1^u2 )</td>
<td>–</td>
</tr>
</tbody>
</table>

The \( \text{atan2} \) function result belongs to the interval \((-\pi, \pi)\). The signs of both inputs \( u1 \) and \( u2 \) are used to determine the appropriate quadrant.

The \( \text{fmod} \) 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 (\( y = yerr \)).

Inputs

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

Outputs

- \( y \): Result of the selected function
- \( E \): Error flag
  - \( \text{off} \ldots \) No error
  - \( \text{on} \ldots \) An error occurred
Parameters

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

Block Symbol

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

$$y = ku.$$ 

Input

$u$ Analog input of the block double

Output

$y$ Analog output of the block double

Parameter

$k$ Gain $\odot 1.0$ double
GRADS – Gradient search optimization

Function Description

The GRADS block performs one-dimensional optimization of the $f(x, v)$ function by gradient method, where $x \in \langle x_{\min}, x_{\max} \rangle$ 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 $k + n \ast T_S$, where $T_S$ is the execution period of the GRADS block. This means that the individual optimization iterations have a period of $n \ast T_S$. The length of step of the gradient method is given by

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

$$(dx)_i = -\text{gamma} \ast grad,$$

where $i$ stands for $i$-th iteration. The step size is restricted to lie within the interval $\langle d_{\min}, d_{\max} \rangle$. 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
iE Error code  long
  1 . . .  x \notin \langle \text{xmin}, \text{xmax} \rangle
  2 . . .  x = \text{xmin} \text{ or } x = \text{ xmax}

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmin</td>
<td>Lower limit for the x variable</td>
<td>double</td>
</tr>
<tr>
<td>xmax</td>
<td>Upper limit for the x variable</td>
<td>10.0</td>
</tr>
<tr>
<td>gamma</td>
<td>Coefficient for determining the step size of the gradient optimization method</td>
<td>0.3</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

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  
\( \downarrow 9.22\times10^{18} \uparrow 9.22\times10^{18} \) long

\( i_2 \)  
Second integer input of the block  
\( \downarrow 9.22\times10^{18} \uparrow 9.22\times10^{18} \) long

Outputs

\( n \)  
Integer sum of the input signals  
long

\( E \)  
Error flag  
bool

\( \text{off} \) ... No error

\( \text{on} \) ... An error occurred

Parameters

\( \text{vtype} \)  
Numeric type  
\( 
\begin{array}{l}
2 \ldots \text{Byte (range } 0 ... 255) \\
3 \ldots \text{Short (range } -32768 ... 32767) \\
4 \ldots \text{Long (range } -2147483648 ... 2147483647) \\
5 \ldots \text{Word (range } 0 ... 65536) \\
6 \ldots \text{DWord (range } 0 ... 4294967295) \\
10 \ldots \text{Large (range } -9223372036854775808...9223372036854775807) \\
\end{array} \\
\( \odot 4 \) long

\( \text{SAT} \)  
Saturation (overflow) checking  
bool

\( \text{off} \) ... Overflow is not checked

\( \text{on} \) ... Overflow is checked
**ISUB – Integer subtraction**

**Block Symbol**

![ISUB block](image)

**Licence:** STANDARD

**Function Description**

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

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

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

**Inputs**

- **i1**: First integer input of the block
  - Value: -9,22E+18 to 9,22E+18

- **i2**: Second integer input of the block
  - Value: -9,22E+18 to 9,22E+18

**Parameters**

- **vtype**: 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)

- **SAT**: Saturation (overflow) checking
  - off: Overflow is not checked
  - on: Overflow is checked

**Outputs**

- **n**: Integer difference between the input signals
  - Value: long

- **E**: Error flag
  - off: No error
  - on: An error occurred
**IMUL – Integer multiplication**

**Function Description**

The **IMUL** block multiplies two integer input signals \( n = i1 \times i2 \). The range of integer numbers in a computer is always restricted by the variable type. This block uses the **vtype** parameter to specify the type. If the multiple fits in the range of the given type, the result is the ordinary multiple. In the other cases the result depends on the **SAT** parameter.

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

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

**Inputs**

- **i1** First integer input of the block
  - \(-9,22E+18\) to \(+9,22E+18\)
- **i2** Second integer input of the block
  - \(-9,22E+18\) to \(+9,22E+18\)

**Parameters**

- **vtype** 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)
- **SAT** Saturation (overflow) checking
  - **off** Overflow is not checked
  - **on** Overflow is checked

**Outputs**

- **n** Integer product of the input signals
  - long
E  Error flag
    off ... No error
    on ... An error occurred
IDIV – Integer division

Block Symbol Licence: STANDARD

Function Description

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

Inputs

- \( i_1 \): First integer input of the block
  - \(-9,22E+18 \) to \(+9,22E+18\) long
- \( i_2 \): Second integer input of the block
  - \(-9,22E+18 \) to \(+9,22E+18\) 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
- \( nerr \): Substitute value for an error case
  - 1 long
IMOD – Remainder after integer division

Block Symbol

Function Description
The IMOD block divides two integer input signals, \( n = i_1 \% i_2 \), 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 \( i_2 = 0 \), the output \( E \) is set to on and the output \( n \) is substituted by \( n = nerr \).

Inputs
\[
\begin{array}{lll}
  i_1 & \text{First integer input of the block} & \downarrow -9.22\times 10^{18} \uparrow 9.22\times 10^{18} \text{ long} \\
  i_2 & \text{Second integer input of the block} & \downarrow -9.22\times 10^{18} \uparrow 9.22\times 10^{18} \text{ long} \\
\end{array}
\]

Outputs
\[
\begin{array}{ll}
  n & \text{Remainder after integer division} \text{ long} \\
  E & \text{Error flag – division by zero} \text{ bool} \\
\end{array}
\]

Parameters
\[
\begin{array}{ll}
  vtype & \text{Numeric type} \text{ long} \\
  & \begin{array}{ll}
    2 & \text{Byte} \\
    3 & \text{Short} \\
    4 & \text{Long} \\
    5 & \text{Word} \\
    6 & \text{DWord} \\
    10 & \text{Large} \\
  \end{array} \\
  nerr & \text{Substitute value for an error case} \text{ long} \\
\end{array}
\]
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  \text{double}

Output

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

Parameters

\(u_1\)  x-coordinate of the 1st interpolation node  \text{double}

\(y_1\)  y-coordinate of the 1st interpolation node  \text{double}

\(u_2\)  x-coordinate of the 2nd interpolation node  \text{double}

\(y_2\)  y-coordinate of the 2nd interpolation node  \text{double}
MUL – Multiplication of two signals

Block Symbol

Licence: STANDARD

Function Description

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

Inputs

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

Output

- \( y \): Product of the input signals, double
**POL – Polynomial evaluation**

**Block Symbol**

Function Description

The **POL** block evaluates the polynomial of the form:

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

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

**Input**

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

**Output**

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

**Parameters**

- **ai**  
  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 \( y = y_{err} \).

Input
\( u \)  Analog input of the block  double

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

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

Block Symbol

License: STANDARD

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

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

Output

$Y$ Logical output indicating whether the relation holds

Parameter

irel Relation type

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

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

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

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

Input

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

Output

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

**Block Symbol**

![SQR block diagram]

**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  
  **double**

**Output**

- $y$  
  Square of the input signal  
  **double**
**SQRT_ – Square root**

**Block Symbol**

![Block Symbol]

**Licence:** STANDARD

**Function Description**

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

$$y = \sqrt{u}.$$ 

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

**Input**

$u$ \hspace{1cm} Analog input of the block \hspace{1cm} double

**Outputs**

$y$ \hspace{1cm} Square root of the input signal \hspace{1cm} double

$E$ \hspace{1cm} Error flag \hspace{1cm} bool

- off ... No error
- on ... Square root of negative number

**Parameter**

$y_{err}$ \hspace{1cm} Substitute value for an error case \hspace{1cm} $\odot 1.0$ double
**SUB** – *Subtraction of two signals*

**Block Symbol**

License: **STANDARD**

**Function Description**

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

\[ y = u_1 - u_2. \]

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

**Inputs**

- **u1**: Analog input of the block
  - Type: `double`
- **u2**: Analog input of the block
  - Type: `double`

**Output**

- **y**: Difference between the two input signals
  - Type: `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, SELC0, SELHEX0</td>
<td>Analog signal selectors</td>
<td>133</td>
</tr>
<tr>
<td>SHIFTOCT</td>
<td>Data shift register</td>
<td>135</td>
</tr>
<tr>
<td>SHLD</td>
<td>Sample and hold</td>
<td>137</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>SINT</td>
<td>Simple integrator</td>
<td>138</td>
</tr>
<tr>
<td>SPIKE</td>
<td>Spike filter</td>
<td>139</td>
</tr>
<tr>
<td>SSW</td>
<td>Simple switch</td>
<td>141</td>
</tr>
<tr>
<td>SWR</td>
<td>Selector with ramp</td>
<td>142</td>
</tr>
<tr>
<td>VDEL</td>
<td>Variable time delay</td>
<td>143</td>
</tr>
<tr>
<td>ZV4IS</td>
<td>Zero vibration input shaper</td>
<td>144</td>
</tr>
</tbody>
</table>
ABSROT – Processing data from absolute position sensor

Block Symbol

© ADVANCED

Function Description

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

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

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

Inputs

<table>
<thead>
<tr>
<th>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

<table>
<thead>
<tr>
<th>lolim</th>
<th>Lower limit of the sensor reading</th>
<th>-3.141592654</th>
<th>double</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>double</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------------------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>RESR</td>
<td>Flag for resetting the revolutions counter</td>
<td>bool</td>
<td></td>
</tr>
<tr>
<td>off</td>
<td>Reset only the estimated position y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>Reset also the irev revolutions counter</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**ASW** – Switch with automatic selection of input

**Block Symbol**

```
+-------------+
| u1          |
| u2          |
| u3          |
| u4          |
| SW          |
+-------------+
```

**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**

- $u_1..u_4$: 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

- **delta**: Threshold for detecting a change
  - Value: $0.000001$
  - Type: double

- **p1..p4**: Parameters to be selected from
  - Type: double
**AVG – Moving average filter**

**Block Symbol**

![AVG Block Symbol](image)

**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

**Output**

- $y$: Filtered output signal

**Parameter**

- $n$: Number of samples to compute the average from
  - Default: $1 \leftrightarrow 10000000 \odot 10$
  - Type: long
- $n_{\text{max}}$: Limit for parameter $n$ (used for internal memory allocation)
  - Default: $10 \leftrightarrow 10000000 \odot 100$
  - Type: 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 (off→on) occurs at the SET input, the block is initialized for current values of the inputs \( am \), \( dm \), \( vm \) and \( sm \). The RDY output is set to off before the first initialization and during the initialization phase, otherwise it is set to 1. When rising edge (off→on) occurs at the START input, the block generates the trajectory at the outputs \( a \), \( v \), \( s \) and \( tt \), where the signals correspond to acceleration, velocity, position and time respectively. The BSY output is set to on while the trajectory is being generated, otherwise it is off.

Inputs

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>Starting signal (rising edge)</td>
<td>bool</td>
</tr>
<tr>
<td>SET</td>
<td>Initialize/compute the trajectory for the current inputs</td>
<td>bool</td>
</tr>
<tr>
<td>( am )</td>
<td>Maximal allowed acceleration ([\text{m/s}^2])</td>
<td>double</td>
</tr>
<tr>
<td>( dm )</td>
<td>Maximal allowed deceleration ([\text{m/s}^2])</td>
<td>double</td>
</tr>
<tr>
<td>( vm )</td>
<td>Maximum allowed velocity ([\text{m/s}])</td>
<td>double</td>
</tr>
<tr>
<td>( sm )</td>
<td>Desired final position ([\text{m}]) (initial position is 0)</td>
<td>double</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>Acceleration ([\text{m/s}^2])</td>
<td>double</td>
</tr>
<tr>
<td>( v )</td>
<td>Velocity ([\text{m/s}])</td>
<td>double</td>
</tr>
<tr>
<td>( s )</td>
<td>Position ([\text{m}])</td>
<td>double</td>
</tr>
<tr>
<td>( tt )</td>
<td>Time ([\text{s}])</td>
<td>double</td>
</tr>
<tr>
<td>RDY</td>
<td>Flag indicating that the block is ready to generate the trajectory</td>
<td>bool</td>
</tr>
<tr>
<td>BSY</td>
<td>Flag indicating that the trajectory is being generated</td>
<td>bool</td>
</tr>
</tbody>
</table>
BPF – Band-pass filter

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 are the block parameters \( f_m \) and \( x_i \) respectively. The \( f_m \) parameter defines the middle of the frequency transmission band and \( x_i \) is the relative damping coefficient.

If \( ISSF = \text{on} \), 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  
\[ \text{double} \]

Output
\[ y \]  
Filtered output signal  
\[ \text{double} \]

Parameters
\[ f_m \]  
Peak frequency, middle of the frequency transmission band [Hz]  
\[ \text{double} \]  
\[ \odot 1.0 \]
\[ x_i \]  
Relative damping coefficient (recommended value 0.5 to 1)  
\[ \text{double} \]  
\[ \odot 0.707 \]
\[ ISSF \]  
Steady state at start-up flag  
\[ \text{bool} \]
\[ \text{off} \ldots \text{Zero initial state} \]
\[ \text{on} \ldots \text{Initial steady state} \]
CMP – Comparator with hysteresis

Block Symbol

Function Description

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

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

where

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

and

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

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

Inputs

- \( u_1 \) First analog input of the block \( \text{double} \)
- \( u_2 \) Second analog input of the block \( \text{double} \)

Output

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

Parameter

- \( \text{hys} \) Hysteresis \( \odot 0.5 \text{ double} \)
CNDR – Nonlinear conditioner

Block Symbol

Function Description

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

![Diagram of CNDR block](image)

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

**Input**

\( u \)
- Analog input of the block
- **double**

**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
- \( \textcircled{6} \) **long**

\( SATF \)
- Saturation flag
- \( \textcircled{on} \) **bool**

off . . . Signal not limited  on . . . Saturation limits active
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Values</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>up</strong></td>
<td>Vector of increasing $u$-coordinates</td>
<td>$\mathbb{R} = [0.0 \ 3.9 \ 3.9 \ 9.0 \ 14.5 \ 20.0]$</td>
<td>double</td>
</tr>
<tr>
<td><strong>yp</strong></td>
<td>Vector of $y$-coordinates</td>
<td>$\mathbb{R} = [0.0 \ 0.0 \ 15.8 \ 38.4 \ 72.0 \ 115.0]$</td>
<td>double</td>
</tr>
</tbody>
</table>
**DEL – Delay with initialization**

**Block Symbol**

![Block Symbol](image)

**Licence:** STANDARD

**Function Description**

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

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

If the last $n$ samples are not yet known, the output is set to

$$y_k = y_0,$$

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

**Inputs**

- $u$: Analog input of the block
- $R1$: Block reset
- $y0$: Initial output value

**Outputs**

- $y$: Delayed input signal
- $RDY$: Ready flag indicating that the buffer is filled with the input signal samples

**Parameters**

- $n$: Delay (number of samples). The resulting time delay is $n \cdot T_S$, where $T_S$ is the block execution period. $\downarrow0 \uparrow10000000 \diamond10$
- $nmax$: Limit for parameter $n$ (used for internal memory allocation). $\downarrow10 \uparrow10000000 \diamond100$
DELM – Time delay

Block Symbol Licence: STANDARD

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

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

Output
\[ y \text{ Delayed input signal} \quad \text{double} \]

Parameter
\[ del \text{ Time delay [s]} \quad \uparrow 1.0 \quad \text{double} \]
\[ nmax \text{ Size of delay buffer del (number of samples). Used for internal memory allocation.} \quad \downarrow 10 \uparrow 10000000 \quad \odot 100 \quad \text{long} \]
DER – Derivation, filtering and prediction from the last \( n+1 \) samples

Function Description

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

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

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

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

**Inputs**

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

**Outputs**

- \( y \) Estimate of input signal derivative \( \text{double} \)
- \( z \) Predicted/filtered input signal \( \text{double} \)
- \( \text{RDY} \) Ready flag (all \( n + 1 \) samples are available) \( \text{bool} \)

**Parameters**

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

Block Symbol

Function Description

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

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

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

where $k$ stands for the current sampling instant.

Input

$u$ Analog input of the block double

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 $\downarrow 2 \uparrow 10000000 \odot 100$

$n_{\text{max}}$ Limit value of parameter $n$ (used for internal memory allocation) long $\downarrow 10 \uparrow 10000000 \odot 200$
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 \( t_i \) and two indicators of the output signal level (\( y_{\text{min}} \) a \( y_{\text{max}} \)). If \( \text{RUN} = \text{on} \) and \( \text{R1} = \text{off} \) then

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

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

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

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

Consider using the SINT block, whose simpler structure and functionality might be sufficient for elementary tasks.

Inputs

\begin{itemize}
  \item \( u \): Analog input of the block \hspace{1cm} \text{double}
  \item \( \text{RUN} \): Enable execution \hspace{1cm} \text{bool}
  \hspace{1cm} \text{off \ldots Integration stopped on \ldots Integration running}
  \item \( \text{R1} \): Block reset, initialization of the integrator output to \( y_0 \) \hspace{1cm} \text{bool}
  \item \( y_0 \): Initial output value \hspace{1cm} \text{double}
  \item \( t_i \): Integral time constant \hspace{1cm} \text{double}
\end{itemize}

Outputs

\begin{itemize}
  \item \( y \): Integrator output \hspace{1cm} \text{double}
  \item \( Q \): Running integration indicator \hspace{1cm} \text{bool}
  \item \( \text{LY} \): Lower level indicator (\( y < y_{\text{min}} \)) \hspace{1cm} \text{bool}
  \item \( \text{HY} \): Upper level indicator (\( y > y_{\text{max}} \)) \hspace{1cm} \text{bool}
\end{itemize}
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ymin</td>
<td>Lower level definition</td>
<td>0.0</td>
<td>double</td>
</tr>
<tr>
<td>ymax</td>
<td>Upper level definition</td>
<td>1.0</td>
<td>double</td>
</tr>
</tbody>
</table>
KDER – Derivation and filtering of the input signal

**Block Symbol**

Licence: **ADVANCED**

**Function Description**

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

The block is configured by only two parameters pbeta and norder. The pbeta parameter depends on the sampling period \( T_S \), frequency properties of the input signal \( u \) and also the noise to signal ratio. An approximate formula \( pbeta \sim T_S \omega_0 \) 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 \hspace{1cm} \text{double}

**Outputs**

\( y \) Filtered input signal \hspace{1cm} \text{double}

\( dy \) Estimated 1st order derivative \hspace{1cm} \text{double}

\( d2y \) Estimated 2nd order derivative \hspace{1cm} \text{double}

\( d3y \) Estimated 3rd order derivative \hspace{1cm} \text{double}

\( d4y \) Estimated 4th order derivative \hspace{1cm} \text{double}

\( d5y \) Estimated 5th order derivative \hspace{1cm} \text{double}

**Parameters**

\( \text{norder} \) Order of the derivative filter \hspace{1cm} \downarrow 2 \uparrow 10 \odot 3 \hspace{1cm} \text{long}
\texttt{pbeta} \hspace{1em} \text{Bandwidth of the derivative filter} \hspace{1em} \downarrow 0.0 \circ 0.1 \hspace{1em} \texttt{double}
LPF – Low-pass filter

Block Symbol  Licence: STANDARD

Function Description

The LPF block implements a second order filter in the form
\[ F_s = \frac{1}{a^2 s^2 + 2\xi as + 1}, \]
where
\[ a = \frac{\sqrt{2\sqrt{2\xi^4 - 2\xi^2 + 1} - 2\xi^2 + 1}}{2\pi f_b} \]
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 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
\[ u \quad \text{Input signal to be filtered} \quad \text{double} \]

Output
\[ y \quad \text{Filtered output signal} \quad \text{double} \]

Parameters
\[ f_b \quad \text{Filter bandwidth [Hz]; the frequencies in the range } (0, f_b) \text{ pass through the filter, the attenuation at the frequency } f_b \text{ is 3 dB and approximately } 40 \text{ dB at } 10 \cdot f_b; \text{ it must hold } f_b \leq \frac{1}{10 T_S} \text{ for proper function of the filter, where } T_S \text{ is the block execution period} \quad \text{double} \]
\[ x_i \quad \text{Relative damping coefficient (recommended value } 0.5 \text{ to } 1) \quad \text{double} \]
\[ \text{ISSF} \quad \text{Steady state at start-up} \quad \text{bool} \]
\[ \text{off . . . Zero initial state} \]
\[ \text{on . . . Initial steady state} \]
**MINMAX – Running minimum and maximum**

**Block Symbol**

![MINMAX Block Symbol]

**Licence:** STANDARD

**Function Description**

The MINMAX function block evaluates minimum and maximum from the last \( n \) samples of the \( u \) input signal. The output \( \text{RDY} = \text{off} \) 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, \( \text{double} \)
- **R1**: Block reset, \( \text{bool} \)

**Outputs**

- **ymin**: Minimal value found, \( \text{double} \)
- **ymax**: Maximal value found, \( \text{double} \)
- **RDY**: Ready flag (buffer filled), \( \text{bool} \)

**Parameters**

- **n**: Number of samples for analysis (buffer length), \( \downarrow 1 \uparrow 10000000 \circ 100 \) \( \text{long} \)
- **nmax**: Limit value of parameter \( n \) (used for internal memory allocation), \( \downarrow 10 \uparrow 10000000 \circ 200 \) \( \text{long} \)
**NSCL – Nonlinear scaling factor**

Block Symbol

![Block Symbol]

Licence: **STANDARD**

**Function Description**

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

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

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

![Graph]

**Input**

\(u\)  Analog input of the block  double

**Output**

\(y\)  Analog output of the block  double

**Parameters**

- \(\text{gain}\)  Signal gain  \(\odot 1.0\)  double
- \(\text{ze}\)  Shaping parameter  \(\odot 1.0\)  double
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 freq and optional higher harmonic frequencies. The computations are performed over the last m samples of the input signal u, where m = nper/freq/Ts, i.e. from the time-window of the length equivalent to nper periods of the fundamental frequency.

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

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

Inputs

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

Outputs

- amp: Amplitude of the fundamental frequency
- thd: Total harmonic distortion (only for nharm ≥ 1)
- vAmp: Vector of amplitudes at given frequencies
- vPhi: Vector of phase-shifts at given frequencies
- vRe: Vector of real parts at given frequencies
- vIm: Vector of imaginary parts at given frequencies
- E: Error flag
- iE: Error code
  - i . . . . REXYGEN general error
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>freq</code></td>
<td>Fundamental frequency</td>
<td>0.0000000001 to 1000000000.0</td>
<td>double</td>
</tr>
<tr>
<td><code>nper</code></td>
<td>Number of periods to calculate upon</td>
<td>1 to 10000</td>
<td>long</td>
</tr>
<tr>
<td><code>nharm</code></td>
<td>Number of monitored harmonic frequencies</td>
<td>0 to 16</td>
<td>long</td>
</tr>
<tr>
<td><code>ifrunit</code></td>
<td>Frequency units</td>
<td>1 Hz, 1 rad/s</td>
<td>long</td>
</tr>
<tr>
<td><code>iphunit</code></td>
<td>Phase shift units</td>
<td>1 degree, 1 rad</td>
<td>long</td>
</tr>
<tr>
<td><code>nmax</code></td>
<td>Allocated size of array</td>
<td>10 to 1000000000</td>
<td>long</td>
</tr>
<tr>
<td><code>freq2</code></td>
<td>Vector of user-defined monitored frequencies</td>
<td>[2.0, 3.0, 4.0]</td>
<td>double</td>
</tr>
</tbody>
</table>
**RLIM – Rate limiter**

**Block Symbol**

![RLIM Block Symbol](image)

**Licence:** STANDARD

**Function Description**

The **RLIM** block copies the input signal \( u \) to the output \( y \), but the maximum allowed rate of change is limited. The limits are given by the time constants \( t_p \) and \( t_n \):

- the steepest rise per second: \( 1/t_p \)
- the steepest descent per second: \( -1/t_n \)

**Input**

| \( u \) | Input signal to be filtered | double |

**Output**

| \( y \) | Filtered output signal | double |

**Parameters**

| \( t_p \) | Time constant defining the maximum allowed rise | \( \leq 2.0 \) | double |
| \( t_n \) | Time constant defining the maximum allowed descent (note that \( t_n > 0 \)) | \( \leq 2.0 \) | double |
**S1OF2 – One of two analog signals selector**

**Block Symbol**

```
\[ S1OF2 \]
```

**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 \( nd \) samples exceed the given limit, which is given by the following inequation

\[
|u_1 - u_2| > \frac{pdev \cdot (vmax - vmin)}{100},
\]

where \( vmin \) and \( vmax \) 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 F1–F4 (see the SAI block) and the \( D \) flag. For the input \( R \) set permanently to on, the invalidity indicator \( E_1 \) (\( E_2 \)) is set to on for only one cycle period whenever some invalidity condition is fulfilled. On the other hand, for \( R = 0 \), the output \( E_1 \) (\( E_2 \)) is set to on and remains true until the reset (\( R: \text{off} \rightarrow \text{on} \)).
similar rule holds for the $E$ output. For the input $R$ set permanently to on, the $E$ output is set to on for only one cycle period whenever a rising edge occurs in the internal $D$ flag ($D = \text{off} \rightarrow \text{on}$). On the other hand, for $R = 0$, the output $E$ is set to on and remains true until the reset (rising edge $R: \text{off} \rightarrow \text{on}$). 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>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>
<tr>
<td>sv</td>
<td>Substitute value for an error case, i.e. $E = \text{on}$</td>
<td>double</td>
</tr>
<tr>
<td>HF1</td>
<td>Hardware error flag for signal $u_1$</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... The input module of the signal works normally</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... Hardware error of the input module occurred</td>
<td></td>
</tr>
<tr>
<td>HF2</td>
<td>Hardware error flag for signal $u_2$</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... The input module of the signal works normally</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... Hardware error of the input module occurred</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>Reset inner error flags of the input signals $u_1$ and $u_2$</td>
<td>bool</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>y</th>
<th>Analog output of the block</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Output signal invalidity indicator</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... Signal is valid</td>
<td>on .... Signal is invalid</td>
</tr>
<tr>
<td>E1</td>
<td>Invalidity indicator for input $u_1$</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... Signal is valid</td>
<td>on .... Signal is invalid, $y = u_2$</td>
</tr>
<tr>
<td>E2</td>
<td>Invalidity indicator for input $u_2$</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... Signal is valid</td>
<td>on .... Signal is invalid, $y = u_1$</td>
</tr>
<tr>
<td>iE1</td>
<td>Reason of input $u_1$ invalidity</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>0 .... Signal valid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 .... Signal out of range</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 .... Signal varies too little</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 .... Signal varies too little and signal out of range</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 .... Signal varies too much</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 .... Signal varies too much and signal out of range</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6 .... Signal varies too much and too little</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7 .... Signal varies too much and too little and signal out of range</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 .... Hardware error</td>
<td></td>
</tr>
<tr>
<td>iE2</td>
<td>Reason of input $u_2$ invalidity, see the iE1 output</td>
<td>long</td>
</tr>
<tr>
<td>W</td>
<td>Warning flag (invalid input signal)</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... Both input signals $u_1$ and $u_2$ are valid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... At least one of the input signals is invalid</td>
<td></td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</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 signals $u_1$ and $u_2$ after initialization of the block</td>
<td>long</td>
</tr>
<tr>
<td>nc</td>
<td>Number of samples for invariability testing (see the SAI block, condition F2)</td>
<td>long</td>
</tr>
<tr>
<td>nbits</td>
<td>Number of A/D converter bits (source of the signals $u_1$ and $u_2$)</td>
<td>long</td>
</tr>
<tr>
<td>nr</td>
<td>Number of samples for variability testing (see the SAI block, condition F3)</td>
<td>long</td>
</tr>
<tr>
<td>prate</td>
<td>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)</td>
<td>double</td>
</tr>
<tr>
<td>nv</td>
<td>Number of samples for out-of-range testing (see the SAI block, condition F4)</td>
<td>long</td>
</tr>
<tr>
<td>vmin</td>
<td>Lower limit for the input signals $u_1$ and $u_2$</td>
<td>double</td>
</tr>
<tr>
<td>vmax</td>
<td>Upper limit for the input signals $u_1$ and $u_2$</td>
<td>double</td>
</tr>
<tr>
<td>nd</td>
<td>Number of samples for deviation testing (inner flag D; D is always off for $nd = 0$)</td>
<td>long</td>
</tr>
<tr>
<td>pdev</td>
<td>Maximum allowed percentage deviation of the inputs $u_1$ and $u_2$ with respect to the overall range of the input signals $v_{max} - v_{min}$</td>
<td>double</td>
</tr>
<tr>
<td>mode</td>
<td>Defines how to compute the output signal $y$ when both input signals are valid ($E_1 = off$, $E_2 = off$ and $D = off$)</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>1 . . . . Average, $y = \frac{u_1 + u_2}{2}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 . . . . Minimum, $y = \min(u_1, u_2)$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 . . . . Maximum, $y = \max(u_1, u_2)$</td>
<td></td>
</tr>
</tbody>
</table>
**SAI – Safety analog input**

**Block Symbol**

![Block Symbol Image]

**Function Description**

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

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

**F2:** The input signal \( u \) varies too little. The last \( \text{nc} \) samples of the input \( u \) lies within the interval of width \( \text{du} \),

\[
\text{du} = \begin{cases} 
\frac{\text{vmax} - \text{vmin}}{2^{\text{nbits}}} , & \text{for } \text{nbits} \in \{8, 9, ..., 16\} \\
0 , & \text{for } \text{nbits} \notin \{8, 9, ..., 16\},
\end{cases}
\]

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

![Diagram showing F2 condition]

If the parameter \( \text{nc} \) is set to \( \text{nc} = 0 \), the condition \( F2 = 0 \) is never fulfilled.

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

\[
\text{rate} = \text{prate} \frac{\text{vmax} - \text{vmin}}{100},
\]

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

![Diagram showing acceptable and unacceptable changes in the signal](image)

If the parameter \( n_r \) is set to \( n_r = 0 \), the condition \( F_3 \) is never fulfilled.

\( F_4 \): The input signal \( u \) is out of range. The last \( n_v \) samples of the input signal \( u \) lie out of the allowed range \( (v_{\text{min}}, v_{\text{max}}) \).

If the parameter \( n_v \) is set to \( n_v = 0 \), the condition \( F_4 \) is never fulfilled.

The signal \( u \) is copied to the output \( y \) without any modification when it is considered valid. In the other case, the output \( y \) is determined by a substitute value from the \( sv \) input. In such a case the output \( E \) is set to \( \text{on} \) and the output \( iE \) provides the error code. The input \( R \) resets the inner error flags \( F_1 \)–\( F_4 \). For the input \( R \) set permanently to \( \text{on} \), the invalidity indicator \( E \) is set to \( \text{on} \) for only one cycle period whenever some invalidity condition is fulfilled. On the other hand, for \( R = \text{off} \), the output \( E \) is set to \( \text{on} \) and remains true until the reset (rising edge \( R: \text{off} \to \text{on} \)).

The table of error codes \( iE \) resulting from the inner error flags \( F_1 \)–\( F_4 \):

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

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

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

\( sv \) Substitute value to be used when the signal \( u \) is marked as invalid \( \text{double} \)

\( \text{HWF} \) Hardware error indicator \( \text{bool} \)

- \( \text{off} \ldots \) The input module of the signal works normally
- \( \text{on} \ldots \) Hardware error of the input module occurred

\( R \) Reset inner error flags F1–F4 \( \text{bool} \)

Outputs

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

\( yf \) Filtered analog output signal \( y \), output of the SPIKE filter \( \text{double} \)

\( E \) Output signal invalidity indicator \( \text{bool} \)

- \( \text{off} \ldots \) Signal is valid \( sv \)
- \( \text{on} \ldots \) Signal is invalid, \( y = yf = \)

\( iE \) Reason of invalidity \( \text{long} \)

- 0 \ldots \) Signal valid
- 1 \ldots \) Signal out of range
- 2 \ldots \) Signal varies too little
- 3 \ldots \) Signal varies too little and signal out of range
- 4 \ldots \) Signal varies too much
- 5 \ldots \) Signal varies too much and signal out of range
- 6 \ldots \) Signal varies too much and too little
- 7 \ldots \) Signal varies too much and too little and signal out of range
- 8 \ldots \) Hardware error

Parameters

\( \text{nb} \) Number of samples which are not included in the validity assessment of the signal \( u \) after initialization of the block \( \text{long} \)

\( \text{nc} \) Number of samples for invariability testing (the F2 condition) \( \text{long} \)

\( \text{nbits} \) Number of A/D converter bits \( \text{long} \)

\( \text{nr} \) Number of samples for variability testing (the F3 condition) \( \text{long} \)

\( \text{prate} \) Maximum allowed percentage change of the input \( u \) within the last \( \text{nr} \) samples (with respect to the overall range of the input signal \( \text{vmax} - \text{vmin} \)) \( \text{double} \)

\( \text{nv} \) Number of samples for out-of-range testing (the F4 condition) \( \text{long} \)

\( \text{vmin} \) Lower limit for the input signal \( u \) \( \text{double} \)

\( \text{vmax} \) Upper limit for the input signal \( u \) \( \text{double} \)
SEL – Analog signal selector

Function Description

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

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

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

Inputs

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

- **iSW**: Active signal selector, active when BINF = off  
  
  **SW1**: Binary signal selector, active when BINF = on  
  
  **SW2**: Binary signal selector, active when BINF = on

Output

- **y**: The selected signal

Parameter

- **BINF**: Enable the binary selectors
  
  - **off**: Disabled (analog selector)  
  - **on**: Enabled (binary selectors)
SELQUAD, SELOCT, SELHEXD – Analog signal selectors

Block Symbols

Licence: STANDARD

Function Description

The SELQUAD, SELOCT and SELHEX blocks select one of the input signals and copy it to the output signal \( y \). The selection of the active signal \( u_0 \ldots u_{15} \) 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.
CHAPTER 5. ANALOG – ANALOG SIGNAL PROCESSING

Inputs

\( u_{0..15} \)  Analog inputs of the block  \( \text{double} \)

\( \text{iSW} \)  Active signal selector  \( \text{long} \)

\( \text{SW0..3} \)  Binary signal selectors  \( \text{bool} \)

Output

\( y \)  The selected input signal  \( \text{double} \)

Parameter

\( \text{BINF} \)  Enable the binary selectors  \( \text{bool} \)

\text{off} . . .  Disabled (analog selector)

\text{on} . . . .  Enabled (binary selectors)
SHIFTOCT – Data shift register

Function Description

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

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

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

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

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

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

Inputs

- \(u\): Data input of the register
- \(\text{RUN}\): Enables outputs shift

Outputs

- \(y_0\): First output of the block
- \(y_1\): Second output of the block
- \(y_2\): Third output of the block
- \(y_3\): Fourth output of the block
- \(y_4\): Fifth output of the block
- \(y_5\): Sixth output of the block
- \(y_6\): Seventh output of the block
- \(y_7\): Eighth output of the block
### Parameters

<table>
<thead>
<tr>
<th>VType</th>
<th>Output Data Type</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>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Large</td>
</tr>
</tbody>
</table>
SHLD – Sample and hold

Block Symbol

Function Description

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

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

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

The binary input $R1$ sets the output $y$ to the value $y0$, it overpowers the SETH input signal.

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

Inputs

| $u$        | Analog input of the block | double |
| $SETH$    | Trigger for the set and hold operation | bool |
| $R1$      | Block reset, $R1 = \text{on} \rightarrow y = y0$ | 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

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  
\text{double}

Output

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

Parameters

\( ti \)  
Integral time constant \( T_i \)  
\text{1.0}  
\text{double}

\( y0 \)  
Initial output value  
\text{double}

\( ymax \)  
Upper limit of the output signal  
\text{1.0}  
\text{double}

\( ymin \)  
Lower limit of the output signal  
\text{-1.0}  
\text{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 \((u, y) \rightarrow y\):

\[
\text{delta} := y - u;
\]
\[
\text{if abs(delta) < gap then begin}
\]
\[
y := u;
\]
\[
gap := \text{gap}/q;
\]
\[
\text{if gap < mingap then gap := mingap;}
\]
\[
\text{end else begin}
\]
\[
\text{if delta < 0 then y := y + gap}
\]
\[
\text{else y := y - gap;}
\]
\[
gap := \text{gap} \times q;
\]
\[
\text{end}
\]

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

\texttt{mingap} \quad \text{Minimum size of the tolerance window} \quad \odot 0.01 \quad \text{double}

\texttt{q} \quad \text{Tolerance window adaptation speed} \quad \downarrow 1.0 \odot 2.0 \quad \text{double}
**SSW – Simple switch**

**Block Symbol**

![Block Symbol Image]

**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 \( SW = off \) (\( SW = on \)), then the selected signal is \( u_1 \) (\( u_2 \)).

**Inputs**

- \( u_1 \) First analog input of the block \hspace{1cm} double
- \( u_2 \) Second analog input of the block \hspace{1cm} double
- \( SW \) Signal selector \hspace{1cm} 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 \hspace{1cm} double
SWR – Selector with ramp

Block Symbol

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 \hspace{1cm} \text{double}
- \( u_2 \) Second analog input of the block \hspace{1cm} \text{double}
- \( SW \) Signal selector \hspace{1cm} \text{bool}
  - \text{off} \ldots The \( u_1 \) signal is selected
  - \text{on} \ldots The \( u_2 \) signal is selected

Parameters

- \( t_1 \) Rate limiter time constant for switching from \( u_2 \) to \( u_1 \) \hspace{1cm} \text{double} \hspace{1cm} 1.0
- \( t_2 \) Rate limiter time constant for switching from \( u_1 \) to \( u_2 \) \hspace{1cm} \text{double} \hspace{1cm} 1.0
- \( y_0 \) Initial output value to start the tracking from (before the first switching of signals occurs) \hspace{1cm} \text{double}

Output

- \( y \) Analog output of the block \hspace{1cm} \text{double}
**VDEL – Variable time delay**

**Block Symbol**

![Block Symbol]

**Licence:** STANDARD

**Function Description**

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

**Inputs**

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

**Output**

- $y$ : Delayed input signal  
  - Type: double

**Parameter**

- $y_0$ : Initial/substitute output value  
  - Type: double
- $\text{nmax}$ : Size of delay buffer (number of samples) for the time delay $d$. Used for internal memory allocation.  
  - Type: long  
  - Range: $\downarrow 10 \uparrow 10000000 \odot 1000$
ZV4IS – Zero vibration input shaper

Block Symbol

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.
CHAPTER 5. ANALOG – ANALOG SIGNAL PROCESSING

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

**Input**

$u$ Input signal to be filtered

$double$
Outputs

\[ y \] Filtered output signal \hspace{1cm} \text{double}
\[ E \] Error flag \hspace{1cm} \text{bool}

- \text{off} ... No error \hspace{1cm} \text{on} ... An error occurred

Parameters

\text{omega} \quad \text{Natural frequency} \quad \odot 1.0 \quad \text{double}
\text{xi} \quad \text{Relative damping coefficient} \quad \text{double}
\text{ipar} \quad \text{Specification} \quad \odot 1 \quad \text{long}

- 1 ..... Basic types of IS
- 2 ..... Complete parametrization

\text{istype} \quad \text{Type} \quad \odot 2 \quad \text{long}

- 1 ..... ZV
- 2 ..... ZVD
- 3 ..... ZVDD
- 4 ..... MISZV
- 5 ..... UEI1
- 6 ..... UEI2
- 7 ..... UEI5
- 8 ..... UTHEI1
- 9 ..... UTHEI2
- 10 ..... UTHEI5

\text{p_alpha} \quad \text{Shaper duration/assymetry parameter} \quad \odot 0.2 \quad \text{double}
\text{p_a2} \quad \text{Insensitivity parameter} \quad \odot 0.5 \quad \text{double}
\text{p_a3} \quad \text{Additional parameter (only for p_alpha = 0)} \quad \odot 0.5 \quad \text{double}
\text{nmax} \quad \text{Size of data buffer (number of samples). Used for internal memory allocation.} \quad \downarrow 10 \uparrow 10000000 \odot 1000 \quad \text{long}
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 | 156 |
| PRBS – Pseudo-random binary sequence generator | 157 |
| SG, SGI – Signal generators | 159 |
**ANLS – Controlled generator of piecewise linear function**

**Block Symbol**

![ANLS Block Symbol]

**Licence:** STANDARD

**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 $i_2 = 0$ becomes the active time interval.

**Input**

- **RUN**
  - Enable execution, run the analog sequence generation
  - Type: bool

**Outputs**

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

- **is**
  - Index of the active time interval
  - Type: long

**Parameters**

- **y0**
  - Initial output value
  - Type: double

- **t1**
  - Node 1 time
  - Type: double

- **y1**
  - Node 1 value
  - Type: double

- **t2**
  - Node 2 time
  - Type: double

- **y2**
  - Node 2 value
  - Type: double
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<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>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>
CHAPTER 6. GEN – SIGNAL GENERATORS

BINS – Controlled binary sequence generator

Block Symbol

Licence: STANDARD

Function Description

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

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

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

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

Input

\texttt{START} \\
Starting signal (rising edge) \hspace{1cm} \texttt{bool}

Output

\texttt{Y} \\
Logical output of the block \hspace{1cm} \texttt{bool}

\texttt{is} \\
Index of the active time interval \hspace{1cm} \texttt{long}
**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y0</td>
<td>Initial output value</td>
<td>bool</td>
<td>off ... Disabled/false on ... Enabled/true</td>
</tr>
<tr>
<td>t1</td>
<td>Switching time 1 [s]</td>
<td>double</td>
<td>t1[0] 0.0 t1[1] 1.0</td>
</tr>
<tr>
<td>t2</td>
<td>Switching time 2 [s]</td>
<td>double</td>
<td>t2[0] 0.0 t2[1] 2.0</td>
</tr>
<tr>
<td>t3</td>
<td>Switching time 3 [s]</td>
<td>double</td>
<td>t3[0] 0.0 t3[1] 3.0</td>
</tr>
<tr>
<td>t4</td>
<td>Switching time 4 [s]</td>
<td>double</td>
<td>t4[0] 0.0 t4[1] 4.0</td>
</tr>
<tr>
<td>t5</td>
<td>Switching time 5 [s]</td>
<td>double</td>
<td>t5[0] 0.0 t5[1] 5.0</td>
</tr>
<tr>
<td>t6</td>
<td>Switching time 6 [s]</td>
<td>double</td>
<td>t6[0] 0.0 t6[1] 6.0</td>
</tr>
<tr>
<td>t7</td>
<td>Switching time 7 [s]</td>
<td>double</td>
<td>t7[0] 0.0 t7[1] 7.0</td>
</tr>
<tr>
<td>t8</td>
<td>Switching time 8 [s]</td>
<td>double</td>
<td>t8[0] 0.0 t8[1] 8.0</td>
</tr>
<tr>
<td>RPT</td>
<td>Repeating sequence</td>
<td>bool</td>
<td>off ... Disabled on ... Enabled</td>
</tr>
</tbody>
</table>
BIS – Binary sequence generator

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

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

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

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

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

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

Parameters
Y0 Initial output value bool
  off . . . Disabled/false on . . . Enabled/true

t1 Switching time 1 [s] ↓0.0 ◀1.0 double
<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td>Switching time 2 [s]</td>
<td>0.0 ( \oplus ) 2.0 double</td>
</tr>
<tr>
<td>t3</td>
<td>Switching time 3 [s]</td>
<td>0.0 ( \oplus ) 3.0 double</td>
</tr>
<tr>
<td>t4</td>
<td>Switching time 4 [s]</td>
<td>0.0 ( \oplus ) 4.0 double</td>
</tr>
<tr>
<td>t5</td>
<td>Switching time 5 [s]</td>
<td>0.0 ( \oplus ) 5.0 double</td>
</tr>
<tr>
<td>t6</td>
<td>Switching time 6 [s]</td>
<td>0.0 ( \oplus ) 6.0 double</td>
</tr>
<tr>
<td>t7</td>
<td>Switching time 7 [s]</td>
<td>0.0 ( \oplus ) 7.0 double</td>
</tr>
<tr>
<td>t8</td>
<td>Switching time 8 [s]</td>
<td>0.0 ( \oplus ) 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>
**MP – Manual pulse generator**

**Block Symbol**  
- **Licence:** STANDARD

**Function Description**

The MP block generates a pulse of width \(\text{pwidth}\) 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 \(\text{pwidth}\).

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

**Output**

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

**Parameters**

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

Block Symbol

Function Description

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

The initial and final values of the sequence are $\text{val0}$. The sequence starts from this value when rising edge occurs at the START input (off→on), the output $y$ is immediately switched to the $\text{valhi}$ value. The generator then switches the output to the other limit value with the period of $\text{swper}$ seconds and the probability of switching $\text{swprob}$. After $\text{seqt}$ seconds the output is set back to $\text{val0}$. A $\text{waitt}$-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 $\text{BRK} = \text{on}$ 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>$\text{BSY}$</td>
<td>Busy flag</td>
<td>bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>$\text{val0}$</th>
<th>Initial and final value</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{valhi}$</td>
<td>Upper level of the $y$ output</td>
<td>$\odot 1.0$ double</td>
</tr>
<tr>
<td>$\text{vallo}$</td>
<td>Lower level of the $y$ output</td>
<td>$\odot -1.0$ double</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>swper</td>
<td>Period of random output switching [s]</td>
<td>1.0 double</td>
</tr>
<tr>
<td>swprob</td>
<td>Probability of switching</td>
<td>0.0 - 1.0 double</td>
</tr>
<tr>
<td>seqt</td>
<td>Length of the sequence [s]</td>
<td>10.0 double</td>
</tr>
<tr>
<td>waitt</td>
<td>Settling period [s]</td>
<td>2.0 double</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 (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

- RUN: Enable execution, run the binary sequence generation
- SYN: Synchronization signal

Output

- y: Analog output of the block

Parameters

- isig: Generated signal type
  - 1: Sine wave
  - 2: Symmetrical rectangular signal
  - 3: Sawtooth signal
  - 4: White noise with uniform distribution

- amp: Amplitude of the generated signal
  - \( \in [0, 1.0] \) double

- freq: Frequency of the generated signal
  - \( \in [0, 1.0] \) double

- phase: Phase shift of the generated signal
  - \( \in [0, 1.0] \) double

- offset: Value added to the generated signal
  - \( \in [0, 1.0] \) double

- ifrunit: Frequency units
  - 1: Hz
  - 2: rad/s

- iphunit: Phase shift units
  - 1: degrees
  - 2: radians
## Chapter 7

**REG – Function blocks for control**

### Contents

<table>
<thead>
<tr>
<th>Function Block</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>166</td>
</tr>
<tr>
<td>I3PM</td>
<td>Identification of a three parameter model</td>
<td>168</td>
</tr>
<tr>
<td>LC</td>
<td>Lead compensator</td>
<td>170</td>
</tr>
<tr>
<td>LLC</td>
<td>Lead-lag compensator</td>
<td>171</td>
</tr>
<tr>
<td>MCU</td>
<td>Manual control unit</td>
<td>172</td>
</tr>
<tr>
<td>PIDAT</td>
<td>PID controller with relay autotuner</td>
<td>174</td>
</tr>
<tr>
<td>PIDE</td>
<td>PID controller with defined static error</td>
<td>177</td>
</tr>
<tr>
<td>PIDGS</td>
<td>PID controller with gain scheduling</td>
<td>179</td>
</tr>
<tr>
<td>PIDMA</td>
<td>PID controller with moment autotuner</td>
<td>181</td>
</tr>
<tr>
<td>PIDU</td>
<td>PID controller unit</td>
<td>187</td>
</tr>
<tr>
<td>PIDUI</td>
<td>PID controller unit with variable parameters</td>
<td>190</td>
</tr>
<tr>
<td>POUT</td>
<td>Pulse output</td>
<td>192</td>
</tr>
<tr>
<td>PRGM</td>
<td>Setpoint programmer</td>
<td>193</td>
</tr>
<tr>
<td>PSMPC</td>
<td>Pulse-step model predictive controller</td>
<td>195</td>
</tr>
<tr>
<td>PWM</td>
<td>Pulse width modulation</td>
<td>199</td>
</tr>
<tr>
<td>RLY</td>
<td>Relay with hysteresis</td>
<td>201</td>
</tr>
<tr>
<td>SAT</td>
<td>Saturation with variable limits</td>
<td>202</td>
</tr>
<tr>
<td>SC2FA</td>
<td>State controller for 2nd order system with frequency autotuner</td>
<td>204</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>217</td>
</tr>
<tr>
<td>SMHCC</td>
<td>Sliding mode heating/cooling controller</td>
<td>218</td>
</tr>
<tr>
<td>SMHCCA</td>
<td>Sliding mode heating/cooling controller with autotuner</td>
<td>222</td>
</tr>
<tr>
<td>SWU</td>
<td>Switch unit</td>
<td>229</td>
</tr>
</tbody>
</table>
ARLY – Advance relay

Block Symbol Licence: STANDARD

Function Description

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

Input

\( u \) Analog input of the block double

Output

\( y \) Analog output of the block double

Parameters

\( ep \) Value for switching the output to the "On" state ⊞-1.0 double

\( en \) Value for switching the output to the "Off" state ⊞1.0 double

\( tol \) Tolerance limit for the superposed noise of the input signal \( u \) ⊞0.0 ⊞0.5 double

\( ap \) Value of the \( y \) output in the "On" state ⊞1.0 double

\( an \) Value of the \( y \) output in the "Off" state ⊞-1.0 double

\( y0 \) 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

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

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.

\[
\begin{align*}
&x_1 \quad x_2 \quad x_3 \quad x_4 \\
1 & & & \downarrow & \uparrow & \;
\end{align*}
\]

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.
**Inputs**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>First analog input of the block</td>
<td>double</td>
</tr>
<tr>
<td>v</td>
<td>Second analog input of the block</td>
<td>double</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>ir</td>
<td>Dominant rule</td>
<td>long</td>
</tr>
<tr>
<td>wr</td>
<td>Degree of truth of the dominant rule</td>
<td>double</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>umax</td>
<td>Upper limit of the u input</td>
<td>double</td>
</tr>
<tr>
<td>umin</td>
<td>Lower limit of the u input</td>
<td>double</td>
</tr>
<tr>
<td>nu</td>
<td>Number of membership functions of the input u</td>
<td>long</td>
</tr>
<tr>
<td>vmax</td>
<td>Upper limit of the v input</td>
<td>double</td>
</tr>
<tr>
<td>vmin</td>
<td>Lower limit of the v input</td>
<td>double</td>
</tr>
<tr>
<td>nv</td>
<td>Number of membership functions of the input v</td>
<td>long</td>
</tr>
<tr>
<td>ny</td>
<td>Number of membership functions of the output y</td>
<td>long</td>
</tr>
<tr>
<td>nr</td>
<td>Number of inference rules</td>
<td>long</td>
</tr>
<tr>
<td>mfu</td>
<td>Matrix of membership functions of the input u</td>
<td>double</td>
</tr>
<tr>
<td>mfv</td>
<td>Matrix of membership functions of the input v</td>
<td>double</td>
</tr>
<tr>
<td>sty</td>
<td>Matrix of membership functions of the output y</td>
<td>double</td>
</tr>
<tr>
<td>rls</td>
<td>Matrix of inference rules</td>
<td>byte</td>
</tr>
</tbody>
</table>

\[
\begin{array}{ccc}
\text{mfu} & \mathbf{\{-1 -1 -1 0; -1 0 0 1; 0 1 1 1\}} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{mfv} & \mathbf{\{-1 -1 -1 0; -1 0 0 1; 0 1 1 1\}} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{sty} & \mathbf{\{-1 0 0; 0 0 0; 1 0 0\}} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{rls} & \mathbf{\{1 2 3 100; 1 1 1 100; 1 0 3 100\}} \\
\end{array}
\]
FRID – Frequency response identification

Function Description

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

Inputs

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

Parameters

ubias Static component of the exciting signal double
uamp Amplitude of the exciting signal $\circ 1.0$ double
wb Frequency interval lower limit [rad/s] $\circ 1.0$ double
wf Frequency interval higher limit [rad/s] $\circ 10.0$ double
isweep Frequency sweeping mode
1 Logarithmic
2 Linear

cp Sweeping Rate $\circ 0.995$ double
iavg Number of values for averaging $\circ 10$ long
obw Observer bandwidth
1 LOW
2 NORMAL
3 HIGH
stime Settling period [s] $\circ 10.0$ double
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<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>0.1</td>
</tr>
<tr>
<td>adapt_rc</td>
<td>Maximum rate of amplitude variation</td>
<td>0.001</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>on</td>
</tr>
<tr>
<td>immode</td>
<td>Measurement mode</td>
<td>1</td>
</tr>
<tr>
<td>nwm</td>
<td>Number of frequency response point for automatic mode</td>
<td>long</td>
</tr>
<tr>
<td>wm</td>
<td>Frequency measurement points for manual meas. mode</td>
<td>[2.0 4.0 6.0 8.0]</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</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>long</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

Block Symbol

Function Description

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

Inputs

- **u**: Input of the identified system, type `double`
- **y**: Output of the identified system, type `double`
- **u0**: Input steady state, type `double`
- **y0**: Output steady state, type `double`
- **RUN**: Execute identification, type `bool`
- **CLR**: Block reset, type `bool`
- **ips**: Meaning of the output signals, type `long`
  
  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
  - **p1**: parameter \( \kappa \)
  - **p2**: parameter \( \mu \)
  - **p3**: parameter \( \sigma^2 \)
  - **p4**: parameter \( \sigma \)
Outputs

<table>
<thead>
<tr>
<th>p(i)</th>
<th>Identified parameters with respect to (\text{ips}, i = 1, \ldots, 8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSY</td>
<td>Busy flag</td>
</tr>
<tr>
<td>RDY</td>
<td>Ready flag</td>
</tr>
<tr>
<td>E</td>
<td>Error flag</td>
</tr>
<tr>
<td>(\text{iE})</td>
<td>Error code</td>
</tr>
</tbody>
</table>

\begin{itemize}
\item 1 \ldots Premature termination (\text{RUN = off})
\item 2 \ldots \(\mu_0 = 0\)
\item 3 \ldots \(\mu_0 = 0\)
\item 4 \ldots \(\sigma^2 < 0\)
\end{itemize}

Parameters

<table>
<thead>
<tr>
<th>tident</th>
<th>Duration of identification [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>irtype</td>
<td>Controller type (control law)</td>
</tr>
<tr>
<td>ispeed</td>
<td>Desired closed loop speed</td>
</tr>
</tbody>
</table>

\begin{itemize}
\item \(tident\) \(\circlearrowleft 100.0\) \hspace{1cm} \text{double}
\item \(\text{iRtype}\) \(\circlearrowleft 6\) \hspace{1cm} \text{long}
\item \(\text{iSpeed}\) \(\circlearrowleft 2\) \hspace{1cm} \text{long}
\end{itemize}

\begin{itemize}
\item 1 \ldots Slow closed loop
\item 2 \ldots Normal (middle fast) closed loop
\item 3 \ldots Fast closed loop
\end{itemize}
LC – Lead compensator

Block Symbol

Function Description

The LC block is a discrete simulator of derivative element

\[ C(s) = \frac{t_d \cdot s}{n_d \cdot s + 1}, \]

where \( t_d \) is the derivative constant and \( n_d \) determines the influence of parasite 1st order filter. It is recommended to use \( 2 \leq n_d \leq 10 \). 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  
\hspace{1cm} double

Output

\( y \)  
Analog output of the block  
\hspace{1cm} double

Parameters

\( t_d \)  
Derivative time constant  
\hspace{1cm} \circ 1.0 \hspace{0.5cm} \text{double}

\( n_d \)  
Derivative filtering parameter  
\hspace{1cm} \circ 10.0 \hspace{0.5cm} \text{double}

ISSF  
Steady state at start-up

\hspace{1cm} off . . . Zero initial state
\hspace{1cm} on . . . Initial steady state  
\hspace{1cm} bool
LLC – Lead-lag compensator

Function Description

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

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

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

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

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

Input

\( u \) Analog input of the block double

Output

\( y \) Analog output of the block double

Parameters

\( \tau \) Time constant ⋅1.0 double
\( a \) Numerator time constant coefficient double
ISSF Steady state at start-up

off ... Zero initial state
on .... Initial steady state bool
MCU — Manual control unit

Block Symbol

Licence: **STANDARD**

**Function Description**

The MCU block is suitable for manual setting of the numerical output value \( y \), e.g. a setpoint. In the local mode (\( \text{LOC} = \text{on} \)) 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 \( tv \). The tracking speed is controlled by the integration time constant \( t_t \).

In the remote mode (\( \text{LOC} = \text{off} \)), the input \( rv \) 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**

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

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

Parameters

\( tt \)  Tracking time constant of the input \( tv \)  \( \odot 1.0 \text{ double} \)

\( tm \)  Initial value of integration time constant  \( \odot 100.0 \text{ double} \)

\( y0 \)  Initial output value  \( \text{double} \)

\( q \)  Multiplication quotient  \( \odot 5.0 \text{ double} \)

\( ta \)  Interval after which the rate is changed [s]  \( \odot 4.0 \text{ double} \)

\( tf \)  Interval after which the rate changes no more [s]  \( \odot 8.0 \text{ double} \)

\( SATF \)  Saturation flag  \( \text{bool} \)

\( \text{off} \ldots \text{Signal not limited} \)

\( \text{on} \ldots \ldots \text{Saturation limits active} \)

\( hilim \)  Upper limit of the output signal  \( \odot 1.0 \text{ double} \)

\( lolim \)  Lower limit of the output signal  \( \odot -1.0 \text{ double} \)
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 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>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>dv</td>
<td>Feedforward control variable</td>
<td>double</td>
</tr>
<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>
</tbody>
</table>

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

Outputs
mv Manipulated variable (controller output) double
de Deviation error double
SAT Saturation flag bool
  off ... The controller implements a linear control law
  on .... The controller output is saturated
TBSY Tuner busy flag bool
TE Tuning error bool
  off ... Autotuning successful
  on .... An error occurred during the experiment
ite Error code; expected time (in seconds) to finishing the tuning experiment while the tuning experiment is active long
  1000 .. Signal/noise ratio too low
  1001 .. Hysteresis too high
  1002 .. Too tight termination rule
  1003 .. Phase out of interval
pk Proposed controller gain double
pti Proposed integral time constant double
ptd Proposed derivative time constant double
pnd Proposed derivative component filtering double
pb Proposed weighting factor – proportional component double

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

\texttt{iainf} Type of apriori information
\begin{itemize}
  \item 1 \ldots \ldots No apriori information
  \item 2 \ldots \ldots Astatic process (process with integration)
  \item 3 \ldots \ldots Low order process
  \item 4 \ldots \ldots Static process + slow closed loop step response
  \item 5 \ldots \ldots Static process + middle fast (normal) closed loop step response
  \item 6 \ldots \ldots Static process + fast closed loop step response
\end{itemize}

\texttt{k0} Static gain of the process (must be provided in case of \texttt{iainf} = 3, 4, 5)
\begin{center}
\begin{tabular}{ll}
  \texttt{n1} & Maximum number of half-periods for estimation of frequency response point \\
  \texttt{mm} & Maximum number of half-periods for averaging \\
  \texttt{amp} & Relay controller amplitude \\
  \texttt{uhys} & Relay controller hysteresis \\
  \texttt{ntime} & Length of noise amplitude estimation period at the beginning of the tuning experiment [s] \\
  \texttt{rerrap} & Termination value of the oscillation amplitude relative error \\
  \texttt{aerrph} & Termination value of the absolute error in oscillation phase \\
  \texttt{maxdev} & Maximal admissible deviation error from the initial steady state
\end{tabular}
\end{center}

It is recommended not to change the parameters \texttt{n1}, \texttt{mm}, \texttt{ntime}, \texttt{rerrap} and \texttt{aerrph}.
PIDE – PID controller with defined static error

Block Symbol

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

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

where

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

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

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

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

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

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

Function Description

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

Inputs

- dv: Feedforward control variable
- sp: Setpoint variable
- pv: Process variable
- tv: Tracking variable
- hv: Manual value
- MAN: Manual or automatic mode
  - off: Automatic mode
  - on: Manual mode
- IH: Integrator hold
  - off: Integration enabled
  - on: Integration disabled
- ip: Parameter set index
- vp: Switching analog signal

Outputs

- mv: Manipulated variable (controller output)
- dmv: Controller velocity output (difference)
- de: Deviation error
- SAT: Saturation flag
  - off: The controller implements a linear control law
  - on: The controller output is saturated
kp  Active parameter set index  long

Parameters

hilim  Upper limit of the controller output  ⊙1.0  double
lolim  Lower limit of the controller output  ⊙-1.0  double
dz  Dead zone  double
icotype  Controller output type  ⊙1  long
  1 . . . . Analog output
  2 . . . . Pulse width modulation (PWM)
  3 . . . . Step controller unit with position feedback (SCU)
  4 . . . . Step controller unit without position feedback (SCUV)

npars  Number of controller parameter sets  ⊙6  long
GSCF  Switch parameters by analog signal vp  bool
  off . . . Index-based switching
  on . . . Analog signal based switching

hys  Hysteresis for controller parameters switching  double

irtypea  Vector of controller types (control laws)  ⊙[6 6 6 6 6 6]  byte
  1 . . . . D
  2 . . . . I
  3 . . . . ID
  4 . . . . P
  5 . . . . PD
  6 . . . . PI

RACTA  Vector of reverse action flags  ⊙[0 0 0 0 0 0]  bool
  0 . . . . Higher mv → higher pv
  1 . . . . Higher mv → lower pv

ka  Vector of controller gains $K$  ⊙[1.0 1.0 1.0 1.0 1.0 1.0]  double

tia  Vector of integral time constants $T_i$  ⊙[4.0 4.0 4.0 4.0 4.0 4.0]  double

tda  Vector of derivative time constants $T_d$  double
  ⊙[1.0 1.0 1.0 1.0 1.0 1.0]

nda  Vector of derivative filtering parameters $N$  double
  ⊙[10.0 10.0 10.0 10.0 10.0 10.0]

ba  Setpoint weighting factors – proportional part  double
  ⊙[1.0 1.0 1.0 1.0 1.0 1.0]

c a  Setpoint weighting factors – derivative part  double
  ⊙[0.0 0.0 0.0 0.0 0.0 0.0]

tta  Vector of tracking time constants. No meaning for controllers without
      integrator.  double
  ⊙[1.0 1.0 1.0 1.0 1.0 1.0]

thrsha  Vector of thresholds for switching the parameters  double
  ⊙[0.1 0.2 0.3 0.4 0.5 0]
PIDMA – PID controller with moment autotuner

Block Symbol

Function Description

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

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

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

where \( U(s) \) is Laplace transform of the manipulated variable \( mv \), \( W(s) \) is Laplace transform of the setpoint variable \( sp \), \( Y(s) \) is Laplace transform of the process variable \( pv \), \( Z(s) \) is Laplace transform of the feedforward control variable \( dv \) and \( K, T_i, T_d, N, b \) and \( c \) are the parameters of the controller. The sign of the right hand side depends on the parameter RACT. The range of the manipulated variable \( mv \) (position controller output) is limited by parameters hilim, lolim. The parameter dz determines the dead zone in the integral part of the controller. The integral part of the control law can be switched off and fixed on the current value by the integrator hold input \( IH = on \). For the proper function of the controller it is necessary to connect the output \( mv \) of the controller to the controller input \( tv \) and properly set the tracking time constant \( tt \) (the rule of thumb is \( tt \approx \sqrt{T_i T_d} \) or \( tt \approx 2 \cdot \sqrt{T_i} \) in the case of a PI controller). In this way we obtain the bumpless operation of the controller in the case of the mode switching (manual, automatic) and also the correct operation of the controller in the saturation of the output \( mv \) (antiwindup). The additional outputs \( dmv, de \) and \( SAT \) generate the velocity output (difference of \( mv \)), deviation error and saturation flag, respectively.

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

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

The block $PIDMA$ extends the control function of the standard PID controller by the built-in autotuning feature. Before start of the autotuner the operator have to reach the steady state of the process at a suitable working point (in manual or automatic mode) and specify the required type of the controller $ittype$ (PI or PID) and other tuning parameters ($aiinf$, $DGC$, $tdg$, $tn$, $amp$, $dy$ 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 \texttt{ite} indicates the autotuner phases. In the phase of estimation of the response decay rate (\texttt{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 \texttt{ite}=100.

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

**Inputs**

- \texttt{dv} Feedforward control variable \hspace{1cm} double
- \texttt{sp} Setpoint variable \hspace{1cm} double
- \texttt{pv} Process variable \hspace{1cm} double
- \texttt{tv} Tracking variable \hspace{1cm} double
- \texttt{hv} Manual value \hspace{1cm} double
- \texttt{MAN} Manual or automatic mode \hspace{1cm} bool
  - \texttt{off} ... Automatic mode
  - \texttt{on} .... Manual mode
- \texttt{IH} Integrator hold \hspace{1cm} bool
  - \texttt{off} ... Integration enabled
  - \texttt{on} .... Integration disabled
- \texttt{TUNE} Start the tuning experiment (\texttt{off}→\texttt{on}) or force transition to the next tuning phase (see the description of the \texttt{ite} output) \hspace{1cm} bool
- \texttt{TBRK} Stop the tuning experiment \hspace{1cm} bool
- \texttt{TAFF} Tuning affirmation; determines the way the computed parameters are handled \hspace{1cm} bool
  - \texttt{off} ... Parameters are only computed
  - \texttt{on} .... Parameters are set into the control law
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)
- **dmv**: Controller velocity output (difference)
- **de**: Deviation error
- **SAT**: Saturation flag
  - **off**: The controller implements a linear control law
  - **on**: The controller output is saturated
- **TBSY**: Tuner busy flag
- **TE**: Tuning error
  - **off**: Autotuning successful
  - **on**: An error occurred during the experiment
- **ite**: Error code

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

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

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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>irtype</td>
<td>Controller type (control law)</td>
<td>1 . . . . D 2 . . . . I 3 . . . . ID 4 . . . . P 5 . . . . PD 6 . . . . PID</td>
<td>6 long</td>
</tr>
<tr>
<td>RACT</td>
<td>Reverse action flag</td>
<td>off . . . Higher mv $\to$ higher pv on . . . Higher mv $\to$ lower pv</td>
<td>bool</td>
</tr>
<tr>
<td>k</td>
<td>Controller gain $K$</td>
<td>$\odot$1.0</td>
<td>double</td>
</tr>
<tr>
<td>ti</td>
<td>Integral time constant $T_i$</td>
<td>$\odot$4.0</td>
<td>double</td>
</tr>
<tr>
<td>td</td>
<td>Derivative time constant $T_d$</td>
<td>$\odot$1.0</td>
<td>double</td>
</tr>
<tr>
<td>nd</td>
<td>Derivative filtering parameter $N$</td>
<td>$\odot$10.0</td>
<td>double</td>
</tr>
<tr>
<td>b</td>
<td>Setpoint weighting – proportional part</td>
<td>$\odot$1.0</td>
<td>double</td>
</tr>
<tr>
<td>c</td>
<td>Setpoint weighting – derivative part</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>tt</td>
<td>Tracking time constant. No meaning for controllers without integrator.</td>
<td>$\odot$1.0</td>
<td>double</td>
</tr>
<tr>
<td>hilim</td>
<td>Upper limit of the controller output</td>
<td>$\odot$1.0</td>
<td>double</td>
</tr>
<tr>
<td>lolim</td>
<td>Lower limit of the controller output</td>
<td>$\odot$-1.0</td>
<td>double</td>
</tr>
<tr>
<td>dz</td>
<td>Dead zone</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>icotype</td>
<td>Controller output type</td>
<td>1 . . . . Analog output 2 . . . . Pulse width modulation (PWM) 3 . . . . Step controller unit with position feedback (SCU) 4 . . . . Step controller unit without position feedback (SCUV)</td>
<td>1 long</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Type</td>
<td>Value</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>ittype</td>
<td>Controller type to be designed</td>
<td>long</td>
<td>6 PI controller, 7 PID controller</td>
</tr>
<tr>
<td>iainf</td>
<td>Type of apriori information</td>
<td>long</td>
<td>1 Static process, 2 Astatic process</td>
</tr>
<tr>
<td>DGC</td>
<td>Drift gradient compensation</td>
<td>bool</td>
<td>on Enabled, off Disabled</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 0.1</td>
</tr>
<tr>
<td>ispeed</td>
<td>Desired closed loop speed</td>
<td>long</td>
<td>2 Slow closed loop, 3 Normal (middle fast) closed loop, 4 Fast closed loop</td>
</tr>
<tr>
<td>ipid</td>
<td>PID controller form</td>
<td>long</td>
<td>1 Parallel form, 2 Series form</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 (MAN = on), the input hv is copied to the output mv unless saturated. The overall control function of the PIDU block is quite clear from the following diagram:

**Inputs**
- \( dv \): Feedforward control variable, double
- \( sp \): Setpoint variable, double
- \( pv \): Process variable, double
- \( tv \): Tracking variable, double
- \( hv \): Manual value, double
- \( MAN \): Manual or automatic mode, bool
  - \( \text{off} \) .... Automatic mode
  - \( \text{on} \) .... Manual mode
- \( IH \): Integrator hold, bool
  - \( \text{off} \) .... Integration enabled
  - \( \text{on} \) .... Integration disabled

**Outputs**
- \( mv \): Manipulated variable (controller output), double
- \( dmv \): Controller velocity output (difference), double
- \( de \): Deviation error, double
- \( SAT \): Saturation flag, bool
  - \( \text{off} \) .... The controller implements a linear control law
  - \( \text{on} \) .... The controller output is saturated
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>irtype</strong></td>
<td>Controller type (control law)</td>
</tr>
<tr>
<td>1 .... D</td>
<td>4 .... P</td>
</tr>
<tr>
<td>2 .... I</td>
<td>5 .... PD</td>
</tr>
<tr>
<td>3 .... ID</td>
<td>6 .... PI</td>
</tr>
<tr>
<td><strong>RACT</strong></td>
<td>Reverse action flag</td>
</tr>
<tr>
<td><em>off</em> .... Higher mv $\rightarrow$ higher pv</td>
<td></td>
</tr>
<tr>
<td><em>on</em> .... Higher mv $\rightarrow$ lower pv</td>
<td></td>
</tr>
<tr>
<td><strong>k</strong></td>
<td>Controller gain $K$</td>
</tr>
<tr>
<td><strong>ti</strong></td>
<td>Integral time constant $T_i$</td>
</tr>
<tr>
<td><strong>td</strong></td>
<td>Derivative time constant $T_d$</td>
</tr>
<tr>
<td><strong>nd</strong></td>
<td>Derivative filtering parameter $N$</td>
</tr>
<tr>
<td><strong>b</strong></td>
<td>Setpoint weighting – proportional part</td>
</tr>
<tr>
<td><strong>c</strong></td>
<td>Setpoint weighting – derivative part</td>
</tr>
<tr>
<td><strong>tt</strong></td>
<td>Tracking time constant. No meaning for controllers without integrator.</td>
</tr>
<tr>
<td><strong>hilim</strong></td>
<td>Upper limit of the controller output</td>
</tr>
<tr>
<td><strong>lolim</strong></td>
<td>Lower limit of the controller output</td>
</tr>
<tr>
<td><strong>dz</strong></td>
<td>Dead zone</td>
</tr>
<tr>
<td><strong>icotype</strong></td>
<td>Controller output type</td>
</tr>
<tr>
<td>1 .... Analog output</td>
<td></td>
</tr>
<tr>
<td>2 .... Pulse width modulation (PWM)</td>
<td></td>
</tr>
<tr>
<td>3 .... Step controller unit with position feedback (SCU)</td>
<td></td>
</tr>
<tr>
<td>4 .... Step controller unit without position feedback (SCUV)</td>
<td></td>
</tr>
</tbody>
</table>
PIDUI – PID controller unit with variable parameters

Block Symbol

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

Outputs

- **mv**: Manipulated variable (controller output)
- **dmv**: Controller velocity output (difference)
- **de**: Deviation error
SAT Saturation flag
- 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>6 long</td>
</tr>
<tr>
<td>RACT</td>
<td>Reverse action flag</td>
<td>bool</td>
</tr>
<tr>
<td>tt</td>
<td>Tracking time constant. No meaning for controllers without integrator.</td>
<td>double</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>
</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)
**POUT – Pulse output**

**Block Symbol**

```
U  Y
```

**Licence:** STANDARD

**Function Description**

The POUT block shapes the input pulses \( U \) in such a way, that the output pulse \( Y \) has a duration of at least \( \text{dtime} \) 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] \( \circ 1.0 \) double
- \( \text{btime} \) Minimum delay between two successive output pulses [s] \( \circ 1.0 \) double
PRGM – Setpoint programmer

Function Description

The PRGM block generates functions of time (programs) composed of \( n \) linear parts defined by \((n+1)\)-dimensional vectors of time \((t_m = [t_0, \ldots, t_n])\) 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 \(\rightarrow\) 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 \(\rightarrow\) off). If RPT = on, the program is generated repeatedly.

Inputs

- **RUN** Enable execution \( \text{bool} \)
- **DEF** Initialize sp to the value of spv \( \text{bool} \)
- **spv** Initializing constant \( \text{double} \)
- **HLD** Output and timer freezing \( \text{bool} \)
### CHAPTER 7. REG – FUNCTION BLOCKS FOR CONTROL

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CON</td>
<td>Continue from defined node</td>
<td>bool</td>
</tr>
<tr>
<td>ind</td>
<td>Index of the node to continue from</td>
<td>long</td>
</tr>
<tr>
<td>trt</td>
<td>Time to reach the defined node with index ind</td>
<td>double</td>
</tr>
<tr>
<td>RPT</td>
<td>Repetition flag</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>sp</td>
<td>Setpoint variable (function value of the time function at given time)</td>
<td>double</td>
</tr>
<tr>
<td>isc</td>
<td>Current function sector</td>
<td>long</td>
</tr>
<tr>
<td>tsc</td>
<td>Time elapsed since the start of current sector</td>
<td>double</td>
</tr>
<tr>
<td>tt</td>
<td>Time elapsed since the start of program generation</td>
<td>double</td>
</tr>
<tr>
<td>rt</td>
<td>Remaining time till the end of program</td>
<td>double</td>
</tr>
<tr>
<td>CNF</td>
<td>Flag indicating that the configured curve is being followed</td>
<td>bool</td>
</tr>
<tr>
<td>E</td>
<td>Error flag – the node times are not ascending</td>
<td>bool</td>
</tr>
</tbody>
</table>

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Number of sectors</td>
<td>long</td>
</tr>
<tr>
<td>tmunits</td>
<td>Time units</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>1 ..... seconds</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2 ..... minutes</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3 ..... hours</td>
<td>3</td>
</tr>
<tr>
<td>tm</td>
<td>(n + 1)-dimensional vector of ascending node times</td>
<td>double</td>
</tr>
<tr>
<td>y</td>
<td>(n + 1)-dimensional vector of node values (values of the time function)</td>
<td>double</td>
</tr>
</tbody>
</table>

- 10000000 ⊙ 2 long
- 1 . . . . . seconds
- 2 . . . . . minutes
- 3 . . . . . hours
- [0 1 2] double
- [0 1 0] double
**PSMPC – Pulse-step model predictive controller**

**Block Symbol**

![Block Symbol](image)

**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 \( \kappa, \mu, \) and \( \sigma^2 \), which can be obtained easily from a very short
and simple experiment. The controlled system can be approximated by first order plus
death-time system

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

(7.1)
or second order plus death-time system

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

(7.2)

with the same characteristic numbers. The type of approximation is selected by the
imtype parameter.

To lower the computational burden of the open-loop optimization, the family of
admissible control sequences contains only sequences in the so-called pulse-step shape
depicted below:

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

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

(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>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nc</code></td>
<td>Control horizon length ($N_C$)</td>
<td>long</td>
</tr>
<tr>
<td><code>np1</code></td>
<td>Start of coincidence interval ($N_1$)</td>
<td>long</td>
</tr>
<tr>
<td><code>np2</code></td>
<td>End of coincidence interval ($N_2$)</td>
<td>long</td>
</tr>
<tr>
<td><code>lambda</code></td>
<td>Control signal penalization coefficient ($\lambda$)</td>
<td>double</td>
</tr>
<tr>
<td><code>umax</code></td>
<td>Upper limit of the controller output ($u^+$)</td>
<td>double</td>
</tr>
<tr>
<td><code>umin</code></td>
<td>Lower limit of the controller output ($u^-$)</td>
<td>double</td>
</tr>
<tr>
<td><code>imtype</code></td>
<td>Controlled process model type</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>1 .... FOPDT model (7.1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 .... SOPDT model (7.2)</td>
<td></td>
</tr>
</tbody>
</table>
|          | 3 .... Discrete step response | <>
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \kappa )</td>
<td>Static gain (( \kappa ))</td>
<td>( \odot 1.0 ) double</td>
</tr>
<tr>
<td>( \mu )</td>
<td>Resident time constant (( \mu ))</td>
<td>( \odot 20.0 ) double</td>
</tr>
<tr>
<td>( \sigma )</td>
<td>Measure of the system response length (( \sqrt{\sigma^2} ))</td>
<td>( \odot 10.0 ) double</td>
</tr>
</tbody>
</table>
| \( nsr \)     | Length of the discrete step response (\( N \)), see the warning above | long  
\( \downarrow 10 \uparrow 1000000 \) \( \odot 11 \) |
| \( sr \)      | Discrete step response sequence ([\( g(1), \ldots, g(N) \)])    | double    |
|                | \( \odot [0 \ 0.2642 \ 0.5940 \ 0.8009 \ 0.9084 \ 0.9596 \ 0.9826 \ 0.9927 \ 0.9970 \ 0.9988 \ 0.9995] \) |
**PWM – Pulse width modulation**

**Block Symbol**

![PWM block symbol](image)

**Licence:** STANDARD

**Function Description**

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

\[
L = \text{pertm} \cdot |u|,
\]

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

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

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

\begin{itemize}
  \item \texttt{pertm} Modulation period length [s] \hspace{1cm} \circ10.0 double
  \item \texttt{dtime} Minimum width of the output pulse [s] \hspace{1cm} \circ0.1 double
  \item \texttt{btime} Minimum delay between output pulses [s] \hspace{1cm} \circ0.1 double
  \item \texttt{offtime} Minimum delay when altering direction [s] \hspace{1cm} \circ1.0 double
  \item \texttt{asyfac} Asymmetry factor \hspace{1cm} \circ1.0 double
  \item \texttt{SYNCH} Synchronization flag of the period start
    \begin{itemize}
      \item \texttt{off} ... Synchronization disabled
      \item \texttt{on} ..... Synchronization enabled
    \end{itemize}
\end{itemize}
RLY – Relay with hysteresis

Block Symbol

Function Description

The RLY block transforms the input signal \( u \) to the output signal \( y \) according to the figure below.

Input

\( u \)  
Analog input of the block  
double

Output

\( y \)  
Analog output of the block  
double

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( ep )</td>
<td>The value ( u &gt; ep ) causes ( y = ap ) (&quot;On&quot;)</td>
<td>( \circ 1.0 )</td>
<td>double</td>
</tr>
<tr>
<td>( en )</td>
<td>The value ( u &lt; en ) causes ( y = an ) (&quot;Off&quot;)</td>
<td>( \circ -1.0 )</td>
<td>double</td>
</tr>
<tr>
<td>( ap )</td>
<td>Output value ( y ) in the &quot;On&quot; state</td>
<td>( \circ 1.0 )</td>
<td>double</td>
</tr>
<tr>
<td>( an )</td>
<td>Output value ( y ) in the &quot;Off&quot; state</td>
<td>( \circ -1.0 )</td>
<td>double</td>
</tr>
<tr>
<td>( y0 )</td>
<td>Initial output value at start-up</td>
<td></td>
<td>double</td>
</tr>
</tbody>
</table>
SAT – Saturation with variable limits

Block Symbol Licence: STANDARD

Function Description

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

Outputs

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

Parameters

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

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

Block Symbol  Licence: AUTOTUNING

Function Description

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

$$F(s) = \frac{b_1 s + b_0}{s^2 + 2\xi\Omega s + \Omega^2},$$  \hspace{1cm} (7.4)$$

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

The "Identification and design mode" is activated by the binary input $\text{ID} = \text{on}$. Two points of frequency response with given phase delay are measured during the identification experiment. Based on these two points a model of the controlled system is built. The experiment itself is initiated by the rising edge of the $\text{RUN}$ input. A harmonic signal with amplitude $\text{amp}$, frequency $\omega$ and bias $\text{bias}$ then appears at the output $m_v$. The frequency runs through the interval $\langle \omega_b, \omega_f \rangle$, it increases gradually. The current frequency is copied to the output $w$. The rate at which the frequency changes (sweeping) is determined by the $\text{cp}$ parameter, which defines the relative shrinking of the initial period $T_b = \frac{2\pi}{\omega_b}$ of the exciting sine wave in time $T_b$, thus

$$c_p = \frac{\omega_b}{\omega(T_b)} = \frac{\omega_b}{\omega_b e^{-\gamma T_b}} = e^{-\gamma T_b}.$$  \hspace{1cm}$$

The $c_p$ parameter usually lies within the interval $c_p \in (0.95; 1)$. The lower the damping coefficient $\xi$ of the controlled system is, the closer to one the $c_p$ parameter must be.
At the beginning of the identification period the exciting signal has a frequency of $\omega = \omega_b$. After a period of $\text{stime}$ seconds the estimation of current frequency response point starts. Its real and imaginary parts are available at the $\text{xre}$ and $\text{xim}$ outputs. If the $\text{MANF}$ parameter is set to 0, then the frequency sweeping is stopped two times during the identification period. This happens when points with phase delay of $\phi_1$ and $\phi_2$ are reached for the first time. The breaks are $\text{stime}$ seconds long. Default phase delay values are $-60^\circ$ and $-120^\circ$, respectively, but these can be changed to arbitrary values within the interval $(-360^\circ, 0^\circ)$, where $\phi_1 > \phi_2$. At the end of each break an arithmetic average is computed from the last $\text{iavg}$ frequency point estimates. Thus we get two points of frequency response which are successively used to compute the controlled process model in the form of (7.4). If the $\text{MANF}$ parameter is set to 1, then the selection of two frequency response points is manual. To select the frequency, set the input $\text{HLD} = \text{on}$, which stops the frequency sweeping. The identification experiment continues after returning the input $\text{HLD}$ to 0. The remaining functionality is unchanged.

It is possible to terminate the identification experiment prematurely in case of necessity by the input $\text{BRK} = \text{on}$. If the two points of frequency response are already identified at that moment, the controller parameters are designed in a standard way. Otherwise the controller design cannot be performed and the identification error is indicated by the output signal $\text{IDE} = \text{on}$.

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

The computed state controller parameters are taken over by the control algorithm as soon as the $\text{SETC}$ input is set to 1 (i.e. immediately if $\text{SETC}$ is constantly set to 0). The identified model and controller parameters can be obtained from the $\text{p1}$, $\text{p2}$, ..., $\text{p6}$ outputs after setting the $\text{ips}$ input to the appropriate value. After a successful identification it is possible to generate the frequency response of the controlled system model, which is initiated by a rising edge at the $\text{MFR}$ input. The frequency response can be read from the $\text{w}$, $\text{xre}$ and $\text{xim}$ outputs, which allows easy confrontation of the model and the measured data.

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

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

and

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

The coefficients of these transfer functions can be found at the outputs p1,...,p6 after the identification experiment (IDBSY = off). The output signals meaning is switched when a change occurs at the ips input.

**Inputs**

- **dv**  Feedforward control variable  
- **sp**  Setpoint variable  
- **pv**  Process variable  
- **tv**  Tracking variable  
- **hv**  Manual value  

All inputs are of type `double`.
### CHAPTER 7. REG – FUNCTION BLOCKS FOR CONTROL

<table>
<thead>
<tr>
<th>MAN</th>
<th>Manual or automatic mode</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>off</strong> ... Automatic mode <strong>on</strong> ... Manual mode</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Identification or controller operating mode</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td><strong>off</strong> ... Controller mode <strong>on</strong> ... Identification and design</td>
<td></td>
</tr>
<tr>
<td>TUNE</td>
<td>Start the tuning experiment (<strong>off</strong>→<strong>on</strong>), the exciting harmonic signal is generated</td>
<td>bool</td>
</tr>
<tr>
<td>HLD</td>
<td>Stop frequency sweeping</td>
<td>bool</td>
</tr>
<tr>
<td>BRK</td>
<td>Termination signal</td>
<td>bool</td>
</tr>
<tr>
<td>SETC</td>
<td>Flag for accepting the new controller parameters and updating the control law</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td><strong>off</strong> ... Parameters are only computed</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>on</strong> ... Parameters are accepted as soon as computed</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>off</strong>→<strong>on</strong> One-shot confirmation of the computed parameters</td>
<td></td>
</tr>
<tr>
<td>ips</td>
<td>Switch for changing the meaning of the output signals</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>0 ... Two points of frequency response</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p1 ... frequency of the 1st measured point in rad/s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p2 ... real part of the 1st point</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p3 ... imaginary part of the 1st point</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p4 ... frequency of the 2nd measured point in rad/s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p5 ... real part of the 2nd point</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p6 ... imaginary part of the 2nd point</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 ... Second order model in the form (7.5)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p1 ... b₁ parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p2 ... b₀ parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p3 ... a₁ parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p4 ... a₀ parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ... Second order model in the form (7.6)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p1 ... K₀ parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p2 ... τ parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p3 ... Ω parameter in rad/s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p4 ... ξ parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p5 ... Ω parameter in Hz</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p6 ... resonance frequency in Hz</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 ... State feedback parameters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p1 ... f₁ parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p2 ... f₂ parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p3 ... f₃ parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p4 ... f₄ parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p5 ... f₅ parameter</td>
<td></td>
</tr>
<tr>
<td>MFR</td>
<td>Generation of the parametric model frequency response at the w, xre, and xim outputs (<strong>off</strong>→<strong>on</strong> triggers the generator)</td>
<td>bool</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>
</tbody>
</table>
SAT Saturation flag
  off ... The controller implements a linear control law
  on ..... The controller output is saturated
IDBSY Identification running
  off ... Identification not running
  on ..... Identification in progress
w Frequency response point estimate - frequency in rad/s double
xre Frequency response point estimate - real part double
xim Frequency response point estimate - imaginary part double
epv Reconstructed pv signal double
IDE Identification error indicator
  off ... Successful identification experiment
  on ..... Identification error occurred
iIDE Error code long
  101 ... Sampling period too low
  102 ... Error identifying one or both frequency response point(s)
  103 ... Manipulated variable saturation occurred during the identification experiment
  104 ... Invalid process model
pl..p6 Results of identification and design phase double

Parameters

ubias Static component of the exciting harmonic signal double
uamp Amplitude of the exciting harmonic signal 0.1 double
wb Frequency interval lower limit [rad/s] 0.1 double
wf Frequency interval upper limit [rad/s] 10.0 double
isweep Frequency sweeping mode
  1 .... Logarithmic
  2 .... Linear (not implemented yet)
cp Sweeping rate \( \downarrow 0.5 \uparrow 1.0 \odot 0.995 \) double
iavg Number of values for averaging 10 long
alpha Relative positioning of the observer poles (in identification phase) double
  \( \odot 2.0 \)
xi Observer damping coefficient (in identification phase) 0.707 double
MANF Manual frequency response points selection bool
  off ... Disabled
  on ..... Enabled
ph1 Phase delay of the 1st point in degrees \( \odot -60.0 \) double
ph2 Phase delay of the 2nd point in degrees \( \odot -120.0 \) double
stme Settling period [s] 10 double
ralpha Relative positioning of the observer poles 4 double
rx1 Observer damping coefficient 0.707 double
acl1 Relative positioning of the 1st closed-loop poles couple 1 double
xicl1 Damping of the 1st closed-loop poles couple 0.707 double
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTGF</td>
<td>Integrator flag</td>
<td>on</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... State-space model without integrator</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... Integrator included in the state-space model</td>
<td></td>
<td></td>
</tr>
<tr>
<td>apcl</td>
<td>Relative position of the real pole</td>
<td>1.0</td>
<td>double</td>
</tr>
<tr>
<td>DISF</td>
<td>Disturbance flag</td>
<td>off</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>on .... State space model without disturbance model</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... Disturbance model is included in the state space model</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dom</td>
<td>Disturbance model natural frequency</td>
<td>1.0</td>
<td>double</td>
</tr>
<tr>
<td>dxi</td>
<td>Disturbance model damping coefficient</td>
<td></td>
<td>double</td>
</tr>
<tr>
<td>acl2</td>
<td>Relative positioning of the 2nd closed-loop poles couple</td>
<td>2.0</td>
<td>double</td>
</tr>
<tr>
<td>xicl2</td>
<td>Damping of the 2nd closed-loop poles couple</td>
<td>0.707</td>
<td>double</td>
</tr>
<tr>
<td>tt</td>
<td>Tracking time constant</td>
<td>1.0</td>
<td>double</td>
</tr>
<tr>
<td>hilim</td>
<td>Upper limit of the controller output</td>
<td>1.0</td>
<td>double</td>
</tr>
<tr>
<td>lolim</td>
<td>Lower limit of the controller output</td>
<td>-1.0</td>
<td>double</td>
</tr>
<tr>
<td>mb1p</td>
<td>Controlled system transfer function coefficient $b_1$</td>
<td></td>
<td>double</td>
</tr>
<tr>
<td>mb0p</td>
<td>Controlled system transfer function coefficient $b_0$</td>
<td>1.0</td>
<td>double</td>
</tr>
<tr>
<td>maip</td>
<td>Controlled system transfer function coefficient $a_1$</td>
<td>0.2</td>
<td>double</td>
</tr>
<tr>
<td>ma0p</td>
<td>Controlled system transfer function coefficient $a_0$</td>
<td>1.0</td>
<td>double</td>
</tr>
</tbody>
</table>
**SCU – Step controller with position feedback**

**Block Symbol**

![Block Symbol]

**Licence:** STANDARD

**Function Description**

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

The **SCU** block processes the control deviation \(sp - pv\) by a three state element with parameters (thresholds) \(\text{thron}\) and \(\text{throff}\) (see the **TSE** block, use parameters \(\text{ep} = \text{thron}, \text{epoff} = \text{throff}, \text{en} = -\text{thron} \) and \(\text{enoff} = -\text{throff}\)). The parameter \(\text{RACT}\) determines whether the \(\text{UP}\) or \(\text{DN}\) pulse is generated for positive or negative value of the controller deviation. Two pulse outputs of the three state element are further shaped so that minimum pulse duration \(\text{dt ime}\) and minimum pulse break time \(\text{btime}\) are guaranteed at the block \(\text{UP}\) and \(\text{DN}\) outputs. If signals from high and low limit switches of the valve are available, they should be connected to the \(\text{HS}\) and \(\text{LS}\) inputs.

There is also a group of input signals for manual control available. The manual mode is activated by the \(\text{MAN} = \text{on}\) input signal. Then it is possible to move the motor back and forth by the \(\text{MUP}\) and \(\text{MDN}\) input signals. It is also possible to specify a position increment/decrement request by the \(\text{mdv}\) input. In this case the request must be confirmed by a rising edge (\(\text{off} \rightarrow \text{on}\)) in the \(\text{DVC}\) input signal.

The control function of the **SCU** block is quite clear from the following diagram.

![Control Diagram]

The complete structure of the three-state step controller is depicted in the following
CHAPTER 7. REG – FUNCTION BLOCKS FOR CONTROL

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
  - 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 \( \downarrow 0.0 \circ 0.02 \text{ double} \)
- **thoff**: Switch-off value \( \downarrow 0.0 \circ 0.01 \text{ double} \)
- **dtime**: Minimum width of the output pulse [s] \( \downarrow 0.0 \circ 0.1 \text{ double} \)
- **btime**: Minimum delay between two subsequent output pulses [s] to do \( \downarrow 0.0 \circ 0.1 \text{ double} \)
- **RACT**: Reverse action flag
  - off ... Higher mv → higher pv
  - on .... Higher mv → lower pv
Motor time constant (determines the time during which the motor position changes by one unit)
CHAPTER 7. REG – FUNCTION BLOCKS FOR CONTROL

SCUV – Step controller unit with velocity input

Block Symbol

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

Function Description

The block SCUV substitutes the secondary position controller SCU in the step controller loop when the position signal is not available. The primary controller PIDU (or some of the derived function blocks) is connected with the block SCUV using the block inputs \( mv, \) \( dmv \) and \( SAT \).

If the primary controller uses PI or PID control law (\( \text{CWOI} = \text{off} \)), 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 \( \text{thron} \) and \( \text{throff} \) (see the TSE block, use parameters \( \text{ep} = \text{thron}, \text{epoff} = \text{throff}, \text{en} = -\text{thron} \) 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 \( dtime \) and minimum pulse break time \( btime \) are guaranteed at the block outputs \( \text{UP} \) and \( \text{DN} \). The parameter \( \text{RACT} \) 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 \( \text{icotype} = 4 \) (working in automatic mode) is less than its dead zone (\( \text{SAT} = \text{on} \)), then the output of the corresponding internal integrator is set to zero.

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

If the primary controller uses P or PD control law (\( \text{CWOI} = \text{on} \)), then the deviation error of the primary controller can be eliminated by the bias \( \text{ub} \) manually. In this case, the control algorithm is slightly modified, the position of the motor \( \text{pos} \) is used and the proper settings of \( \text{thron}, \text{throff} \) and the tracking time constant \( \text{tt} \) 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 \( \text{MAN} = \text{on} \) input signal. Then it is possible to move the motor back and forth by the \( \text{MUP} \) and \( \text{MDN} \) input signals. It is also possible to specify a position increment/decrement request by the \( \text{mdv} \) input. In this case the request must be confirmed by a rising edge (\( \text{off} \rightarrow \text{on} \)) in the \( \text{DVC} \) input signal.
The overall control function of the SCUV block is obvious from the following diagram:

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

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

**Primary controller without integration: P, PD**

### Inputs

- `mv` Manipulated variable (controller output) double
- `dmv` Controller velocity output (difference) double
### CHAPTER 7. REG – FUNCTION BLOCKS FOR CONTROL

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

** Outputs **

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

** Parameters **

- **thron** Switch-on value | \( \downarrow 0.0 \circ 0.02 \) | double |
- **thoff** Switch-off value | \( \downarrow 0.0 \circ 0.01 \) | double |
- **dtime** Minimum width of the output pulse [s] | \( \downarrow 0.0 \circ 0.1 \) | double |
- **btime** Minimum delay between two subsequent output pulses [s] | \( \downarrow 0.0 \circ 0.1 \) | double |
- **RACT** Reverse action flag | bool |
  - **off** ... Higher \( mv \rightarrow \) higher \( pv \)
  - **on** .... Higher \( mv \rightarrow \) lower \( pv \)
- **trun** Motor time constant (determines the time during which the motor position changes by one unit) | \( \downarrow 0.0 \circ 10.0 \) | double |
- **CWOI** Controller without integration flag | bool |
  - **off** ... The primary controller has an integrator (I, PI, PID)
  - **on** .... The primary controller does not have an integrator (P, PD)
- **tt** Tracking time constant | \( \downarrow 0.0 \circ 1.0 \) | double |
**SELU – Controller selector unit**

**Block Symbol**

```
+-----------------+-----------------+-----------------+-----------------+
|                 |                 |                 |                 |
|     u1           |     u2           |     u3           |     u4           |
|     iSW          |     SW1          |     SW2          |     y            |
|     U1           |     U2           |     U3           |     U4           |
|                 |                 |                 |                 |
+-----------------+-----------------+-----------------+-----------------+
```

**Licence:** STANDARD

**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 $BINF = \text{off}$ the active signal is selected by the $iSW$ input. In the case of $BINF = \text{on}$ the selection is based on the binary inputs $SW1$ and $SW2$ according to the following table:

<table>
<thead>
<tr>
<th>$iSW$</th>
<th>$SW1$</th>
<th>$SW2$</th>
<th>$y$</th>
<th>$U1$</th>
<th>$U2$</th>
<th>$U3$</th>
<th>$U4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>off</td>
<td>off</td>
<td>$u_1$</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 SWU blocks).

**Inputs**

- **$u_1...u_4$** Signals to be selected from $double$
- **$iSW$** Active signal selector in case of $BINF = \text{off}$ $long$
- **$SW1$** Binary signal selector, used when $BINF = \text{on}$ $bool$
- **$SW2$** Binary signal selector, used when $BINF = \text{on}$ $bool$

**Outputs**

- **$y$** Analog output of the block $double$
- **$U_1...U_4$** Binary output signal for selector control $bool$

**Parameter**

- **$BINF$** Enable the binary selectors $bool$
  - **off** ... Disabled (analog selector)
  - **on** .... Enabled (binary selectors)
SMHCC – Sliding mode heating/cooling controller

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:

$$T_S = 0.1, \ i_{pwmc} = 100, \ T_C = 10s, \ T_R = 0.01s.$$
The control law of the block SMHCC in automatic mode (MAN = off) is based on the discrete dynamic sliding mode control technique and special 3rd order filters for estimation of the first and second derivatives of the control error.

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

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

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

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

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

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

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 \( \pi/(5 TC) \).

The manipulated variable \( \text{mv} \) usually ranges in the interval \([-1, 1]\). The positive (negative) value corresponds to heating (cooling). For example, \( \text{mv} = 1 \) means the full heating. The limits of \( \text{mv} \) can be reduced when needed by the controller parameters \( \text{hilim}_p \) and \( \text{hilim}_m \). This reduction is probably necessary when the asymmetry between heating and cooling is significant. For example, if in the working zone the cooling is much more aggressive than heating, then these parameters should be set as \( \text{hilim}_p = 1 \) and \( \text{hilim}_m < 1 \). If we want to apply such limitation only in some time interval after a change of setpoint (during the transient response) then it is necessary to set initial value of the heating (cooling) action amplitude \( u_0_p \) (\( u_0_m \)) to the suitable value less than \( \text{hilim}_p \) (\( \text{hilim}_m \)). Otherwise set \( u_0_p = \text{hilim}_p \) and \( u_0_m = \text{hilim}_m \).

The current amplitudes of heating and cooling \( \text{uk}_p, \text{uk}_m \), respectively, are automatically adapted by the special algorithm to achieve so called quasi sliding mode, where the sign of \( s_k \) alternately changes its value. In such a case the controller output \( \text{i sv} \) alternates the values 1 and -1. The rate of adaptation of the heating (cooling) amplitude is given by the time constant \( \tau_{up} \) (\( \tau_{um} \)). Both of these time constants have to be sufficiently high to provide the proper function of adaptation but the fine tuning is not necessary.
CHAPTER 7. REG – FUNCTION BLOCKS FOR CONTROL

Note for completeness that the manipulated variable \( mv \) is determined from the action amplitudes \( uk_p, uk_m \) by the following expression

\[
\text{if } (s_k < 0.0) \text{ then } \text{mv} = uk_p \text{ else } \text{mv} = -uk_m.
\]

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 \( mv \).

**Inputs**

- \( sp \) setpoint variable
- \( pv \) process variable
- \( hv \) manual value
- \( MAN \) controller mode
  - 0 ..... automatic mode
  - 1 ..... manual mode

**Outputs**

- \( mv \) manipulated variable (position controller output)
- \( mve \) equivalent manipulated variable
- \( de \) deviation error
- \( SAT \) saturation flag
  - 0 ..... the controller implements a linear control law
  - 1 ..... the controller output is saturated, \( mv \geq hilim_p \) or \( mv \leq -hilim_m \)
- \( isv \) number of the positive (+) or negative (−) sliding variable steps
- \( t_{ukp} \) current amplitude of heating
- \( t_{ukm} \) current amplitude of cooling
- \( t_{sk} \) discrete dynamic sliding variable \( s_k \)
- \( t_{pv} \) filtered control error -\( de \)
- \( t_{dpv} \) filtered first derivative of the control error \( t_{ek} \)
- \( t_{d2pv} \) filtered second derivative of the control error \( t_{ek} \)

**Parameters**

- \( ipwmc \) PWM cycle in the sampling periods of SMHCC \( (T_c/T_S) \)
- \( xi \) relative damping \( \xi \) of sliding zero dynamics \( xi \geq 0 \)
- \( om \) natural frequency \( \Omega \) of sliding zero dynamics \( (0.0) \)
- \( taup \) time constant for adaptation of heating action amplitude in seconds
- \( taum \) time constant for adaptation of cooling action amplitude in seconds
- \( beta \) bandwidth parameter of the derivative filter
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>hilim_p</td>
<td>high limit of the heating action amplitude</td>
<td>↓0.0 ↑1.0</td>
<td>double</td>
</tr>
<tr>
<td>hilim_m</td>
<td>high limit of the cooling action amplitude</td>
<td>↓0.0 ↑1.0</td>
<td>double</td>
</tr>
<tr>
<td>u0_p</td>
<td>initial value of the heating action amplitude after setpoint change and start of the block</td>
<td></td>
<td>double</td>
</tr>
<tr>
<td>u0_m</td>
<td>initial value of the cooling action amplitude after setpoint change and start of the block</td>
<td></td>
<td>double</td>
</tr>
<tr>
<td>sp_dif</td>
<td>Setpoint difference threshold</td>
<td>⊙10.0</td>
<td>double</td>
</tr>
<tr>
<td>tauf</td>
<td>Equivalent manipulated variable filter time constant</td>
<td>⊙400.0</td>
<td>double</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 $T_C = i_{parmc} T_S$ 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:

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

Notice however that for a faster controlled system the sampling periods $T_S$, $T_C$ and
$T_R$ must be shortened! More precisely, the three minimal time constant of the process
are important for selection of these time periods (all real thermal process has at least
three time constants). For example, the sampling period $T_S = 0.1$ is sufficiently short for
such processes that have at least three time constants, the minimal of them is greater
than 10s and the maximal is greater than 100s. For the proper function of the controller
it is necessary that these time parameters are suitably chosen by the user according
to the actual dynamics of the process! If SMHCCA is implemented on a processor with floating point arithmetic then the accurate setting of the sampling periods $T_S$, $T_C$, $T_R$
and the parameter $\beta$ is critical for correct function of the controller. Also, some other
parameters with the clear meaning described below have to be chosen manually. All the
remaining parameters ($x_i$, $om$, $taup$, $taum$, $tauf$) 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 pro-
cesses and is not implemented yet (So far, this method has been developed and
tested in Simulink only).

Despite the fact that the first method of the tuning is based only on the heating
regime, the resulting parameters are usually satisfactory for both heating and cooling
regimes because of the strong robustness of sliding mode control. The tuning proce-
dure is very quick and can be accomplished during the normal rise time period of the
process temperature from cold state to the setpoint usually without any temporization
or degradation of control performance. Thus the tuning procedure can be included in
every start up from cold state to the working point specified by the sufficiently high
temperature setpoint. Now the implemented procedure will be described in detail. The
 tuning procedure starts in the tuning mode or in the manual mode. If the tuning mode
($TMODE = on$) is selected the manipulated variable $mv$ is automatically set to zero and
the output $TBSY$ is set to 1 for indication of the tuning stage of the controller. The cold
state of the process is preserved until the initialization pulse is applied to the input $TUNE$
($0 \rightarrow 1$). After some time (depending on $\beta$), when the noise amplitude is estimated,
the heating is switched on with the amplitude given by the parameter $ut_p$. The process temperature $pv$ and its two derivatives (outputs $t_pv$, $t_dpv$, $t_d2pv$) are observed to obtain the optimal parameters of the controller. If the tuning procedure ends without errors, then $TBSY$ is set to 0 and the controller begins to work in manual or automatic mode according to the input $MAN$. If $MAN = \text{off}$ 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 $TAFF = \text{off}$, 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 $MAN = \text{off}$ 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 ($MAN = \text{off}$) with any constant value of the input $hv$. However, the steady state must be provided in this case. Again, the tuning is started by the initialization pulse at the input $TUNE (0 \rightarrow 1)$ and after the stop of tuning the controller continues in the manual mode. In both cases the resulting parameters appear on the outputs $p1,\ldots,p6$. 

![Diagram](image-url)
The control law of the block SMHCCA in automatic mode (MAN = off) is based on the discrete dynamic sliding mode control technique and special 3rd order filters for estimation of the first and second derivatives of the control error.

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

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

is forced to zero. In the above definition of the sliding variable, \( e_k, \dot{e}_k, \ddot{e}_k \) denote the filtered deviation error (pv – sp) and its first and second derivatives in the control period \( k \), respectively, and \( \xi, \Omega \) are the control parameters described below. In the second phase, \( s_k \) is hold at the zero value (the sliding phase) by the proper control "bangs". Here, the heating action is alternated by cooling action and vice versa rapidly. The amplitudes of control actions are adapted appropriately to guarantee \( s_k = 0 \) approximately. Thus, the hypothetical continuous dynamic sliding variable

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

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

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

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

The manipulated variable \( \text{mv} \) usually ranges in the interval \([-1, 1]\). The positive (negative) value corresponds to heating (cooling). For example, \( \text{mv} = 1 \) means the full heating. The limits of \( \text{mv} \) can be reduced when needed by the controller parameters \( \text{hilim}_p \).
and \( \text{hilim}_m \). This reduction is probably necessary when the asymmetry between heating and cooling is significant. For example, if in the working zone the cooling is much more aggressive than heating, then these parameters should be set as \( \text{hilim}_p = 1 \) and \( \text{hilim}_m < 1 \). If we want to apply such limitation only in some time interval after a change of setpoint (during the transient response) then it is necessary to set initial value of the heating (cooling) action amplitude \( u_0.p \) (\( u_0.m \)) to the suitable value less than \( \text{hilim}_p \) (\( \text{hilim}_m \)). Otherwise set \( u_0.p = \text{hilim}_p \) and \( u_0.m = \text{hilim}_m \).

The current amplitudes of heating and cooling \( u_k.p \), \( u_k.m \), 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 \( \text{isv} \) alternates the values 1 and \(-1\). The rate of adaptation of the heating (cooling) amplitude is given by time constant \( \text{taup} \) (\( \text{taum} \)). Both of these time constants have to be sufficiently high to provide the proper function of adaptation but the fine tuning is not necessary. Note for completeness that the manipulated variable \( \text{mv} \) is determined from the action amplitudes \( u_k.p \), \( u_k.m \) by the following expression

\[
\text{if } (s_k < 0.0) \ \text{then } \text{mv} = u_k.p \ \text{else } \text{mv} = -u_k.m.
\]

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

In the manual mode (\( \text{MAN} = \text{on} \)) the controller input \( \text{hv} \) is (after limitation to the range \([-\text{hilim}_m, \text{hilim}_p]\)) copied to the manipulated variable \( \text{mv} \). The controller output \( \text{mve} \) provides the equivalent amplitude-modulated value of the manipulated variable \( \text{mv} \) for informative purposes. The output \( \text{mve} \) is obtained by the first order filter with the time constant \( \text{tauf} \) applied to \( \text{mv} \).

**Inputs**

- **sp**: Setpoint variable, \( \text{double} \)
- **pv**: Process variable, \( \text{double} \)
- **hv**: Manual value, \( \text{double} \)
- **MAN**: Manual or automatic mode, \( \text{bool} \)
  - 0 .... Automatic mode 1 .... Manual mode
- **TMODE**: Tuning mode, \( \text{bool} \)
- **TUNE**: Start the tuning experiment: \( \text{TUNE off→on} \), \( \text{bool} \)
- **TBRK**: Stop the tuning experiment: \( \text{TBRK off→on} \), \( \text{bool} \)
- **TAFF**: Affirmation of the parameter set provided by the tuning procedure: \( \text{bool} \)
  - \( \text{TAFF} = \text{on} \)
ips  Meaning of the output signals p1,...,p6  long
  0 ...... Controller parameters
      p1 ... recommended control period TC
      p2 ... xi
      p3 ... om
      p4 ... taup
      p5 ... taum
      p6 ... tauf
  1 ...... Auxiliary parameters
      p1 ... htp2 - time of the peak in the second derivative of
              pv
      p2 ... hpeak2 - peak value in the second derivative of pv
      p3 ... d2 - peak to peak amplitude of t_d2pv
      p4 ... tgain

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 >= hilim_p or mv <= -hilim_m

isv  Number of the positive (+) or negative (-) sliding variable steps  long

ite  Error code  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 TS OR too high
              level of noise in the second derivative of pv (try to decrease
              the beta parameter)
      5 ...... Premature termination of the tuning procedure (TBRK)

p_i  Identified parameters with respect to ips, i = 1,...,6  double
CHAPTER 7. REG – FUNCTION BLOCKS FOR CONTROL

Parameters

- **ipwmc**: PWM cycle (in sampling periods of the block, $T_C/T_S$)  
  - Value: $100$  
  - Type: long
- **xi**: Relative damping of sliding zero dynamics  
  - Value: $5.0$  
  - Type: double
- **om**: Natural frequency $\omega$ of sliding zero dynamics  
  - Value: $0.0$  
  - Type: double
- **taup**: Time constant for adaptation of heating action amplitude [s]  
  - Value: $700.0$  
  - Type: double
- **taum**: Time constant for adaptation of cooling action amplitude [s]  
  - Value: $400.0$  
  - Type: double
- **beta**: Bandwidth parameter of the derivative filter  
  - Value: $0.01$  
  - Type: double
- **hilim_p**: Upper limit of the heating action amplitude  
  - Value: $0.0$  
  - Type: double
- **hilim_m**: Upper limit of the cooling action amplitude  
  - Value: $0.0$  
  - Type: double
- **u0_p**: Initial amplitude of the heating action  
  - Value: $1.0$  
  - Type: double
- **u0_m**: Initial amplitude of the cooling action  
  - Value: $1.0$  
  - Type: double
- **sp_dif**: Setpoint difference threshold for blocking of heating/cooling amplitudes reset  
  - Value: $10.0$  
  - Type: double
- **tauf**: Time constant of the filter for obtaining the equivalent manipulated variable  
  - Value: $400.0$  
  - Type: double
- **itm**: Tuning method  
  - Value: $1$  
  - Type: long
  - Description: Restricted to symmetrical processes
  - Description: Asymmetrical processes (not implemented yet)
- **ut_p**: Amplitude of heating for tuning experiment  
  - Value: $0.0$  
  - Type: double
- **ut_m**: Amplitude of cooling for tuning experiment  
  - Value: $0.0$  
  - Type: double
SWU – Switch unit

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

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uc</td>
<td>This input is copied to output y when all the binary inputs OR1, OR2, OR3 and OR4 are off</td>
</tr>
<tr>
<td>uo</td>
<td>This input is copied to output y when any of the binary inputs OR1, OR2, OR3, OR4 is on</td>
</tr>
<tr>
<td>OR1</td>
<td>First logical output of the block</td>
</tr>
<tr>
<td>OR2</td>
<td>Second logical output of the block</td>
</tr>
<tr>
<td>OR3</td>
<td>Third logical output of the block</td>
</tr>
<tr>
<td>OR4</td>
<td>Fourth logical output of the block</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Analog output of the block</td>
</tr>
</tbody>
</table>
TSE – Three-state element

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

\( u \)  \hspace{1cm} \text{Analog input of the block}  \hspace{1cm} \text{double}

Outputs

\( \text{UP} \)  \hspace{1cm} \text{The "up" signal}  \hspace{1cm} \text{bool}
\( \text{DN} \)  \hspace{1cm} \text{The "down" signal}  \hspace{1cm} \text{bool}

Parameters

\( \text{ep} \)  \hspace{1cm} \text{The input value } u > \text{ep} \text{ results in UP = on and DN = off}  \hspace{1cm} \textcircled{1.0}  \hspace{1cm} \text{double}
\( \text{en} \)  \hspace{1cm} \text{The input value } u < \text{en} \text{ results in UP = off and DN = off}  \hspace{1cm} \textcircled{-1.0}  \hspace{1cm} \text{double}
\( \text{epoff} \)  \hspace{1cm} \text{UP switch off value; if UP = on and } u < \text{epoff} \text{ then UP = off}  \hspace{1cm} \textcircled{0.5}  \hspace{1cm} \text{double}
\( \text{enoff} \)  \hspace{1cm} \text{DN switch off value; if DN = on and } u > \text{enoff} \text{ then DN = off}  \hspace{1cm} \textcircled{-0.5}  \hspace{1cm} \text{double}
Chapter 8

LOGIC – Logic control

Contents

- AND_ – Logical product of two signals .................................................. 232
- ANDQUAD, ANDOCT, ANDHEXD – Logical product of multiple signals .. 233
- ATMT – Finite-state automaton .............................................................. 234
- BDOCT, BDHEXD – Bitwise demultiplexers ............................................. 236
- BITOP – Bitwise operation ................................................................. 237
- BMOCT, BMHEXD – Bitwise multiplexers .............................................. 238
- COUNT – Controlled counter ................................................................. 239
- EATMT – Extended finite-state automaton ............................................. 241
- EDGE_ – Falling/rising edge detection in a binary signal ......................... 244
- INTSM – Integer number bit shift and mask .......................................... 245
- ISSW – Simple switch for integer signals ............................................... 246
- INTSM – Integer number bit shift and mask .......................................... 247
- ITOI – Transformation of integer and binary numbers ........................... 248
- NOT_ – Boolean complementation .......................................................... 249
- OR_ – Logical sum of two signals ......................................................... 250
- ORQUAD, OROCT, ORHEXD – Logical sum of multiple signals .......... 251
- RS – Reset-set flip-flop circuit ............................................................ 252
- SR – Set-reset flip-flop circuit ............................................................. 253
- TIMER_ – Multipurpose timer ............................................................... 254
AND_ – Logical product of two signals

Block Symbol

Function Description
The AND_ block computes the logical product of two input signals U1 and U2.
If you need to work with more input signals, use the ANDOCT block.

Inputs
- U1: First logical input of the block (bool)
- U2: Second logical input of the block (bool)

Outputs
- Y: Output signal, logical product (U1 ∧ U2) (bool)
- NY: Boolean complementation of Y (NY = ¬Y) (bool)
ANDQUAD, ANDOCT, ANDHEXD – Logical product of multiple signals

Block Symbols

Licence: STANDARD

Function Description

The ANDQUAD, ANDOCT and ANDHEXD blocks compute the logical product of up to sixteen input signals U1, U2, ..., U16. The signals listed in the n1 parameter are negated prior to computing the logical product.

For an empty n1 parameter a simple logical product \( Y = U_1 \land U_2 \land U_3 \land U_4 \land U_5 \land U_6 \land U_7 \land U_8 \) is computed. For e.g. \( n1=1,3..5 \), the logical function is \( Y = \neg U_1 \land U_2 \land \neg U_3 \land \neg U_4 \land \neg U_5 \land U_6 \land \ldots U_{16} \).

If you have less than 4/8/16 signals, use the n1 parameter to handle the unconnected inputs. If you have only two input signals, consider using the AND_ block.

Inputs

\( U_1..U_{16} \quad \text{Logical inputs of the block} \quad \text{bool} \)

Outputs

\( Y \quad \text{Result of the logical operation} \quad \text{bool} \)

\( NY \quad \text{Boolean complementation of } Y \quad \text{bool} \)

Parameter

\( n1 \quad \text{List of signals to negate. The format of the list is e.g. } 1,3..5,8. \quad \text{long} \)

Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.
ATMT – Finite-state automaton

Block Symbol

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

The transition conditions $C_k, k = 0, 1, ..., 15$ are activated by the binary inputs $C_0, C_1, ..., C_{15}$. 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:

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$C_1$</th>
<th>$FS_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_2$</td>
<td>$C_2$</td>
<td>$FS_2$</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S_n$</td>
<td>$C_n$</td>
<td>$FS_n$</td>
</tr>
</tbody>
</table>

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

$S_1 \quad C_1 \quad FS_1$

has the meaning

If (state $S_1$ 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 $S_0$. The $\text{SET}$ input allows manual transition from the current state to the $n\text{s}0$ state when rising edge occurs. The
The input overpowers the SET input. The HLD = on input freezes the automat activity, the automat stays in the current state regardless of the Ci input signals and the tstep timer is not incremented. The TOUT output indicates that the machine remains in the given state longer than expected. The time limits TOi for individual states are defined by the touts array. There is no time limit for the given state when TOi is set to zero. The TOUT output is set to off whenever the automat changes its state.

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

The development tools of the REXYGEN system include also the SFCEditor program. You can create SFC schemes graphically using this tool. Run this editor from REXYGEN Studio by clicking the Configure button in the parameter dialog of the ATMT block.

**Inputs**

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

**Outputs**

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

**Parameters**

<table>
<thead>
<tr>
<th>morestps</th>
<th>Allow multiple transitions in one cycle of the automat</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>off ...</td>
<td>Disabled</td>
<td></td>
</tr>
<tr>
<td>on ....</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>ntr</td>
<td>Number of state transition table rows</td>
<td>long</td>
</tr>
<tr>
<td>sfcname</td>
<td>Filename of block configurator data file (filename is generated by system if parameter is empty)</td>
<td>string</td>
</tr>
<tr>
<td>STT</td>
<td>State transition table (matrix)</td>
<td>byte</td>
</tr>
<tr>
<td>touts</td>
<td>Vector of timeouts TO0, TO1, ..., TO15 for the states S0, S1, ..., S15</td>
<td>double</td>
</tr>
</tbody>
</table>
BDOCT, BDHEXD – Bitwise demultiplexers

Block Symbols

Licence: STANDARD

Function Description
Both BDOCT and BDHEXD are bitwise demultiplexers for easy decomposition of the input signal to individual bits. The only difference is the number of outputs, the BDOCT block has 8 Boolean outputs while the BDHEXD block offers 16-bit decomposition. The output signals $Y_i$ correspond with the individual bits of the input signal $iu$, the $Y_0$ output is the least significant bit.

Input
$iu$ Input signal to be decomposed long

Outputs
$Y_0...Y_{15}$ Individual bits of the input signal bool

Parameter
$shift$ Bit shift of the input signal $\downarrow 0 \uparrow 31$ long
BITOP – Bitwise operation

Block Symbol 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
- $i_1$: First integer input of the block
- $i_2$: Second integer input of the block

Output
- $n$: Result of the bitwise operation $iop$

Parameter
- $iop$: Bitwise operation

<table>
<thead>
<tr>
<th>$iop$</th>
<th>Bitwise operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bitwise negation (Bit NOT)</td>
</tr>
<tr>
<td>2</td>
<td>Bitwise logical sum (Bit OR)</td>
</tr>
<tr>
<td>3</td>
<td>Bitwise logical product (Bit AND)</td>
</tr>
<tr>
<td>4</td>
<td>Bitwise logical exclusive sum (Bit XOR)</td>
</tr>
<tr>
<td>5</td>
<td>Shift of the $i_1$ signal by $i_2$ bits to the left (Shift Left)</td>
</tr>
<tr>
<td>6</td>
<td>Shift of the $i_1$ signal by $i_2$ bits to the right (Shift Right)</td>
</tr>
<tr>
<td>7</td>
<td>2's complement of the $i_1$ signal on 8 bits (2’s Complement - Byte)</td>
</tr>
<tr>
<td>8</td>
<td>2's complement of the $i_1$ signal on 16 bits (2’s Complement - Word)</td>
</tr>
<tr>
<td>9</td>
<td>2's complement of the $i_1$ signal on 32 bits (2’s Complement - Long)</td>
</tr>
</tbody>
</table>
**BMOCT, BMHEXD – Bitwise multiplexers**

**Block Symbols**

**Function Description**

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

**Inputs**

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

**Output**

$iy$  Composed output signal  
$long$

**Parameter**

$shift$  Bit shift of the output signal  
$\downarrow 0 \uparrow 31$  $long$
COUNT – Controlled counter

Block Symbol

Function Description

The COUNT block is designed for bidirectional pulse counting – more precisely, counting rising edges of the UP and DN input signals. When a rising edge occurs at the UP (DN) input, the cnt output is incremented (decremented) by 1. Simultaneous occurrence of rising edges at both inputs is indicated by the error output E set to on. The R1 input resets the counter to 0 and no addition or subtraction is performed unless the R1 input returns to off again. It is also possible to set the output cnt to the value n0 by the SETH input. Again, no addition or subtraction is performed unless the SETH input returns to off again. The R1 input has higher priority than the SETH input. The input HLD = on prevents both incrementing and decrementing. When the counter reaches the value cnt ≥ nmax, the Q output is set to on.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Block reset (R1 = on)</td>
<td>bool</td>
</tr>
<tr>
<td>n0</td>
<td>Value to set the counter to (using the SETH input)</td>
<td>long</td>
</tr>
<tr>
<td>SETH</td>
<td>Set the counter value to n0 (SETH = on)</td>
<td>bool</td>
</tr>
<tr>
<td>UP</td>
<td>Incrementing input signal</td>
<td>bool</td>
</tr>
<tr>
<td>DN</td>
<td>Decrementing input signal</td>
<td>bool</td>
</tr>
<tr>
<td>HLD</td>
<td>Counter freeze</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... Counter is running</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... Counter is locked</td>
<td></td>
</tr>
<tr>
<td>nmax</td>
<td>Counter target value</td>
<td>long</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnt</td>
<td>Total number of pulses</td>
<td>long</td>
</tr>
<tr>
<td>SGN</td>
<td>Sign of the cnt output</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... for cnt ≤ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... for cnt &gt; 0</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>Target value indicator</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... for cnt &lt; nmax</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... for cnt ≥ nmax</td>
<td></td>
</tr>
</tbody>
</table>
E: Indicator of simultaneous occurrence of rising edges at both inputs \texttt{UP} and \texttt{DN}
EATMT – Extended finite-state automaton

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 \( i \text{ MOD } 16 \) of the \( q(i \text{ DIV } 16) \) output is set to 1. The remaining bits of that output and the other outputs are zero. The bits are numbered from zero, least significant bit first. Note that the DIV and MOD operators denote integer division and remainder after integer division respectively. The current state is also indicated by the \( \text{ksa} \in \{0,1,\ldots,255\} \) 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 \( (k \text{ MOD } 16) \)-th bit of the input \( c(k \text{ DIV } 16) \) is equal to 1. The transition cannot happen otherwise.

The \text{BMHEXD} or \text{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 \text{BDHEXD} or \text{BDOCT} bitwise demultiplexers.

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

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

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

\[
S_1 \ C_1 \ FS_1
\]

has the meaning

\textbf{If} (\( S_1 \) is the current state AND transition condition \( C_1 \) is fulfilled) \\
\textbf{then} proceed to the following state \( FS_1 \).
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 SET input allows manual transition from the current state to the $ns_0$ state when rising edge occurs. The $R_1$ input overpowers the SET input. The $HLD = \text{on}$ input freezes the automat activity, the automat stays in the current state regardless of the $c_i$ input signals and the $tstep$ timer is not incremented. The $TOUT$ output indicates that the machine remains in the given state longer than expected. The time limits $TO_i$ for individual states are defined by the $touts$ array. There is no time limit for the given state when $TO_i$ is set to zero. The $TOUT$ 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 $TOUT$ output is used in transition conditions. In such a case it is strongly recommended to incorporate the $ksa$ output in the transition conditions as well.

The development tools of the REXYGEN system include also the SFCEditor program. You can create SFC schemes graphically using this tool. Run this editor from REXYGEN Studio by clicking the $\text{Configure}$ button in the parameter dialog of the $EATMT$ 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 SET input
- $ns_0$ This state is reached when rising edge occurs at the the SET input
- $\text{SET}$ The rising edge of this signal forces the transition to the $ns_0$ state
- $HLD$ The $HLD = \text{on}$ freezes the automat, no transitions occur regardless of the input signals, $tstep$ 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
- $ksa$ Integer code of the active state
- $tstep$ Time elapsed since the current state was reached; the timer is set to 0 whenever a state transition occurs
- $TOUT$ Flag indicating that the time limit for the current state was exceeded

**Parameters**

- $\text{morestps}$ Allow multiple transitions in one cycle of the automat
  - off ... Disabled
  - on .... Enabled
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntr</td>
<td>Number of state transition table rows</td>
<td>(0 \uparrow 1024 \circ 4) long</td>
</tr>
<tr>
<td>sfcname</td>
<td>Filename of block configurator data file (filename is generated by system if parameter is empty)</td>
<td>string</td>
</tr>
<tr>
<td>STT</td>
<td>State transition table (matrix)</td>
<td>([0 \ 0 \ 1; \ 1 \ 1 \ 2; \ 2 \ 2 \ 3; \ 3 \ 3 \ 0]) short</td>
</tr>
<tr>
<td>touts</td>
<td>Vector of timeouts (T_{00}, T_{01}, \ldots, T_{0255}) for the states (S_0, S_1, \ldots, S_{255})</td>
<td>double</td>
</tr>
</tbody>
</table>

\([1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \ 16]\)
EDGE_ – Falling/rising edge detection in a binary signal

Function Description

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

Input

U Logical input of the block bool

Output

Y Logical output of the block bool

Parameter

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 $\downarrow-9,22337E+18 \uparrow9,22337E+18$ large

Parameters

$\text{shift}$ Bit shift (negative=left, positive=right) $\downarrow-63 \uparrow63$ long

$\text{mask}$ Bit mask (applied after bit shift) $\downarrow0 \uparrow4294970000 \ominus4294967295$ large

Output

$n$ Resulting integer value large
ISSW – Simple switch for integer signals

Block Symbol

Licence: STANDARD

Function Description
The ISSW block is a simple switch for integer input signals i1 and i2 whose decision variable is the binary input SW. If SW is off, then the output n is equal to the i1 signal. If SW is on, then the output n is equal to the i2 signal.

Inputs

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>First integer input of the block</td>
<td>long</td>
</tr>
<tr>
<td>i2</td>
<td>Second integer input of the block</td>
<td>long</td>
</tr>
<tr>
<td>SW</td>
<td>Signal selector</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>off ... The i1 signal is selected</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on .... The i2 signal is selected</td>
<td></td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Integer output of the block</td>
<td>long</td>
</tr>
</tbody>
</table>
**INTSM – Integer number bit shift and mask**

**Block Symbol**

![INTSM](image)

**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  
  \([-9,22337E+18, 9,22337E+18]\) large

**Parameters**

- \(\text{shift}\)  
  Bit shift (negative=left, positive=right)  
  \([-63, 63]\) long

- \(\text{mask}\)  
  Bit mask (applied after bit shift)  
  \([0, 4294970000, \odot 4294967295]\) large

**Output**

- \(n\)  
  Resulting integer value  
  large
CHAPTER 8. LOGIC – LOGIC CONTROL

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), from the set \{0, 1, 2, \ldots, 15\} to the output number nk and its binary representation (Y3 Y2 Y1 Y0) 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>\ldots</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>nk</td>
<td>n0</td>
<td>n1</td>
<td>n2</td>
<td>\ldots</td>
<td>n15</td>
</tr>
</tbody>
</table>

where n0, \ldots, n15 are given by the mapping table target vector fktab.

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

Inputs

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

Outputs

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

Parameters

- BINF: Enable the binary selectors, \( \circ \)on bool
  - off ... Disabled (integer input k)
  - on ... Enabled (binary input signals U3...U0)

- fktab: Vector of mapping table target values, byte
  - \( \circ \)[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
**NOT** – Boolean complementation

**Block Symbol**

![NOT symbol](image)

**Licence:** STANDARD

**Function Description**

The **NOT** block negates the input signal.

**Input**

<table>
<thead>
<tr>
<th>U</th>
<th>Logical input of the block</th>
<th>bool</th>
</tr>
</thead>
</table>

**Output**

<table>
<thead>
<tr>
<th>Y</th>
<th>Logical output of the block ((Y = \neg U))</th>
<th>bool</th>
</tr>
</thead>
</table>
OR – Logical sum of two signals

Block Symbol Licence: STANDARD

Function Description
The OR block computes the logical sum of two input signals U1 and U2.
If you need to work with more input signals, use the OROCT block.

Inputs

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

Outputs

| Y   | Logical output of the block (U1 ∨ U2) | bool |
| NY  | Boolean complementation of Y (NY = ¬Y) | bool |
ORQUAD, OROCT, ORHEXD – Logical sum of multiple signals

Block Symbols

Function Description

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

For an empty n1 parameter a simple logical sum \( Y = U_1 \lor U_2 \lor U_3 \lor U_4 \lor U_5 \lor U_6 \lor U_7 \lor \ldots \lor U_{16} \) is computed. For e.g. n1=1,3..5, the logical function is \( Y = \neg U_1 \lor U_2 \lor \neg U_3 \lor \neg U_4 \lor \neg U_5 \lor U_6 \lor \ldots \lor U_{16} \).

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

Inputs

U1..U16 Logical inputs of the block

Outputs

Y Result of the logical operation
NY Boolean complementation of Y

Parameter

n1 List of signals to negate. The format of the list is e.g. 1,3..5,8. 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**

![RS 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.

![Inner Block Structure]

**Inputs**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Flip-flop set, sets the Q output to <strong>on</strong></td>
<td>bool</td>
</tr>
<tr>
<td>R1</td>
<td>Priority flip-flop reset, sets the Q output to <strong>off</strong>, overpowers the S input</td>
<td>bool</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>Flip-flop circuit state</td>
<td>bool</td>
</tr>
<tr>
<td>NQ</td>
<td>Boolean complementation of Q</td>
<td>bool</td>
</tr>
</tbody>
</table>
**SR – Set-reset flip-flop circuit**

**Block Symbol**

![SR Block Symbol](image)

**Licence:** STANDARD

**Function Description**

The SR block is a flip-flop circuit, which sets its output permanently to on as soon as the input signal S1 is on. The other input signal R resets the Q output to off, but only if the S1 input is off. The NQ output is simply the negation of the signal Q.

The block function is evident from the inner block structure depicted below.

![SR Block Diagram](image)

**Inputs**

<table>
<thead>
<tr>
<th>S1</th>
<th>Priority flip-flop set, sets the Q output to on, overpowers the R input</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Flip-flop reset, sets the Q output to off</td>
<td>bool</td>
</tr>
</tbody>
</table>

**Outputs**

| Q   | Flip-flop circuit state                                               | bool |
| NQ  | Boolean complementation of Q                                          | bool |
CHAPTER 8. LOGIC – LOGIC CONTROL

TIMER_ – Multipurpose timer

Block Symbol

Licence: STANDARD

Function Description

The TIMER_ block either generates an output pulse of the given width $pt$ (in seconds) or filters narrow pulses in the $U$ input signal whose width is less than $pt$ seconds. The operation mode is determined by the mode parameter.

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

The timer can be paused by the HLD input. The R1 input resets the timer. The reset signal overpowers the $U$ input.

Inputs

- $U$: Trigger of the timer
- HLD: Timer hold
- R1: Block reset ($R1 = \text{on}$)

Outputs

- $Q$: Timer output
- et: Elapsed time [s]
- rt: Remaining time [s]
### Parameters

**mode**  Timer mode

1. **Pulse** – an output pulse of the length $pt$ is generated upon rising edge at the $U$ input. All input pulses during the generation of the output pulse are ignored.

2. **Delayed ON** – the input signal $U$ is copied to the $Q$ output, but the start of the pulse is delayed by $pt$ seconds. Any pulse shorter than $pt$ does not pass through the block.

3. **Delayed OFF** – the input signal $U$ is copied to the $Q$ output, but the end of the pulse is delayed by $pt$ seconds. If the break between two pulses is shorter than $pt$, the output remains on for the whole time.

4. **Delayed change** – the $Q$ output is set to the value of the $U$ input no sooner than the input remains unchanged for $pt$ seconds

**pt**  Timer interval [s]

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
</tr>
</tbody>
</table>
Chapter 9

TIME – Blocks for handling time

Contents

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE_</td>
<td>Current date</td>
<td>258</td>
</tr>
<tr>
<td>DATETIME</td>
<td>Get, set and convert time</td>
<td>259</td>
</tr>
<tr>
<td>TIME</td>
<td>Current time</td>
<td>261</td>
</tr>
<tr>
<td>WSCH</td>
<td>Weekly schedule</td>
<td>262</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>Variable</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>Variable</th>
<th>Description</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>tz</td>
<td>Timezone</td>
<td>1 Local time, 2 UTC</td>
</tr>
</tbody>
</table>
DATETIME – Get, set and convert time

Block Symbol Licence: STANDARD

Function Description

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

It allows synchronization of the operating system clock and the clock of the REXYGEN system. When the executive of the REXYGEN system is initialized, both clocks are the same. But during long-term operation the clocks may 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 REXYGEN system according to the block inputs and parameters.

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

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

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

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

For simple date/time reading use the DATE_ and TIME function blocks.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uyear</td>
<td>Input for setting year</td>
<td>long</td>
</tr>
<tr>
<td>umonth</td>
<td>Input for setting month</td>
<td>long</td>
</tr>
<tr>
<td>uday</td>
<td>Input for setting day</td>
<td>long</td>
</tr>
<tr>
<td>uhour</td>
<td>Input for setting hours</td>
<td>long</td>
</tr>
<tr>
<td>umin</td>
<td>Input for setting minutes</td>
<td>long</td>
</tr>
</tbody>
</table>
CHAPTER 9. TIME – BLOCKS FOR HANDLING TIME

usec Input for setting seconds long
unsec Input for setting nanoseconds ↓-9,22E+18 ↑9,22E+18 large
SET Trigger for setting time bool
GET Trigger for getting time bool

Outputs

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

Parameters

isetmode Source for setting time ⊙1 long
1 ...... OS clock
2 ...... Block inputs
3 ...... The unsec input
4 ...... The usec input
5 ...... The unsec input, relative

igetmode Source for getting or converting time ⊙6 long
1 ...... OS clock
2 ...... Block inputs
3 ...... The unsec input
4 ...... The usec input
5 ...... The uday input
6 ...... REXYGEN clock

settoll Tolerance for setting the REXYGEN clock [s] ⊙1.0 double
setper Period for setting time [s] (0=not periodic) double
getper Period for getting time [s] (0=not periodic) ⊙0.001 double
FDOW First day of week is Sunday bool
off .... Week starts on Monday
on .... Week starts on Sunday
tz Timezone ⊙1 long
1 ...... Local time
2 ...... UTC
TIME – Current time

Block Symbol

Function Description

The outputs of the TIME function block correspond with the actual time of the operating system. Use the DATETIME block for advanced operations with date and time.

Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>hour</td>
<td>Hours</td>
<td>long</td>
</tr>
<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>Symbol</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></td>
<td>2</td>
</tr>
</tbody>
</table>

Local time

UTC
WSCH – Weekly schedule

Block Symbol

Function Description

The WSCH function block is a weekly scheduler for e.g. heating (day, night, eco), ventilation (high, low, off), lighting, irrigation etc. Its outputs can be used for switching individual appliances on/off or adjusting the intensity or power of the connected devices.

During regular weekly schedule the outputs iy and y reflect the values from the wst table. This table contains triplets day-hour-value. E.g. the notation [2 6.5 21.5] states that on Tuesday, at 6:30 in the morning (24-hour format), the output y will be set to 21.5. The output iy will be set to 22 (rounding to nearest integer). The individual triplets are separated by semicolons.

The days in a week are numbered from 1 (Monday) to 7 (Sunday). Higher values can be used for special daily schedules, which can be forced using the fsch input or the specdays table. The active daily program is indicated by the isch output.

Alternatively it is possible to temporarily force a specific output value using the val input and a rising edge at the SET input (off→on). When a rising edge occurs at the SET input, the val input is copied to the y output and the isch output is set to 0. The forced value remains set until:

- the next interval as defined by the wst table, or
- another rising edge occurs at the SET input, or
- a different daily schedule is forced using the fsch input.

The list of special days (specdays) can be used for forcing a special daily schedule at given dates. E.g. you can force a Sunday daily schedule on holidays, Christmas, New Year, etc. The date is entered in the YYYYMMDD format. The notation [20160328 7] thus means that on March 28th, 2016, the Sunday daily schedule should be used. Individual pairs are separated by semicolons.

The trem and ynext outputs can be used for triggering specific actions in advance, before the y and iy are changed.

The iy output is meant for direct connection to function blocks with Boolean inputs (the conversion from type long to type bool is done automatically).

The nmax parameter defines memory allocation for the wst and specdays arrays. For nmax = 100 the wst list can contain up to 100 triplets day-hour-value. In typical applications there is no need to modify the nmax parameter.
Inputs

**SET**
Trigger for setting the y and iy outputs

**val**
Temporary value to set the y and iy outputs to

**fsch**
Forced schedule

- 0 ...... standard weekly schedule
- 1 ...... Monday
- 2 ...... Tuesday
- .... ... 
- 7 ...... Sunday
- 8 and above additional daily programs as defined by the wst table

Outputs

**iy**
Integer output value

**y**
Output value

**isch**
Daily schedule identifier

**trem**
Time remaining in the current section (in seconds)

**ynext**
Output in the next section

Parameters

**tz**
Timezone

- 1 ...... Local time
- 2 ...... UTC

**nmax**
Allocated size of arrays

- 10 \[1000000 \times 100 \]

**wst**
Weekly schedule table (list of triplets day-hour-value)

- 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

**specdays**
List of special days (list of pairs date-daily program)

- 20150406 1; 20151224 1; 20151225 1; 20151226 1; 20160101 1; 20160328 1; 20170417 1; 20180402 1; 20190422 1; 20200413 1
Chapter 10

ARC – Data archiving

Contents

10.1 Functionality of the archiving subsystem .................. 266
10.2 Generating alarms and events ................................. 267
   ALB, ALBI – Alarms for Boolean value ...................... 267
   ALN, ALNI – Alarms for numerical value .................... 269
10.3 Trends recording .............................................. 271
   ACD – Archive compression using Delta criterion .......... 271
   TRND – Real-time trend recording ............................ 273
   TRNDV – Real-time trend recording with vector input ...... 276
10.4 Archive management ........................................... 277
   AFLUSH – Forced archive flushing ........................... 277

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

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

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

- Blocks for generating alarms and events
- Blocks for recording trends
- Blocks for handling archives
10.1 Functionality of the archiving subsystem

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

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

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

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

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

ALB, ALBI – Alarms for Boolean value

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 REXYGEN system. The range 1 ≤ lvl ≤ 127 is reserved for alarms. All starts, ends and acknowledgements of the alarms are stored in the archive. On the contrary, the range 128 ≤ lvl ≤ 255 indicates events. Only the start (the time instant) of the event is stored in the archive.

Inputs

<table>
<thead>
<tr>
<th>U</th>
<th>Logical input of the block whose changes are watched</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>men</td>
<td>Enable alarms</td>
<td>long</td>
</tr>
<tr>
<td>0</td>
<td>All alarms disabled</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Low-alarm enabled (LA) (falling edge in the input signal U)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>High-alarm enabled (HA) (rising edge in the input signal U)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>All alarms enabled</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iACK</th>
<th>Acknowledge alarm</th>
<th>byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Low-alarm acknowledge</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>High-alarm acknowledge</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Both alarms acknowledge</td>
<td></td>
</tr>
</tbody>
</table>

Alarm is acknowledged on rising edge

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>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>HA</td>
<td>High-alarm indicator</td>
<td>bool</td>
</tr>
<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>
<tr>
<td>arc</td>
<td>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.</td>
<td>word</td>
</tr>
<tr>
<td>id</td>
<td>Identification code of the alarm in the archive. This identifier must be unique in the whole target device with the REXYGEN control system (i.e. in all archiving blocks). Disabled for id = 0.</td>
<td>word</td>
</tr>
<tr>
<td>lvl</td>
<td>The level of the alarms (HA and LA) which differentiates alarms from events and defines the severity of the alarm/event</td>
<td>byte</td>
</tr>
<tr>
<td>Desc</td>
<td>Extended description of the alarm which is displayed by the diagnostic tools of the REXYGEN system</td>
<td>string</td>
</tr>
</tbody>
</table>

- **HA**: High-alarm indicator
- **LA**: Low-alarm indicator
- **NACK**: Alarm-not-acknowledged indicator
- **arc**: List of archives to store the events
- **id**: Identification code of the alarm in the archive
- **lvl**: Level of the alarms
- **Desc**: Extended description of the alarm
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 $i\text{ack}$ 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 REXYGEN system. The range $1 \leq lvl \leq 127$ is reserved for alarms. All starts, ends and acknowledge-ments 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

<table>
<thead>
<tr>
<th>$u$</th>
<th>Analog input of the block which is checked to remain within the given limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$hys$</td>
<td>Alarm hysteresis for switching the alarm off $↓0.0 ↑10000000000.0$</td>
</tr>
<tr>
<td>$hh$</td>
<td>The second high-alarm limit, must be greater than $h$</td>
</tr>
<tr>
<td>$h$</td>
<td>High-alarm limit, must be greater than $l$</td>
</tr>
<tr>
<td>$l$</td>
<td>Low-alarm limit, must be greater than $ll$</td>
</tr>
<tr>
<td>$ll$</td>
<td>The second low-alarm limit</td>
</tr>
<tr>
<td>$i\text{ack}$</td>
<td>Alarm is acknowledged on rising edge of the individual bits of this input/parameter. E.g. value 15 acknowledges all alarms. byte</td>
</tr>
<tr>
<td>1</td>
<td>Second low-alarm acknowledge</td>
</tr>
<tr>
<td>2</td>
<td>Low-alarm acknowledge</td>
</tr>
<tr>
<td>4</td>
<td>High-alarm acknowledge</td>
</tr>
<tr>
<td>8</td>
<td>Second high-alarm acknowledge</td>
</tr>
</tbody>
</table>

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

E
Error flag
off . . . No error
on . . . An error occurred, alarm limits disordered

bool

HHA
The second high-alarm indicator

bool

HA
High-alarm indicator

bool

LA
Low-alarm indicator

bool

LLA
The second low-alarm indicator

bool

NACK
Alarm-not-acknowledged indicator

bool

Parameters

acls
Alarm class (data type to store)

⊙8 byte

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.

word

id
Identification code of the alarm in the archive. This identifier must be unique in the whole target device with the REXYGEN control system (i.e. in all archiving blocks). Disabled for id = 0.

word

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

byte

1

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

byte

1

1

Desc
Extended description of the alarm which is displayed by the diagnostic tools of the REXYGEN system

string

⊙Alarm Description
10.3 Trends recording

**ACD – Archive compression using Delta criterion**

**Block Symbol**

![Diagram]

**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 \( TR = \text{off} \) the condition \( |u - u_0| > \delta \) 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 \( TR = \text{on} \) the input signal values are compared to a signal with compensated trend. The condition for storing the signal is the same as in the previous case.

The following figure shows the archiving process for both cases: a) \( TR = \text{off} \), b) \( TR = \text{on} \). The stored samples are marked by the symbol ×.

**Inputs**

- **u**: Signal to compress and store \hspace{1cm} **double**
- **\( \delta \)**: Threshold for storing the signal \hspace{1cm} \( \downarrow 0.0 \uparrow 10000000000.0 \) **double**
**Outputs**

- **y**: The last value stored in the archive
  - Type: `double`

- **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 REXYGEN control system (i.e. in all archiving blocks). Disabled for `id = 0`.

- **tmin**: The shortest interval between two samples of the input signal stored in the archive [s]
  - Type: `double`
  - Value: `0.001` to `1000000.0`

- **tmax**: The longest interval between two samples of the input signal stored in the archive [s]
  - Type: `double`
  - Value: `1.0` to `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 REXYGEN system
  - Type: `string`
10.3. TRENDS RECORDING

TRND – Real-time trend recording

Block Symbol

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

It can happen that the processed input value exceeds the representable limits when using different type of buffer than Double. In such a case the highest (lowest) representable number of the corresponding type is stored in the buffer and an error is binary encoded to the iE output according to the following table (the unused bits are omitted):

<table>
<thead>
<tr>
<th>Error</th>
<th>Range underflow</th>
<th>Range overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>u4</td>
<td>u3</td>
</tr>
<tr>
<td>Bit number</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>Bit weight</td>
<td>2048</td>
<td>1024</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 REXGEN Diagnostics 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 to the last values stored in the trend buffers `double`

`iE` Error code, see the table above `long`

**Parameters**

`n` Number of signals to process and store in the trend buffers `long`

`l` Number of samples reserved in memory for each trend buffer `long`

`btype` Type of all `n` trend buffers `long`

1. ... Bool 2. ... Byte 3. ... Short 4. ... Long 5. ... Word 6. ... DWord 7. ... Float 8. ... Double

`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 · T_S` unless `RUN = off`, where `T_S` is the block execution period in seconds. `long`

`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 · T_S`, where `T_S` is the block execution period in seconds. `long`
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 REXYGEN system (i.e. in all archiving blocks). Disabled for id = 0.

Title  Title of the trend to be displayed in the diagnostic tools of the REXYGEN system, e.g. in the REXYGEN Diagnostics program

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 – REXYGEN internal technological time which is incremented by nominal period each base tick. For long running trends where you are interested mostly in absolute time shared with operating system (and possibly synchronized by NTP), choose RTC. Other values are intended for debug or special purposes.

1 . . . . CORETIMER – technological time – at current tick
2 . . . . CORETIMER-PRECISE – technological time – at block execution
3 . . . . RTC – real time clock (wallclock) from operating system – at current tick
4 . . . . RTC-PRECISE – real time clock (wallclock) from operating system – at block execution
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 l samples gets full, the oldest samples are overwritten. Data can be stored once in pfac executions of the block (decimation). The other decimation factor afac can be used for storing data in archives.

The type of trend buffers can be specified in order to conserve memory of the target device. The memory requirements of the trend buffers are defined by the formula $s \cdot n \cdot l$, 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 $l = 1000$, $8 \cdot 4 \cdot 1000 = 32000$ bytes are required. In case the input signals come from 16-bit A/D converter the Word type can be used and memory requirements drop to one quarter. Memory requirements and allowed ranges of individual types are summarized in table 1.1 on page 16 of this reference guide.

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

Inputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uVec</td>
<td>Vector signal to record</td>
</tr>
<tr>
<td>HLD</td>
<td>Input for freezing the cyclic buffers, no data is appended when HLD = bool on</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iE</td>
<td>Error code</td>
</tr>
</tbody>
</table>

i ....... REXYGEN general error
10.4 Archive management

AFLUSH – Forced archive flushing

Block Symbol Licence: STANDARD

Function Description

The AFLUSH block is intended for immediate storing of archive data to permanent memory (hard drive, flash disk, etc.). It is useful when power loss can be anticipated, e.g., emergency shutdown of the system following some failure. It forces the archive subsystem to write all archive data to avoid data loss. The write operation is initiated by a rising edge (off→on) at the FLUSH input regardless of the period parameter of the ARC block.
CHAPTER 10. ARC – DATA ARCHIVING

Input

FLUSH  Force archive flushing  bool

Parameter

arc  List of archives to store the events. The format of the list is e.g. 1,3..5,8. The event will be stored in all listed archives (see the 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>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNS – String constant</td>
<td>280</td>
</tr>
<tr>
<td>CONCAT – Concat string by pattern</td>
<td>281</td>
</tr>
<tr>
<td>FIND – Find a Substring</td>
<td>282</td>
</tr>
<tr>
<td>ITOS – Integer number to string conversion</td>
<td>283</td>
</tr>
<tr>
<td>LEN – String length</td>
<td>284</td>
</tr>
<tr>
<td>MID – Substring Extraction</td>
<td>285</td>
</tr>
<tr>
<td>PJROCT – Parse JSON string (real output)</td>
<td>286</td>
</tr>
<tr>
<td>PJSOCT – Parse JSON string (string output)</td>
<td>287</td>
</tr>
<tr>
<td>REGEXP – Regular expression parser</td>
<td>288</td>
</tr>
<tr>
<td>REPLACE – Replace substring</td>
<td>291</td>
</tr>
<tr>
<td>RTOS – Real Number to String Conversion</td>
<td>292</td>
</tr>
<tr>
<td>SELSOCT – String selector</td>
<td>293</td>
</tr>
<tr>
<td>STOR – String to real number conversion</td>
<td>294</td>
</tr>
</tbody>
</table>
CNS – String constant

Block Symbol

Function Description

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

Parameters

- **scv**: String (constant) value
- **nmax**: Allocated size of string [bytes]

Output

- **sy**: String output value
**CONCAT – Concat string by pattern**

**Block Symbol**

![Block Symbol]

**Licence:** STANDARD

**Function Description**

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

**Inputs**

| su1..8 | String input value | string |

**Parameters**

<table>
<thead>
<tr>
<th>ptrn</th>
<th>Concatenation pattern</th>
<th>⊙%1%2%3%4 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**

| sy | String output value | string |


FIND – Find a Substring

Block Symbol

License: 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] \( \downarrow 0 \uparrow 65520 \) long

Output

- *pos* Position of substring long
**ITOS – Integer number to string conversion**

**Block Symbol**

![ITOS Block Symbol](image)

**Licence:** STANDARD

**Function Description**

The **ITOS** block is used for converting an integer into text. The **len** parameter specifies the minimum length of the output string. If the number has a smaller number of digits, zeroes or spaces will be added according to the **mode** parameter. The **radix** parameter specifies the numerical system in which the conversion is to be performed. The output string does not contain any identification of the numerical system used (e.g. the 0x prefix for the hexadecimal system).

**Input**

- **n** Integer input of the block
- **long**

**Output**

- **sy** String output value
- **string**

**Parameters**

- **len** Minimum length of output string
  - 0 - 30 **long**
- **mode** Output string format
  - 1 - Align right, fill with spaces
  - 2 - Align right, fill with zeroes
  - 3 - Align left, fill with spaces
- **radix** Radix
  - 2 - Binary
  - 8 - Octal
  - 10 - Decimal
  - 16 - Hexadecimal
  - 0 - 10 **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

\(su\) String input value \(\text{string}\)

Parameter

\(nmax\) Allocated size of string [bytes] \(\downarrow0 \uparrow65520\) \(\text{long}\)

Output

\(len\) Length of input string \(\text{long}\)
MID – Substring Extraction

Block Symbol

Function Description
The MID block extracts a substring sy from su. The parameters 1 and p specify position and length of the string being extracted in UTF-8 characters. The parameter p is one-based.

Inputs

- **su**: String input value
- **l**: Length of output string
- **p**: Position of output string (one-based)

Parameter

- **nmax**: Allocated size of string [bytes]

Output

- **sy**: String output value
PJROCT – Parse JSON string (real output)

Block Symbol

Function Description

Parses input JSON string \texttt{jtxt} according to specified \texttt{name*} parameters when the input \texttt{RUN} is on. Output signals are \texttt{real} type.

Inputs

\begin{itemize}
  \item \texttt{jtxt} \quad JSON formatted string \quad \textit{string}
  \item \texttt{RUN} \quad Enable execution \quad \textit{bool}
\end{itemize}

Parameters

\begin{itemize}
  \item \texttt{name1..8} \quad Name of JSON object \quad \textit{string}
  \item \texttt{nmax} \quad Allocated size of string [bytes] \quad \downarrow 0 \uparrow 65520 \quad \textit{long}
  \item \texttt{yerr} \quad Substitute value for an error case \quad \textit{double}
\end{itemize}

Outputs

\begin{itemize}
  \item \texttt{y1..8} \quad Block output signal \quad \textit{double}
  \item \texttt{iE} \quad Error code \quad \textit{error}
\end{itemize}
PJSOCT – Parse JSON string (string output)

Block Symbol

Function Description

Parses input JSON string jtxt according to specified name* parameters when the input RUN is on. Output signals are string type.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>jtxt</td>
<td>JSON formatted string</td>
<td>string</td>
</tr>
<tr>
<td>RUN</td>
<td>Enable execution</td>
<td>bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>name1..8</td>
<td>Name of JSON object</td>
<td>string</td>
</tr>
<tr>
<td>nmax</td>
<td>Allocated size of string [bytes]</td>
<td>↓0 ↑65520 long</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sy1..8</td>
<td>String output value</td>
<td>string</td>
</tr>
<tr>
<td>iE</td>
<td>Error code</td>
<td>error</td>
</tr>
</tbody>
</table>
CHAPTER 11. STRING – BLOCKS FOR STRING OPERATIONS

REGEXP – Regular expression parser

Block Symbol

Licence: ADVANCED

Function Description

This block implements a subset of Perl or C# or Unix command grep regular expression syntax.

Supported syntax is:

- `(?)` ... Must be at the beginning of the regex. Makes match case-insensitive
- `^` ... Match beginning of a buffer
- `$` ... Match end of a buffer
- `()` ... Grouping and substring capturing
- `\s` ... Match whitespace
- `\S` ... Match non-whitespace
- `\d` ... Match decimal digit
- `\n` ... Match new line character
- `\r` ... Match line feed character
- `\f` ... Match form feed character
- `\v` ... Match vertical tab character
- `\t` ... Match horizontal tab character
- `\b` ... Match backspace character
- `+` ... Match one or more times (greedy)
- `+?` ... Match one or more times (non-greedy)
- `*` ... Match zero or more times (greedy)
• *? . . . Match zero or more times (non-greedy)
• ? . . . Match zero or once (non-greedy)
• x|y . . . Match x or y (alternation operator)
• \meta . . . Match one of the meta characters: ^$()[ ]*+?|\n
Examples

• [0-9]+ . . . Find first integer in input string (and put it into cap output)
• [-+]?[0-9]*\.[0-9]+([-+]?[0-9]+)? . . . Find first real number in input string (and put it into cap output)
• ^\s*(.*?)\s*$ . . . Put trimmed input string into cap1 output
• num\s*:\s*([0-9]*\.[0-9]*) . . . Expect input string in JSON format; find integer parameter num, and put its value into cap1

Inputs

text String to parse string
RUN Enable execution bool

Parameters

expr Regular expression pattern string
nmax Allocated size of string ↓0 ↑65534 long
bufmax Parser internal buffer size (0 = autodetect) ↓0 ↑10000000 long

Outputs

MATCH Pattern match flag bool
cap Whole matching string string
cap1 Captured string (string matched to 1st bracket) string
cap2 Captured string (string matched to 2nd bracket) string
cap3 Captured string (string matched to 3rd bracket) string
cap4 Captured string (string matched to 4th bracket) string
### CHAPTER 11. STRING – BLOCKS FOR STRING OPERATIONS

<table>
<thead>
<tr>
<th>cap5</th>
<th>Captured string (string matched to 5th bracket)</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>cap6</td>
<td>Captured string (string matched to 6th bracket)</td>
<td>string</td>
</tr>
<tr>
<td>cap7</td>
<td>Captured string (string matched to 7th bracket)</td>
<td>string</td>
</tr>
<tr>
<td>cap8</td>
<td>Captured string (string matched to 8th bracket)</td>
<td>string</td>
</tr>
</tbody>
</table>
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>▽0 ▲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>
CHAPTER 11. STRING – BLOCKS FOR STRING OPERATIONS

RTOS – Real Number to String Conversion

Block Symbol

Function Description

TheRTOSconverts a real number inuinto a string value in su. Precision and format are specified by theprecandmodeparame ters.

Input

u Analog input of the block double

Output

sy String output value string

Parameters

| prec | Precision (number of digits) | \(0 \leq 20\) long |
| mode | Output string format          | 0 | long |
|------|-----------------------------|---------------------|
| 1    | .... Best fit               |                     |
| 2    | .... Normal                 |                     |
| 3    | .... Exponential            |                     |
**SELSOCT – String selector**

**Function Description**

The SELSOCT block selects one of the input strings and copy it to the output string sy. The selection of the active signal $u_0 \ldots u_{15}$ 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  
- iSW      Active signal selector  
- SW1..3   Binary signal selector

**Parameters**

- BINF   Enable the binary selectors   
- nmax   Allocated size of string [bytes]  

**Output**

- sy      The selected input signal
STOR – String to real number conversion

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  string

Parameter
yerr  Substitute value for an error case  double

Outputs
y  Analog output of the block  double
E  Error indicator  bool
Chapter 12
PARAM – Blocks for parameter handling

Contents

<table>
<thead>
<tr>
<th>Block Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETPA – Block for remote array parameter acquirement</td>
<td>296</td>
</tr>
<tr>
<td>GETPR, GETPI, GETPB – Blocks for remote parameter acquirement</td>
<td>298</td>
</tr>
<tr>
<td>GETPS – * Block for remote string parameter acquirement</td>
<td>300</td>
</tr>
<tr>
<td>PARA – Block with input-defined array parameter</td>
<td>301</td>
</tr>
<tr>
<td>PARR, PARI, PARB – Blocks with input-defined parameter</td>
<td>302</td>
</tr>
<tr>
<td>PARS – * Block with input-defined string parameter</td>
<td>304</td>
</tr>
<tr>
<td>SETPA – Block for remote array parameter setting</td>
<td>305</td>
</tr>
<tr>
<td>SETPR, SETPI, SETPB – Blocks for remote parameter setting</td>
<td>307</td>
</tr>
<tr>
<td>SETPS – * Block for remote string parameter setting</td>
<td>309</td>
</tr>
<tr>
<td>SGSLP – Set, get, save and load parameters</td>
<td>310</td>
</tr>
<tr>
<td>SILO – Save input value, load output value</td>
<td>314</td>
</tr>
<tr>
<td>SILOS – Save input string, load output string</td>
<td>316</td>
</tr>
</tbody>
</table>
GETPA – Block for remote array parameter acquisition

Block Symbol

Licence: STANDARD

Function Description

The GETPA block is used for acquiring the array parameters of other blocks in the model remotely. The block operates in two modes, which are switched by the GETF parameter. For GETF = off the output arrRef is set to the value of the remote parameter at the start and every time when the remote parameter changes. If the GETF parameter is set to on, then the block works in single-shot read mode. In that case the remote parameter is read only when rising edge (off→on) occurs at the GET input.

The name of the remote parameter is determined by the string parameter sc (string connection), which has the form <block_path:parameter_name>. The path to the block whose parameter should be read can contain hierarchic levels separated by dots followed by the block name. The path can be either relative or absolute:

- Relative – starts at the level where the GETPA block is located. The string has to be prefixed with '.' in this case. Examples of relative paths: "CNDR:yp", "Lights.ATMT:touts".
- Absolute – complete sequence of hierarchic levels down to the block. For referring to blocks located in the driver task (see the IOTASK block for details on configuration) the '&' followed by the driver’s name is used at the beginning of the absolute path. Examples of absolute paths: "task1.inputs.ATMT:touts", "EfaDrv.measurements.CNDR:yp".

The order and names of individual hierarchic levels are displayed in a tree structure in the REXYGEN Diagnostics program.

Input

GET

Input for initiating one-shot parameter read

bool

Outputs

arrRef

Array reference

reference

E

Error flag

bool
## 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>GETF</td>
<td>Get parameter only when forced to</td>
<td>bool</td>
</tr>
<tr>
<td>off</td>
<td>Remote parameter is continuously read</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>One-shot mode, the remote parameter is read only when forced to by the GET input (rising edge)</td>
<td></td>
</tr>
<tr>
<td>nmax</td>
<td>Maximum size of array</td>
<td>256 long</td>
</tr>
</tbody>
</table>
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 \( \text{GETF} = \text{off} \) 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 \( \text{on} \), then the blocks work in single-shot read mode. In that case the remote parameter is read only when rising edge (\( \text{off} \rightarrow \text{on} \)) occurs at the \( \text{GET} \) input.

The name of the remote parameter is determined by the string parameter \( \text{sc} \) (string connection), which has the form \(<\text{block_path:parameter_name}>\). It is also possible to access individual items of array-type parameters (e.g. the \( \text{tout} \) parameter of the ATMT block). This can be achieved using the square brackets and item number, e.g. \( .\text{ATMT:touts}[2] \). The items are numbered from zero, thus the string connection stated above refers to the third element of the array.

The path to the block whose parameter should be read can contain hierarchic levels separated by dots followed by the block name. The path can be either relative or absolute:

- Relative – starts at the level where the GETPR block (or GETPI, GETPB) is located. The string has to be prefixed with ‘.’ in this case. Examples of relative paths: "\.GAIN:k", ".Motor1.Position:ycn".

- Absolute – complete sequence of hierarchic levels down to the block. For referring to blocks located in the driver task (see the IOTASK block for details on configuration) the ‘&’ followed by the driver’s name is used at the beginning of the absolute path. Examples of absolute paths: "task1.inputs.lin1:u2", "&EfaDrv.measurements.DER1:n".

The order and names of individual hierarchic levels are displayed in a tree structure in the REXYGEN Diagnostics program.

Input

\( \text{GET} \) Input for initiating one-shot parameter read (\( \text{off} \rightarrow \text{on} \) ) 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>
</tbody>
</table>

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>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>
</tbody>
</table>

- off ... Remote parameter is continuously read
- on .... One-shot mode, the remote parameter is read only when forced to by the GET input (rising edge)
**GETPS — * Block for remote string parameter acquirement**

**Block Symbol**

![Block Symbol](GETPS)

**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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Input for initiating one-shot parameter read</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>sc</td>
<td>String connection to the parameter</td>
<td>string</td>
</tr>
<tr>
<td>GETF</td>
<td>Get parameter only when forced to</td>
<td>bool</td>
</tr>
<tr>
<td>off</td>
<td>Remote parameter is continuously read</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>One-shot mode</td>
<td></td>
</tr>
<tr>
<td>nmax</td>
<td>Allocated size of string</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>sy</td>
<td>Parameter value</td>
<td>string</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</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>
PARA – Block with input-defined array parameter

Block Symbol

Licence: STANDARD

Function Description

The PARA block allows, additionally to the standard way of parameter setting, changing one of its parameters by the input signal. The input-parameter pair is `uRef` and `apar`.

The Boolean input `LOC` (LOCal) determines whether the value of the `apar` parameter is read from the input `uRef` or is input-independent (LOC = on). In the local mode LOC = on the parameter apar contains the last value of input `uRef` entering the block right before LOC was set to on.

The output value is equivalent to the value of the parameter (yRef = apar).

Inputs

- **uRef**: Array reference
- **LOC**: Activation of local mode
  - `off` . . . The parameter follows the input
  - `on` . . . Local mode active

Output

- **yRef**: Array reference

Parameters

- **SETS**: Set array size flag. Use this flag to adjust the size of array when setting the parameter.
- **apar**: Internal value of the parameter
  - $[0.0 1.0 2.0 3.0 4.0 5.0]$
PARR, PARI, PARB – Blocks with input-defined parameter

Block Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>Activation of local mode</td>
</tr>
<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>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>

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. Afterwards it is possible to modify the value manually.

The output value is equivalent to the value of the parameter y = par, (or k = ipar, Y = PAR). The output of the PARR and PARI blocks can be additionally constrained by the saturation limits ⟨lolim; hilim⟩. The saturation is active only when SATF = on.

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

Inputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</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>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</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>
<tr>
<td>SATF</td>
<td>Activation of the saturation limits for the PARR and PARI blocks</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>off ... Signal not limited</td>
</tr>
<tr>
<td></td>
<td>on .... Saturation limits active</td>
</tr>
<tr>
<td>hilim</td>
<td>Upper limit of the output signal (the PARR and PARI blocks)</td>
</tr>
<tr>
<td>lolim</td>
<td>Lower limit of the output signal (the PARR and PARI blocks)</td>
</tr>
</tbody>
</table>
PARS – * Block with input-defined string parameter

Block Symbol

License: STANDARD

Function Description

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

Inputs

<table>
<thead>
<tr>
<th>sp</th>
<th>Parameter value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>Activation of local mode</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>spar</th>
<th>Internal value of the parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmax</td>
<td>Allocated size of string</td>
</tr>
</tbody>
</table>

Output

| sy  | String output of the block    |

string
SETPA – Block for remote array parameter setting

Function Description

The SETPA block is used for setting the array parameters of other blocks in the model remotely. The block operates in two modes, which are switched by the SETF parameter. For SETF = off the remote parameter cs is set to the value of the input vector signal arrRef at the start and every time when the input signal changes. If the SETF parameter is set to on, then the block works in one-shot write mode. In that case the remote parameter is set only when rising edge (off→on) occurs at the SET input.

The name of the remote parameter is determined by the string parameter sc (string connection), which has the form <block_path:parameter_name>. The path to the block whose parameter should be read can contain hierarchic levels separated by dots followed by the block name. The path can be either relative or absolute:

- Relative – starts at the level where the GETPA block is located. The string has to be prefixed with './' in this case. Examples of relative paths: ".CNDR:yp", ".Lights.ATMT:touts".

- Absolute – complete sequence of hierarchic levels down to the block. For referring to blocks located in the driver task (see the IOTASK block for details on configuration) the '& followed by the driver’s name is used at the beginning of the absolute path. Examples of absolute paths: "task1.inputs.ATMT:touts", "&EfaDrv.measurements.CNDR:yp".

The order and names of individual hierarchic levels are displayed in a tree structure in the REXYGEN Diagnostics program.

Inputs

| arrRef     | Array reference | reference |
| SET        | Input for initiating one-shot parameter write | bool |

Output

| E          | Error flag | bool |
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 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 the parameter.</td>
<td>bool</td>
</tr>
</tbody>
</table>
SETPR, SETPI, SETPB – Blocks for remote parameter setting

Block Symbols

 licence: STANDARD

Function Description

The SETPR, SETPI and SETPB blocks are used for setting the parameters of other blocks in the model remotely. The only difference among the three blocks is the type of parameter which they are setting. The SETPR block is used for setting real parameters, the SETPI block for integer parameters and the SETPB block for Boolean parameters.

The blocks operate in two modes, which are switched by the SETF parameter. For SETF = off the remote parameter sc is set to the value of the input signal p (or ip, P) at the start and every time when the input changes. If the SETF parameter is set to on, then the blocks work in one-shot write mode. In that case the remote parameter is set only when rising edge (off→on) occurs at the SET input. Successful modification of the remote parameter is indicated by zero error output E = off and the output y (or k, Y) is set to the value of the modified parameter. The error output is set to E = on in case of write error.

The name of the remote parameter is determined by the string parameter sc (string connection), which has the form <block_path:parameter_name>. It is also possible to access individual items of array-type parameters (e.g. the tout parameter of the ATMT block). This can be achieved using the square brackets and item number, e.g. .ATMT:touts[2]. The items are numbered from zero, thus the string connection stated above refers to the third element of the array.

The path to the block whose parameter should be set can contain hierarchic levels separated by dots followed by the block name. The path can be either relative or absolute:

- Relative – starts at the level where the SETPR block (or SETPI, SETPB) is located. The string has to be prefixed with "." in this case. Examples of relative paths: ".GAIN:k", ".Motor1.Position:ycn".

- Absolute – complete sequence of hierarchic levels down to the block. For referring to blocks located in the driver task (see the IOTASK block for details on configuration) the "&" followed by the driver’s name is used at the beginning of the absolute path. Examples of absolute paths: "task1.inputs.lin1:u2", "&EfaDrv.measurements.DER1:n".

The order and names of individual hierarchic levels are displayed in a tree structure in the REXYGEN Diagnostics program.
CHAPTER 12. PARAM – BLOCKS FOR PARAMETER HANDLING

**Inputs**

<table>
<thead>
<tr>
<th>p</th>
<th>Desired parameter value at the SETPR block input</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip</td>
<td>Desired parameter value at the SETPI block input</td>
<td>long</td>
</tr>
<tr>
<td>P</td>
<td>Desired parameter value at the SETPB block input</td>
<td>bool</td>
</tr>
<tr>
<td>SET</td>
<td>Input for initiating one-shot parameter write (off→on)</td>
<td>bool</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>y</th>
<th>Parameter value (the SETPR block)</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>Parameter value (the SETPI block)</td>
<td>long</td>
</tr>
<tr>
<td>Y</td>
<td>Parameter value (the SETPB 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>sc</th>
<th>String connection to the remote parameter respecting the above mentioned notation</th>
<th>string</th>
</tr>
</thead>
<tbody>
<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 when forced to by the SET input (rising edge)</td>
<td></td>
</tr>
</tbody>
</table>
SETPS – * Block for remote string parameter setting

Block Symbol

Function Description

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

Inputs

<table>
<thead>
<tr>
<th>sp</th>
<th>Desired parameter value</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>Input for initiating one-shot parameter write</td>
<td>bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>sc</th>
<th>String connection to the parameter</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETF</td>
<td>Set parameter only when forced to</td>
<td>bool</td>
</tr>
<tr>
<td>nmax</td>
<td>Allocated size of string</td>
<td>long</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>sy</th>
<th>Parameter value</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
SGSLP – Set, get, save and load parameters

Function Description

The SGSLP block is a special function block for manipulation with parameters of other function blocks in the REXYGEN system configuration. It works also in the Matlab-Simulink system but its scope is limited to the .mdl file it is included in.

The block can manage up to 16 parameter sets, which are numbered from 0 to 15. The number of parameter sets is given by the nps parameter and the active set is defined by the ips input. If the ips input remains unconnected, the active parameter set is ips = 0. Each set contains up to 16 different parameters defined by the string parameters sc0 to sc15. Thus the SGSLP block can work with a maximum of 256 parameters of the REXYGEN system. An empty sci string means that no parameter is specified, otherwise one of the following syntaxes is used:

1. <block>:<param> – Specifies one function block named block and its parameter param. The same block and parameter are used for all nps parameter sets in this case.

2. <block>:<param><sep>...<block>:<param> – This syntax allows the parameters to differ among the parameter sets. In general, each sci string can contain up to 16 items in the form <block>:<param> separated by comma or semi-colon. E.g. the third item of these is active for ips = 2. There should be exactly nps items in each non-empty sci string. If there is less items than nps none of the below described operations can be executed on the incomplete parameter set.

It is recommended not to use both syntaxes in one SGSLP block, all 16 sci strings should have the same form. The first syntax is for example used when producing nps types of goods, where many parameters must be changed for each type of production. The second syntax is usually used for saving user-defined parameters to disk (see the
SAVE operation below). In that case it is desirable to arrange automated switching of the ips input (e.g. using the ATMT block from the LOGIC library).

The \texttt{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 \texttt{<block>} substring in the \texttt{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 \texttt{broot} parameter is an empty string, all \texttt{<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, \texttt{broot = .} and \texttt{sc0 = CNR:ycn} must be set. Or it is possible to leave the \texttt{broot} parameter empty and put the ‘.’ character to the \texttt{sc0} string. See the \texttt{GETPR} or \texttt{SETPR} blocks description for more details about full paths in the \texttt{REXYGEN} system.

The SGSLP block executes one of the below described operations when a rising edge (\texttt{off→on}) occurs at the input of the same name. The operations are:

\texttt{SET} – Sets the parameters of the corresponding parameter set \texttt{ips} to the values of the input signals \texttt{ui}. In case the parameter is successfully set, the same value is also sent to the \texttt{yi} output.

\texttt{GET} – Gets the parameters of the corresponding parameter set \texttt{ips}. In case the parameter is successfully read, its value is sent to the \texttt{yi} output.

\texttt{SAVE} – Saves the parameters of the corresponding parameter set \texttt{ips} to a file on the target platform. The parameters of the procedure and the format of the resulting file are described below.

\texttt{LOAD} – Loads the parameters of the corresponding parameter set \texttt{ips} from a file on the target platform. This operation is executed also during the initialization of the block but only when \(0 \leq \texttt{ips0} \leq \texttt{nps} - 1\). The parameters of the procedure and the format of the file are described below.

The \texttt{LOAD} and \texttt{SAVE} operations work with a file on the target platform. The name of the file is given by the \texttt{fname} parameter and the following rules:

- If no extension is specified in the \texttt{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 \texttt{REXYGEN} system are stored. The file should be located on a media which is not erased by system restart (flash drive or hard drive, not RAM).
The SAVE operation stores the data in a text file. Two lines are added for each parameter $\text{sci}_i$, $i = 0, \ldots, m$, where $m < 16$ defines the nonempty $\text{sc}_m$ string with the highest number. The lines have the form:

"<block>:<param>", ..., "<block>:<param>"
<value>, ..., <value>

There are $\text{nps}$ 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 $\text{sci}$ containing only one <block>:<param> item (see the syntax no. 1 above). The "<block>:<param>" item is always listed $\text{nps}$-times in the file, which allows seamless switching of the $\text{sci}$ parameters syntax without modifying the file.

Consider using the SILO block if working with only a few values.

**Inputs**

- $\text{ui}_i$ $i$-th analog input signal, $i = 0, \ldots, 15$ double
- $\text{ips}$ Parameter set index (numbered from zero) long

**SET**
Set the parameters of the $\text{ips}$ parameter set according to the values of the $\text{ui}$ inputs. The values can be found at the $\text{yi}$ outputs after a successful operation.

**GET**
Get the parameters of the $\text{ips}$ parameter set. The values can be found at the $\text{yi}$ outputs after a successful operation.

**SAVE**
Save the $\text{ips}$ parameter set to a file on the target device bool

**LOAD**
Load the $\text{ips}$ parameter set from a file on the target device bool

**Outputs**

- $\text{yi}_i$ $i$-th analog output signal, $i = 0, \ldots, 15$ double
- $E$ Error flag bool
  - off ... No error
  - on .... An error occurred (see $iE$)

**Tips**

- Consider using the SILO block if working with only a few values.
- The format of the file remains the same even for $\text{sci}$ containing only one <block>:<param> item (see the syntax no. 1 above).
- The "<block>:<param>" item is always listed $\text{nps}$-times in the file, which allows seamless switching of the $\text{sci}$ parameters syntax without modifying the file.
iE  Error or warning code of the last operation  long
 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

nps  Number of parameter sets  ↓1 ↑16 ⊙1 long
ips0 Index of parameter set to load and set during the block initialization.
       No set is read for ips0 < 0 or ips0 ≥ nps  ↓-1 ↑15 long
iprec Precision (number of digits) for storing the values of double type in
       a file  ↓2 ↑15 ⊙12 long
icolw Requested column width in the status file. Spaces are appended to
       the parameter value when necessary.  ↓0 ↑22 long
fname Name of the file the SAVE and LOAD operations work with ⊙status string
broot Root block in hierarchy, inserted at the beginning of all sci
       parameters, see the description above ⊙. string
sci Strings defining the connection of ui inputs and yi outputs to the
      parameters, i = 0, . . . , 15, see details above ⊙. string
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.

The outputs E and lastErr indicate an error during disk operation. The E indicator is reset on falling edge at the SAVE or LOAD input while the lastErr output holds the value until another disk operation is invoked. If the error occurs during the LOAD operation, a substitute value yerr is set to the y output.

Alternatively it is possible to write or read the value continuously if the corresponding flag (CSF, CLF) is set to on. The disk operation is then performed when the corresponding input is set to on. Beware, in that case the disk operation is executed in each cycle, which can cause excessive use of the storage medium. Thus it is necessary to use this feature with caution.

The fname parameter defines the location of the file on the target platform. The path is relative to the data folder of the RexCore runtime module.

Use the SGSLP function block for advanced and complex operations.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>Input signal</td>
<td>double</td>
</tr>
<tr>
<td>SAVE</td>
<td>Save value to file</td>
<td>bool</td>
</tr>
<tr>
<td>LOAD</td>
<td>Load value from file</td>
<td>bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>fname</td>
<td>Name of persistent storage file</td>
<td>string</td>
</tr>
<tr>
<td>CSF</td>
<td>Flag for continuous saving</td>
<td>bool</td>
</tr>
<tr>
<td>CLF</td>
<td>Flag for continuous loading</td>
<td>bool</td>
</tr>
<tr>
<td>yerr</td>
<td>Substitute value for an error case</td>
<td>double</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Output signal</td>
<td>double</td>
</tr>
<tr>
<td>E</td>
<td>Error flag</td>
<td>bool</td>
</tr>
<tr>
<td>lastErr</td>
<td>Result of last operation</td>
<td>long</td>
</tr>
</tbody>
</table>
**SILOS – Save input string, load output string**

**Block Symbol**  
Licence: STANDARD

**Function Description**

The **SILOS** block can be used to export or import a string to/from a file. The string is saved when a rising edge (off→on) occurs at the **SAVE** input and the string is also set to the **sy** output. The string is loaded at startup and when a rising edge (off→on) occurs at the **LOAD** input.

If a logical true (on) is brought to the **APPEND** input, the input string is added to the end of the file when it is saved. This mode is useful for logging events into text files. This input signal has no effect on loading from the file.

The **LLO** parameter is intended for choosing whether to load the entire file (off) or its last line only (on).

The outputs **E** and **lastErr** indicate an error during disk operation. The **E** indicator is reset on falling edge at the **SAVE** or **LOAD** input while the **lastErr** output holds the value until another disk operation is invoked.

Alternatively it is possible to write or read the string continuously if the corresponding flag (**CSF**, **CLF**) is set to on. The disk operation is then performed when the corresponding input is set to on. Beware, in that case the disk operation is executed in each cycle, which can cause excessive use of the storage medium. Thus it is necessary to use this feature with caution.

The **fname** parameter defines the location of the file on the target platform. The path is relative to the data folder of the **RexCore** runtime module.

**Inputs**

- **su**: String input of the block  
  - **SAVE**: Save string to file  
  - **LOAD**: Load string from file  
  - **APPEND**: Append saved string to file

**Outputs**

- **sy**: String output of the block  
  - **E**: Error indicator  
    - off: No error  
    - on: An error occurred  
  - **lastErr**: Result of last operation
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fname</code></td>
<td>Name of persistent storage file</td>
<td>string</td>
</tr>
<tr>
<td><code>CSF</code></td>
<td>Continuous saving</td>
<td>bool</td>
</tr>
<tr>
<td><code>CLF</code></td>
<td>Continuous loading</td>
<td>bool</td>
</tr>
<tr>
<td><code>LL0</code></td>
<td>Last line only loading</td>
<td>bool</td>
</tr>
<tr>
<td><code>nmax</code></td>
<td>Allocated size of string</td>
<td>long</td>
</tr>
</tbody>
</table>

\[0 \leq n \leq 65520\]
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>320</td>
</tr>
<tr>
<td>CSSM</td>
<td>Continuous state space model of a linear system</td>
<td>323</td>
</tr>
<tr>
<td>DDELSSM</td>
<td>Discrete state space model of a linear system with time delay</td>
<td>325</td>
</tr>
<tr>
<td>DSSM</td>
<td>Discrete state space model of a linear system</td>
<td>327</td>
</tr>
<tr>
<td>FOPDT</td>
<td>First order plus dead-time model</td>
<td>329</td>
</tr>
<tr>
<td>MDL</td>
<td>Process model</td>
<td>330</td>
</tr>
<tr>
<td>MDLI</td>
<td>Process model with input-defined parameters</td>
<td>331</td>
</tr>
<tr>
<td>MVD</td>
<td>Motorized valve drive</td>
<td>332</td>
</tr>
<tr>
<td>SOPDT</td>
<td>Second order plus dead-time model</td>
<td>333</td>
</tr>
</tbody>
</table>
**CDELSSM** – Continuous state space model of a linear system with time delay

**Block Symbol**

CDELSSM

<table>
<thead>
<tr>
<th>R1</th>
<th>HLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>u2</td>
</tr>
<tr>
<td>u3</td>
<td>u4</td>
</tr>
<tr>
<td>u5</td>
<td>u6</td>
</tr>
<tr>
<td>u7</td>
<td>u8</td>
</tr>
<tr>
<td>u9</td>
<td>u10</td>
</tr>
<tr>
<td>u11</td>
<td>u12</td>
</tr>
<tr>
<td>u13</td>
<td>u14</td>
</tr>
<tr>
<td>u15</td>
<td>u16</td>
</tr>
</tbody>
</table>

**Function Description**

The **CDELSSM** block (Continuous State Space Model with time DELay) simulates behavior of a linear system with time delay \( del \)

\[
\frac{dx(t)}{dt} = A_c x(t) + B_c u(t - del), \quad x(0) = x_0
\]

\[
y(t) = C_c x(t) + D_c u(t),
\]

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

\[
x((k+1)T) = A_d x(kT) + B_{d1} u((k-d)T) + B_{d2} u((k-d+1)T), \quad x(0) = x_0
\]

\[
y(kT) = C_c x(kT) + D_c u(kT),
\]

where \( k \in \{1, 2, \ldots \} \) is the simulation step, \( T \) is the execution period of the block in seconds and \( d \) is a delay in simulation step such that \((d-1)T < del \leq dT\). The period \( T \) is not entered in the block, it is determined automatically as a period of the task (\texttt{TASK}, \texttt{QTASK} nebo \texttt{IOTASK}) containing the block.

If the input \( u(t) \) is changed only in the moments of sampling and between two consecutive sampling instants is constant, i.e. \( u(t) = u(kT) \) for \( t \in [kT, (k+1)T) \), then the
matrices \( A_d, B_{d1} \) and \( B_{d2} \) are determined by

\[
A_d = e^{AcT} \\
B_{d1} = e^{Ac(T-\Delta)} \int_0^\Delta e^{Ac\tau} B_c d\tau \\
B_{d2} = \int_{T-\Delta}^T e^{Ac\tau} B_c d\tau,
\]

where \( \Delta = del - (d - 1)T \).

Computation of discrete matrices \( A_d, B_{d1} \) and \( B_{d2} \) is based on a method described in [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 \( x0 \). The simulation continues on the falling edge of \( R1 \) (\( \text{on} \rightarrow \text{off} \)).
  - **bool**

- **HLD**
  - Simulation output holds its value if \( \text{HLD} = \text{on} \).
  - **bool**

- **u1..u16**
  - Simulated system inputs. First \( m \) simulation inputs are used where \( m \) is the number of columns of the matrix \( B_c \).
  - **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 REXYGEN, see appendix C for details
  - **error**

- **y1..y16**
  - Simulated system outputs. First \( p \) simulation outputs are used where \( p \) is the number of rows of the matrix \( C_c \).
  - **double**

**Parameters**

- **UD**
  - Matrix \( D_c \) usage flag. If \( UD = \text{off} \) then the \( D_c \) matrix is not used for simulation (simulation behaves as if the \( D_c \) matrix is zero).
  - **bool**

- **del**
  - Model time delay [s].
  - \( \downarrow 0.0 \)
  - **double**

- **is**
  - Order of the Padé approximation of the matrix exponential for the computation of the discretized system matrices.
  - \( \downarrow 0 \uparrow 14 \circ 2 \)
  - **long**

- **eps**
  - Required accuracy of the Padé approximation.
  - \( \downarrow 0.0 \uparrow 1.0 \circ 0.0 \)
  - **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**
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dc</td>
<td>Direct transmission (feedthrough) matrix ((p \times m)) of the continuous linear system. The matrix is used only if the parameter UD=on. If UD=off, the dimensions of the Dc matrix are not checked.</td>
<td>double</td>
</tr>
<tr>
<td>x0</td>
<td>Initial value of the state vector (of dimension (n)) of the continuous linear system.</td>
<td>double</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

\[
x((k+1)T) = A_d x(kT) + B_d u(kT), \quad x(0) = x_0
\]

\[
y(kT) = C_c x(kT) + D_c u(kT),
\]

where \( k \in \{1, 2, \ldots\} \) is the simulation step, \( T \) is the execution period of the block in seconds. 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

\[
A_d = e^{A_c T}
\]

\[
B_d = \int_0^T e^{A_c \tau} B_c d\tau
\]
CHAPTER 13. MODEL – DYNAMIC SYSTEMS SIMULATION

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 $HLD=\text{on}$.
- **u1..u16** Simulated system inputs. First $m$ simulation inputs are used where $m$ is the number of columns of the matrix $B_c$.

**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 REXYGEN, see appendix C for details
- **y1..y16** Simulated system outputs. First $p$ simulation outputs are used where $p$ is the number of rows of the matrix $C_c$.

**Parameters**

- **UD** Matrix $D_c$ usage flag. If $UD=\text{off}$ then the $D_c$ matrix is not used for simulation (simulation behaves as if the $D_c$ matrix is zero).
- **is** Order of the Padé approximation of the matrix exponential for the computation of the discretized system matrices. $\downarrow 0 \uparrow 4 \odot 2$
- **eps** Required accuracy of the Padé approximation. $\downarrow 0.0 \uparrow 1.0 \odot 0.0$
- **Ac** Matrix $(n \times n)$ of the continuous linear system dynamics.
- **Bc** Input matrix $(n \times m)$ of the continuous linear system.
- **Cc** Output matrix $(p \times n)$ of the continuous linear system.
- **Dc** Direct transmission (feedthrough) matrix $(p \times m)$ of the continuous linear system. The matrix is used only if the parameter $UD=\text{on}$. If $UD=\text{off}$, the dimensions of the $D_c$ matrix are not checked.
- **x0** Initial value of the state vector (of dimension $n$) of the continuous linear system.
**DDELSSM – Discrete state space model of a linear system with time delay**

Function Description

The **DDELSSM** block (Discrete State Space Model with time DELay) simulates behavior of a linear system with time delay $d$

$$
\begin{align*}
x(k+1) &= A_d x(k) + B_d u(k-d), \quad x(0) = x_0 \\
y(k) &= C_d x(k) + D_d u(k),
\end{align*}
$$

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. Number of steps of the delay $d$ is the largest integer such that $dT \leq del$, where $T$ is the block execution period.

All matrices are specified in the same format as in Matlab, i.e. the whole matrix is placed in brackets, elements are entered by rows, elements of a row are separated by spaces (blanks), rows are separated by semicolons. The $x_0$ vector is a column, therefore the elements are separated by semicolons (each element is in a separate row).

During the real-time simulation, single simulation step of the above discrete state space model is computed in each execution time instant.

**Inputs**

- **R1**
  - Reset signal. When $\text{R1} = \text{on}$, the state vector $x$ is set to its initial value $x_0$. The simulation continues on the falling edge of $\text{R1}$ ($\text{on} \rightarrow \text{off}$).
  - bool

- **HLD**
  - Simulation output holds its value if $\text{HLD} = \text{on}$.
  - bool

- **u1..u16**
  - Simulated system inputs. First $m$ simulation inputs are used where $m$ is the number of columns of the matrix $B_d$.
  - double
Outpus

iE  Block error code  
    0 . . . . O.K., the simulation runs correctly
    -213 . . incompatibility of the state space model matrices dimensions
    xxx . . error code xxx of REXYGEN, see appendix C for details

y1..y16  Simulated system outputs. First p simulation outputs are used where p is the number of rows of the matrix Cd.

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).

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 Dd matrix are not checked.

x0   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

\[
\begin{align*}
x(k+1) &= A_dx(k) + B_d u(k), \quad x(0) = x_0 \\
y(k) &= C_d x(k) + D_d u(k),
\end{align*}
\]

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 \( R1 = \text{on} \), the state vector \( x \) is set to its initial value \( x_0 \). The simulation continues on the falling edge of \( R1 \) (on→off).)
  - bool

- **HLD** (simulation output holds its value if \( HLD=\text{on} \).)
  - bool

- **u1..u16** (simulated system inputs. First \( m \) simulation inputs are used where \( m \) is the number of columns of the matrix \( B_d \).)
  - double
CHAPTER 13. MODEL – DYNAMIC SYSTEMS SIMULATION

Outputs

\textbf{iE}  
Block error code  
\begin{itemize}  
\item 0 \ldots \text{O.K., the simulation runs correctly}  
\item -213 \ldots \text{incompatibility of the state space model matrices dimensions}  
\item xxx \ldots \text{error code xxx of REXYGEN, see appendix C for details}  
\end{itemize} 

\textbf{y1..y16}  
Simulated system outputs. First \( p \) simulation outputs are used where \( p \) is the number of rows of the matrix \( Cd \).

Parameters

\textbf{UD}  
Matrix \( Dd \) usage flag. If \( UD=\text{off} \) then the \( Dd \) matrix is not used for simulation (simulation behaves as if the \( Dd \) matrix is zero).

\textbf{Ad}  
Matrix \((n \times n)\) of the discrete linear system dynamics.

\textbf{Bd}  
Input matrix \((n \times m)\) of the discrete linear system.

\textbf{Cd}  
Output matrix \((p \times n)\) of the discrete linear system.

\textbf{Dd}  
Direct transmission (feedthrough) matrix \((p \times m)\) of the discrete linear system. The matrix is used only if the parameter \( UD=\text{on} \). If \( UD=\text{off} \), the dimensions of the \( Dd \) matrix are not checked.

\textbf{x0}  
Initial value of the state vector (of dimension \( n \)) of the discrete linear system.
**FOPDT – First order plus dead-time model**

**Block Symbol**

![FOPDT Block Symbol](image)

**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:

\[ P(s) = \frac{k_0}{(\tau s + 1)} e^{-\text{del} \cdot s} \]

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

**Input**

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

**Output**

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

**Parameters**

- \( k_0 \): Static gain  
  \( \ominus 1.0 \text{ double} \)
- \( \text{del} \): Dead time [s]  
  \text{double}
- \( \tau \): Time constant  
  \( \ominus 1.0 \text{ double} \)
- \( \text{nmax} \): Size of delay buffer (number of samples) for the time delay \( \text{del} \). Used for internal memory allocation.  
  \(|10 \uparrow 1000000 \ominus 1000| \) 
  
  \text{long}
**MDL – Process model**

**Block Symbol**

![MDL Block Diagram](image)

**Licence:** STANDARD

**Function Description**

The MDL block is a discrete simulator of continuous-time system with transfer function

\[ F(s) = \frac{K_0 e^{-Ds}}{(\tau_1 s + 1)(\tau_2 s + 1)}, \]

where \( K_0 > 0 \) is the static gain \( k_0 \), \( D \geq 0 \) is the time-delay \( \text{del} \) and \( \tau_1, \tau_2 > 0 \) are the system time-constants \( \tau_1 \) and \( \tau_2 \).

**Input**

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

**Output**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k_0 )</td>
<td>Static gain</td>
<td>( \odot 1.0 )</td>
<td>double</td>
</tr>
<tr>
<td>( \text{del} )</td>
<td>Dead time [s]</td>
<td></td>
<td>double</td>
</tr>
<tr>
<td>( \tau_1 )</td>
<td>The first time constant</td>
<td>( \odot 1.0 )</td>
<td>double</td>
</tr>
<tr>
<td>( \tau_2 )</td>
<td>The second time constant</td>
<td>( \odot 2.0 )</td>
<td>double</td>
</tr>
<tr>
<td>( n_{\text{max}} )</td>
<td>Size of delay buffer (number of samples) for the time delay ( \text{del} ). Used for internal memory allocation.</td>
<td>( \downarrow 10 \uparrow 10000000 \odot 1000 )</td>
<td>long</td>
</tr>
</tbody>
</table>
**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**

<table>
<thead>
<tr>
<th>Input</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>k0</td>
<td>Static gain</td>
<td>double</td>
</tr>
<tr>
<td>del</td>
<td>Dead time [s]</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>
</tbody>
</table>

**Output**

<table>
<thead>
<tr>
<th>Output</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>nmax</td>
<td>Size of delay buffer (number of samples) for the time delay ( \text{del} ). Used for internal memory allocation.</td>
<td>long</td>
</tr>
</tbody>
</table>

\( \downarrow10 \uparrow10000000 \odot1000 \)
CHAPTER 13. MODEL – DYNAMIC SYSTEMS SIMULATION

MVD – Motorized valve drive

Block Symbol Licence: STANDARD

Function Description

The MVD block simulates a servo valve. The UP (DN) input is a binary command for opening (closing) the valve at a constant speed \( \frac{1}{tv} \), where \( tv \) is a parameter of the block. The opening (closing) continues for \( UP = \text{on} \) (\( DN = \text{on} \)) until the full open \( y = \text{hilim} \) (full closed \( y = \text{lolim} \)) position is reached. The full open (full closed) position is signalized by the end switch HS (LS). The initial position at start-up is \( y = y_0 \). If \( UP = DN = \text{on} \) or \( UP = DN = \text{off} \), then the position of the valve remains unchanged (neither opening nor closing).

Inputs

\[
\begin{align*}
\text{UP} & \quad \text{Open} & \text{bool} \\
\text{DN} & \quad \text{Close} & \text{bool}
\end{align*}
\]

Outputs

\[
\begin{align*}
y & \quad \text{Valve position} & \text{double} \\
\text{HS} & \quad \text{Upper end switch} & \text{bool} \\
\text{LS} & \quad \text{Lower end switch} & \text{bool}
\end{align*}
\]

Parameters

\[
\begin{align*}
y_0 & \quad \text{Initial valve position} & \text{double} \\
tv & \quad \text{Time required for transition between } y = 0 \text{ and } y = 1 \text{ [s]} & \odot 10.0 & \text{double} \\
\text{hilim} & \quad \text{Upper limit position (open)} & \odot 1.0 & \text{double} \\
\text{lolim} & \quad \text{Lower limit position (closed)} & \odot 0.0 & \text{double}
\end{align*}
\]
**SOPDT – Second order plus dead-time model**

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.

\[
\begin{align*}
\text{itf} = 1: & \quad P(s) = \frac{pb1 \cdot s + pb0}{s^2 + pa1 \cdot s + pa0} \cdot e^{-\text{del} \cdot s} \\
\text{itf} = 2: & \quad P(s) = \frac{k0(\tau \cdot s + 1)}{(\tau_1 \cdot s + 1)(\tau_2 \cdot s + 1)} \cdot e^{-\text{del} \cdot s} \\
\text{itf} = 3: & \quad P(s) = \frac{k0 \cdot \omega^2 \cdot (\tau / \omega \cdot s + 1)}{(s^2 + 2 \cdot \xi \cdot \omega \cdot s + \omega^2)} \cdot e^{-\text{del} \cdot s} \\
\text{itf} = 4: & \quad P(s) = \frac{k0(\tau \cdot s + 1)}{(\tau_1 \cdot s + 1) \cdot s} \cdot e^{-\text{del} \cdot s}
\end{align*}
\]

For simulation of first order plus dead time systems (FOPDT) use the LLC block with parameter \( a \) 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 \hspace{1cm} \text{double}

**Output**

- \( y \) Analog output of the block \hspace{1cm} \text{double}

**Parameters**

- \( \text{itf} \) Transfer function form \hspace{1cm} \( \odot 1 \hspace{0.1cm} \text{long} \)
  - 1: A general form
  - 2: A form with real poles
  - 3: A form with complex poles
  - 4: A form with integrator
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Default Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>k0</td>
<td>Static gain</td>
<td>1.0</td>
<td>double</td>
</tr>
<tr>
<td>tau</td>
<td>Numerator time constant</td>
<td></td>
<td>double</td>
</tr>
<tr>
<td>tau1</td>
<td>The first time constant</td>
<td>1.0</td>
<td>double</td>
</tr>
<tr>
<td>tau2</td>
<td>The second time constant</td>
<td>1.0</td>
<td>double</td>
</tr>
<tr>
<td>om</td>
<td>Natural frequency</td>
<td>1.0</td>
<td>double</td>
</tr>
<tr>
<td>xi</td>
<td>Relative damping coefficient</td>
<td>1.0</td>
<td>double</td>
</tr>
<tr>
<td>pb0</td>
<td>Numerator coefficient: $s^0$</td>
<td>1.0</td>
<td>double</td>
</tr>
<tr>
<td>pb1</td>
<td>Numerator coefficient: $s^1$</td>
<td>1.0</td>
<td>double</td>
</tr>
<tr>
<td>pa0</td>
<td>Denominator coefficient: $s^0$</td>
<td>1.0</td>
<td>double</td>
</tr>
<tr>
<td>pa1</td>
<td>Denominator coefficient: $s^1$</td>
<td>1.0</td>
<td>double</td>
</tr>
<tr>
<td>del</td>
<td>Dead time [s]</td>
<td></td>
<td>double</td>
</tr>
<tr>
<td>nmax</td>
<td>Size of delay buffer (number of samples) for the time delay del. Used for internal memory allocation.</td>
<td>10000000</td>
<td>long</td>
</tr>
</tbody>
</table>
Chapter 14

MATRIX – Blocks for matrix and vector operations

Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNA</td>
<td>Array (vector/matrix) constant</td>
<td>338</td>
</tr>
<tr>
<td>MB_DASUM</td>
<td>Sum of the absolute values</td>
<td>339</td>
</tr>
<tr>
<td>MB_DAXPY</td>
<td>Performs y := a*x + y for vectors x,y</td>
<td>340</td>
</tr>
<tr>
<td>MB_DCOPY</td>
<td>Copies vector x to vector y</td>
<td>342</td>
</tr>
<tr>
<td>MB_DDOT</td>
<td>Dot product of two vectors</td>
<td>344</td>
</tr>
<tr>
<td>MB_DGEMM</td>
<td>Performs C := alpha*op(A)<em>op(B) + beta</em>C, where op(X) = X or op(X) = X^T</td>
<td>346</td>
</tr>
<tr>
<td>MB_DGEMV</td>
<td>Performs y := alpha<em>A</em>x + beta<em>y or y := alpha</em>A^T*x</td>
<td>348</td>
</tr>
<tr>
<td>MB_DGER</td>
<td>Performs A := alpha<em>x</em>y^T + A</td>
<td>350</td>
</tr>
<tr>
<td>MB_DNRM</td>
<td>Euclidean norm of a vector</td>
<td>352</td>
</tr>
<tr>
<td>MB_DROT</td>
<td>Plain rotation of a vector</td>
<td>353</td>
</tr>
<tr>
<td>MB_DSCAL</td>
<td>Scales a vector by a constant</td>
<td>355</td>
</tr>
<tr>
<td>MB_DSWAP</td>
<td>Interchanges two vectors</td>
<td>356</td>
</tr>
<tr>
<td>MB_DTRMM</td>
<td>Performs B := alpha<em>op(A)<em>B or B := alpha</em>B</em>op(A), where op(X) = X or op(X) = X^T for triangular matrix A</td>
<td>358</td>
</tr>
<tr>
<td>MB_DTRMV</td>
<td>Performs x := A<em>x or x := A^T</em>x for triangular matrix A</td>
<td>360</td>
</tr>
<tr>
<td>MB_DTRSV</td>
<td>Solves one of the system of equations A<em>x = b or A^T</em>x = b for triangular matrix A</td>
<td>363</td>
</tr>
<tr>
<td>ML_DGEBAK</td>
<td>Backward transformation to ML_DGEBAL of left or right eigenvectors</td>
<td>366</td>
</tr>
<tr>
<td>ML_DGEBAL</td>
<td>Balancing of a general real matrix</td>
<td>368</td>
</tr>
<tr>
<td>ML_DGEBRD</td>
<td>Reduces a general real matrix to bidiagonal form by an orthogonal transformation</td>
<td>370</td>
</tr>
</tbody>
</table>
**CHAPTER 14. MATRIX – BLOCKS FOR MATRIX AND VECTOR OPERATIONS**

- **ML_DGECON** – Estimates the reciprocal of the condition number of a general real matrix ................................................. 372
- **ML_DGEEES** – Computes the eigenvalues, the Schur form, and, optionally, the matrix of Schur vectors ........................................... 374
- **ML_DGEEV** – Computes the eigenvalues and, optionally, the left and/or right eigenvectors .............................................. 376
- **ML_DGEBRD** – Reduces a real general matrix A to upper Hessenberg form ................................................................................. 378
- **ML_DGELQF** – Computes an LQ factorization of a real M-by-N matrix A .................................................................................. 380
- **ML_DGELSD** – Computes the minimum-norm solution to a real linear least squares problem .................................................. 382
- **ML_DGEQRF** – Computes an QR factorization of a real M-by-N matrix A .................................................................................. 384
- **ML_DGESDD** – Computes the singular value decomposition (SVD) of a real M-by-N matrix A .................................................. 386
- **ML_DLACPY** – Copies all or part of one matrix to another matrix ......................................................................................... 388
- **ML_DLANGE** – Computes one of the matrix norms of a general matrix .................................................................................. 390
- **ML_DLASET** – Initializes the off-diagonal elements and the diagonal elements of a matrix to given values .......................... 392
- **ML_DTRSYL** – Solves the real Sylvester matrix equation for quasi-triangular matrices A and B ................................................. 394
- **MX_AT** – Get Matrix/Vector element ................................................................................................................................. 396
- **MX_ATSET** – Set Matrix/Vector element ................................................................................................................................. 397
- **MX_CNADD** – Add scalar to each Matrix/Vector element .......................................................................................................... 398
- **MX_CMUL** – Multiply a Matrix/Vector by a scalar ..................................................................................................................... 399
- **MX_CTODPA** – Discretizes continuous model given by (A,B) to (Ad,Bd) using Padé approximations .............................................. 400
- **MX_DIM** – Matrix/Vector dimensions ................................................................................................................................. 402
- **MX_DIMSET** – Set Matrix/Vector dimensions ......................................................................................................................... 403
- **MX_DSAGET** – Set subarray of A into B ................................................................................................................................. 404
- **MX_DSAREF** – Set reference to subarray of A into B .................................................................................................................... 406
- **MX_DSASET** – Set A into subarray of B ................................................................................................................................. 408
- **MX_DTRNSP** – General matrix transposition: B := alpha*A^T ................................................................................................. 410
- **MX_DTRNSQ** – Square matrix in-place transposition: A := alpha*A^T ..................................................................................... 412
- **MX_FILL** – Fill real matrix or vector ................................................................................................................................. 413
- **MX_MAT** – Matrix data storage block ................................................................................................................................. 414
- **MX_RAND** – Randomly generated matrix or vector ................................................................................................................ 415
- **MX_REFCOPY** – Copies input references of matrices A and B to their output references ..................................................... 417
- **MX_VEC** – Vector data storage block ................................................................................................................................. 418
- **MX_WRITE** – Write a Matrix/Vector to the console/system log ............................................................................................ 419
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTOV</td>
<td>Vector multiplexer</td>
<td>421</td>
</tr>
<tr>
<td>SWVMR</td>
<td>Vector/matrix/reference signal switch</td>
<td>422</td>
</tr>
<tr>
<td>VTOR</td>
<td>Vector demultiplexer</td>
<td>423</td>
</tr>
</tbody>
</table>
CNA – Array (vector/matrix) constant

Block Symbol Licence: STANDARD

Function Description

The block CNA allocates memory for nmax elements of the type etype of the vector/matrix referenced by the output vec and initializes all elements to data stored in the parameter acn. If the string parameter filename is not empty then it loads initialzation data from the filename. If the parameter TRN = on then the output reference vec contains transposed data.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>CSV data file</td>
</tr>
<tr>
<td>TRN</td>
<td>Transpose loaded matrix</td>
</tr>
<tr>
<td>nmax</td>
<td>Allocated size of output matrix (total number of items)</td>
</tr>
<tr>
<td>etype</td>
<td>Type of elements</td>
</tr>
<tr>
<td>acn</td>
<td>Initial array value</td>
</tr>
</tbody>
</table>

Output

| vec | Reference to vector/matrix data | reference |
MB_DASUM – Sum of the absolute values

Block Symbol

Function Description

The output reference $yX$ is always set to the input reference $uX$. If $HLD = \text{on}$ then nothing is computed otherwise the BLAS function DASUM is called internally:

$$\text{value} = \text{DASUM}(N, uX, \text{INCX});$$

where the values $N$ and $\text{INCX}$ are set in the following way:

- If the input $n > 0$ then $N$ is set to $n$ else $N$ is set to the current number of the vector or matrix elements $\text{CNT}$ referenced by $uX$.
- If the input $\text{incx} > 0$ then $\text{INCX}$ is set to $\text{incx}$ else $\text{INCX}$ is set to 1.

The error flag $E$ is set to on if:

- the reference $uX$ is not defined (i.e. input $uX$ is not connected),
- $n < 0$ or $\text{incx} < 0$,
- $(N - 1) \times \text{INCX} + 1 > \text{CNT}$.


Inputs

| $uX$ | Input reference to vector $x$ | reference |
| $n$  | Number of processed vector elements | long |
| $\text{incx}$ | Index increment of vector $x$ | long |
| $\text{HLD}$ | Hold | bool |

Outputs

| $yX$ | Output reference to vector $x$ | reference |
| $\text{value}$ | Return value of the function | double |
| $E$ | Error flag | bool |
**MB_DAXPY** – Performs \( y := a \cdot x + y \) for vectors \( x, y \)

**Block Symbol**

![MB_DAXPY Block Symbol](image)

**Licence:** STANDARD

**Function Description**

The output references \( yX \) and \( yY \) are always set to the corresponding input references \( uX \) and \( uY \). If \( HLD = \text{on} \) then nothing is computed otherwise the BLAS function \( \text{DAXPY} \) is called internally:

\[
\text{DAXPY}(N, a, uX, \text{INCX}, uY, \text{INCY});
\]

where the values \( N, \text{INCX} \) and \( \text{INCY} \) are set in the following way:

- If the input \( n > 0 \) then \( N \) is set to \( n \) else \( N \) is set to the current number of the vector or matrix elements \( \text{CNTX} \) referenced by \( uY \).
- If the input \( \text{INCX} \neq 0 \) then \( \text{INCX} \) is set to \( \text{INCX} \) else \( \text{INCX} \) is set to 1.
- If the input \( \text{INCY} \neq 0 \) then \( \text{INCY} \) is set to \( \text{INCY} \) else \( \text{INCY} \) is set to 1.

The error flag \( E \) is set to \( \text{on} \) if:

- the reference \( uX \) or \( uY \) is not defined (i.e. input \( uX \) or \( uY \) is not connected),
- \( n < 0 \),
- \( (N - 1) \ast |\text{INCX}| + 1 > \text{CNTX} \), where \( \text{CNTX} \) is a number of the vector or matrix elements referenced by \( uX \),
- \( (N - 1) \ast |\text{INCY}| + 1 > \text{CNTY} \).


**Inputs**

<p>| ( uX ) | Input reference to vector ( x ) | reference |
| ( uY ) | Input reference to vector ( y ) | reference |
| ( a )   | Scalar coefficient ( a )         | double    |
| ( n )   | Number of processed vector elements | long      |</p>
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>incx</td>
<td>Index increment of vector x</td>
<td>long</td>
</tr>
<tr>
<td>incy</td>
<td>Index increment of vector y</td>
<td>long</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yX</td>
<td>Output reference to vector x</td>
<td>reference</td>
</tr>
<tr>
<td>yY</td>
<td>Output reference to vector y</td>
<td>reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
MB_DCOPY – Copies vector x to vector y

Block Symbol

<table>
<thead>
<tr>
<th>uX</th>
<th>uY</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>incx</td>
</tr>
<tr>
<td>incy</td>
<td>HLD</td>
</tr>
</tbody>
</table>

Licence: STANDARD

Function Description

The output references yX and yY are always set to the corresponding input references uX and uY. If HLD = on then nothing is computed otherwise the BLAS function DCOPY is called internally:

\[
\text{DCOPY}(N, uX, \text{INCX}, uY, \text{INCY});
\]

where the values N, INGX and INCY are set in the following way:

- If the input \( n > 0 \) then \( N \) is set to \( n \) else \( N \) is set to the current number of the vector or matrix elements CNTX referenced by uX.
- If the input \( \text{incx} \neq 0 \) then \( \text{INCX} \) is set to \( \text{incx} \) else \( \text{INCX} \) is set to 1.
- If the input \( \text{incy} \neq 0 \) then \( \text{INCY} \) is set to \( \text{incy} \) else \( \text{INCY} \) is set to 1.

The error flag \( E \) is set to on if:

- the reference uX or uY is not defined (i.e. input uX or uY is not connected),
- \( n < 0 \),
- \( (N - 1) \times |\text{INCX}| + 1 > CNTX \),
- \( (N - 1) \times |\text{INCY}| + 1 > CNTY \), where CNTY is a number of the vector or matrix elements referenced by uY.


Inputs

<table>
<thead>
<tr>
<th>uX</th>
<th>Input reference to vector x</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uY</td>
<td>Input reference to vector y</td>
<td>reference</td>
</tr>
<tr>
<td>n</td>
<td>Number of processed vector elements</td>
<td>long</td>
</tr>
<tr>
<td>incx</td>
<td>Index increment of vector x</td>
<td>long</td>
</tr>
<tr>
<td>incy</td>
<td>Index increment of vector y</td>
<td>long</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yX</td>
<td>Output reference to vector x</td>
<td>reference</td>
</tr>
<tr>
<td>yY</td>
<td>Output reference to vector y</td>
<td>reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
MB_DDOT – Dot product of two vectors

Function Description

The output references yX and yY are always set to the corresponding input references uX and uY. If HLD = on then nothing is computed otherwise the BLAS function DDOT is called internally:

\[ \text{DDOT}(N, \text{uX}, \text{INCX}, \text{uY}, \text{INCY}); \]

where the values N, INCX and INCY are set in the following way:

- If the input \( n > 0 \) then \( N \) is set to \( n \) else \( N \) is set to the current number of the vector or matrix elements CNTX referenced by \( \text{uX} \).
- If the input \( \text{incx} \neq 0 \) then \( \text{INCX} \) is set to \( \text{incx} \) else \( \text{INCX} \) is set to 1.
- If the input \( \text{incy} \neq 0 \) then \( \text{INCY} \) is set to \( \text{incy} \) else \( \text{INCY} \) is set to 1.

The error flag \( E \) is set to on if:

- the reference \( \text{uX} \) or \( \text{uY} \) is not defined (i.e. input \( \text{uX} \) or \( \text{uY} \) is not connected),
- \( n < 0 \),
- \( (N - 1) \cdot |\text{INCX}| + 1 > \text{CNTX} \),
- \( (N - 1) \cdot |\text{INCY}| + 1 > \text{CNTY} \), where \( \text{CNTY} \) is a number of the vector or matrix elements referenced by \( \text{uY} \).


Inputs

| \( \text{uX} \) | Input reference to vector x | \( \text{reference} \) |
| \( \text{uY} \) | Input reference to vector y | \( \text{reference} \) |
| \( \text{n} \) | Number of processed vector elements | \( \text{long} \) |
| \( \text{incx} \) | Index increment of vector x | \( \text{long} \) |
| \( \text{incy} \) | Index increment of vector y | \( \text{long} \) |
| \( \text{HLD} \) | Hold | \( \text{bool} \) |
Outputs

- **yX**: Output reference to vector x
- **yY**: Output reference to vector y
- **value**: Return value of the function
- **E**: Error indicator

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yX</td>
<td>Output reference to vector x</td>
<td>reference</td>
</tr>
<tr>
<td>yY</td>
<td>Output reference to vector y</td>
<td>reference</td>
</tr>
<tr>
<td>value</td>
<td>Return value of the function</td>
<td>double</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
**MB_DGEMM** – Performs $C := \alpha \cdot \text{op}(A) \cdot \text{op}(B) + \beta \cdot C$, where $\text{op}(X) = X$ or $\text{op}(X) = X^T$

**Function Description**

The output references $yA$, $yB$ and $yC$ are always set to the corresponding input references $uA$, $uB$ and $uC$. If $HLD = \text{on}$ then nothing is computed otherwise the BLAS function $\text{DGEMM}$ is called internally:

$$\text{DGEMM}(\text{sTRANSA}, \text{sTRANSB}, M, N, KA, \alpha, uA, LDA, uB, LDB, \beta, uC, LDC);$$

where parameters of $\text{DGEMM}$ are set in the following way:

- Integer inputs $\text{transa}$ and $\text{transb}$ are mapped to strings $\text{sTRANSA}$ and $\text{sTRANSB}$:
  $\{0, 1\} \rightarrow "N", \{2\} \rightarrow "T"$ and $\{3\} \rightarrow "C"$.
- $M$ is number of rows of the matrix referenced by $uC$.
- $N$ is number of columns of the matrix referenced by $uC$.
- If the input $\text{transa}$ is equal to 0 or 1 then $KA$ is number of columns else $KA$ is number rows of the matrix referenced by $uA$.
- $LDA$, $LDB$ and $LDC$ are leading dimensions of matrices referenced by $uA$, $uB$ and $uC$.

The error flag $E$ is set to $\text{on}$ if:

- the reference $uA$ or $uB$ or $uC$ is not defined (i.e. input $uA$ or $uB$ or $uC$ is not connected),
- $\text{transa}$ or $\text{transb}$ is less than 0 or greater than 3
- $KA \neq KB$; if the input $\text{transb}$ is equal to 0 or 1 then $KB$ is number of rows else $KB$ is number of columns of the matrix referenced by $uB$ (i.e. matrices $\text{op}(A)$ and $\text{op}(B)$ have to be multipliable).
- the call of the function $\text{DGEMM}$ returns error using the function $\text{XERBLA}$, see the system log.

Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>uB</td>
<td>Input reference to matrix B</td>
<td>reference</td>
</tr>
<tr>
<td>uC</td>
<td>Input reference to matrix C</td>
<td>reference</td>
</tr>
<tr>
<td>transa</td>
<td>Transposition of matrix A</td>
<td>↓0 ↑3 long</td>
</tr>
<tr>
<td>transb</td>
<td>Transposition of matrix B</td>
<td>↓0 ↑3 long</td>
</tr>
<tr>
<td>alpha</td>
<td>Scalar coefficient alpha</td>
<td>double</td>
</tr>
<tr>
<td>beta</td>
<td>Scalar coefficient beta</td>
<td>double</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>yB</td>
<td>Output reference to matrix B</td>
<td>reference</td>
</tr>
<tr>
<td>yC</td>
<td>Output reference to matrix C</td>
<td>reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
**MB_DGEMV** – Performs \( y := \alpha A^* x + \beta y \) or \( y := \alpha A^T x + \beta y \)

**Function Description**

The output references \( yA, yX \) and \( yY \) are always set to the corresponding input references \( uA, uX \) and \( uY \). If \( HLD = \text{on} \) then nothing is computed otherwise the BLAS function \texttt{DGEMV} is called internally:

\[
\texttt{DGEMV(sTRANS, M, N, alpha, uA, LDA, uX, INCX, beta, uY, INCY)};
\]

where parameters of \texttt{DGEMV} are set in the following way:

- Integer input \( \text{trans} \) is mapped to the string \( \text{sTRANS}: \{0,1\} \to \"N\", \{2\} \to \"T\" \) and \( \{3\} \to \"C\" \).
- \( M \) is number of rows of the matrix referenced by \( uA \).
- \( N \) is number of columns of the matrix referenced by \( uA \).
- \( LDA \) is the leading dimension of matrix referenced by \( uA \).
- If the input \( \text{incx} \neq 0 \) then \( \text{INCX} \) is set to \( \text{incx} \) else \( \text{INCY} \) is set to 1.
- If the input \( \text{incy} \neq 0 \) then \( \text{INCY} \) is set to \( \text{incy} \) else \( \text{INCY} \) is set to 1.

The error flag \( E \) is set to \( \text{on} \) if:

- the reference \( uA \) or \( uX \) or \( uY \) is not defined (i.e. input \( uA \) or \( uX \) or \( uY \) is not connected),
- \( \text{trans} \) is less than 0 or greater than 3
- the call of the function \texttt{DGEMV} returns error using the function \texttt{XERBLA}, see the system log.

## Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>uX</td>
<td>Input reference to vector x</td>
<td>reference</td>
</tr>
<tr>
<td>uY</td>
<td>Input reference to vector y</td>
<td>reference</td>
</tr>
<tr>
<td>trans</td>
<td>Transposition of the input matrix</td>
<td>long</td>
</tr>
<tr>
<td>incx</td>
<td>Index increment of vector x</td>
<td>long</td>
</tr>
<tr>
<td>incy</td>
<td>Index increment of vector y</td>
<td>long</td>
</tr>
<tr>
<td>alpha</td>
<td>Scalar coefficient alpha</td>
<td>double</td>
</tr>
<tr>
<td>beta</td>
<td>Scalar coefficient beta</td>
<td>double</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>yX</td>
<td>Output reference to vector x</td>
<td>reference</td>
</tr>
<tr>
<td>yY</td>
<td>Output reference to vector y</td>
<td>reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
**MB_DGER** – Performs $A := \alpha x^* y^T + A$

**Block Symbol**

![Block Symbol](image)

**Function Description**

The output references $yX$, $yY$ and $yA$ are always set to the corresponding input references $uX$, $uY$ and $uA$. If HLD = on then nothing is computed otherwise the BLAS function DGER is called internally:

$$\text{DGER}(M, N, \alpha, uX, \text{INCX}, uY, \text{INCY}, uA, \text{LDA});$$

where parameters of DGER are set in the following way:

- $M$ is number of rows of the matrix referenced by $uA$.
- $N$ is number of columns of the matrix referenced by $uA$.
- If the input $\text{incx} \neq 0$ then $\text{INCX}$ is set to $\text{incx}$ else $\text{INCX}$ is set to 1.
- If the input $\text{incy} \neq 0$ then $\text{INCY}$ is set to $\text{incy}$ else $\text{INCY}$ is set to 1.
- $\text{LDA}$ is the leading dimension of matrix referenced by $uA$.

The error flag $E$ is set to on if:

- the reference $uX$ or $uY$ or $uA$ is not defined (i.e. input $uX$ or $uY$ or $uA$ is not connected),
- the call of the function DGER returns error using the function XERBLA, see the system log.


**Inputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uX</td>
<td>Input reference to vector x</td>
<td>reference</td>
</tr>
<tr>
<td>uY</td>
<td>Input reference to vector y</td>
<td>reference</td>
</tr>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>incx</td>
<td>Index increment of vector x</td>
<td>long</td>
</tr>
<tr>
<td>incy</td>
<td>Index increment of vector y</td>
<td>long</td>
</tr>
<tr>
<td>alpha</td>
<td>Scalar coefficient alpha</td>
<td>double</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>
## Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yX</td>
<td>Output reference to vector x</td>
<td>reference</td>
</tr>
<tr>
<td>yY</td>
<td>Output reference to vector y</td>
<td>reference</td>
</tr>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
**MB_DNRM2 – Euclidean norm of a vector**

**Function Description**

The output reference \( yX \) is always set to the input reference \( uX \). If \( HLD = \text{on} \) then nothing is computed otherwise the BLAS function \( \text{DNRM2} \) is called internally:

\[
\text{value} = \text{DNRM2}(N, uX, \text{INCX});
\]

where the values \( N \) and \( \text{INCX} \) are set in the following way:

- If the input \( n > 0 \) then \( N \) is set to \( n \) else \( N \) is set to the current number of the vector or matrix elements \( \text{CNT} \) referenced by \( uX \).
- If the input \( \text{incx} > 0 \) then \( \text{INCX} \) is set to \( \text{incx} \) else \( \text{INCX} \) is set to 1.

The error flag \( E \) is set to \( \text{on} \) if:

- the reference \( uX \) is not defined (i.e. input \( uX \) is not connected),
- \( n < 0 \) or \( \text{incx} < 0 \),
- \( (N - 1) \ast |\text{INCX}| + 1 > \text{CNT} \).


Use the block \( \text{ML_DLANGE} \) for computation of various norms of a matrix.

**Inputs**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( uX )</td>
<td>Input reference to vector x</td>
<td>reference</td>
</tr>
<tr>
<td>( n )</td>
<td>Number of processed vector elements</td>
<td>long</td>
</tr>
<tr>
<td>( \text{incx} )</td>
<td>Index increment of vector x</td>
<td>long</td>
</tr>
<tr>
<td>( HLD )</td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( yX )</td>
<td>Output reference to vector x</td>
<td>reference</td>
</tr>
<tr>
<td>( \text{value} )</td>
<td>Return value of the function</td>
<td>double</td>
</tr>
<tr>
<td>( E )</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
MB_DROT – Plain rotation of a vector

Block Symbol

Function Description

The output references yX and yY are always set to the corresponding input references uX and uY. If HLD = on then nothing is computed otherwise the BLAS function DROT is called internally:

\[ \text{DROT}(N, uX, \text{INCX}, uY, \text{INCY}, c, s); \]

where parameters of DROT are set in the following way:

- If the input \( n > 0 \) then \( N \) is set to \( n \) else \( N \) is set to the current number of the vector or matrix elements \( \text{CNTX} \) referenced by \( uX \).
- If the input \( \text{INCX} \neq 0 \) then \( \text{INCX} \) is set to \( \text{INCX} \) else \( \text{INCX} \) is set to 1.
- If the input \( \text{INCY} \neq 0 \) then \( \text{INCY} \) is set to \( \text{INCY} \) else \( \text{INCY} \) is set to 1.

The error flag E is set to on if:

- the reference \( uX \) or \( uY \) is not defined (i.e. input \( uX \) or \( uY \) is not connected),
- \( n < 0 \),
- \((N - 1) * |\text{INCX}| + 1 > \text{CNTX}\),
- \((N - 1) * |\text{INCY}| + 1 > \text{CNTY}\), where \( \text{CNTY} \) is a number of the vector or matrix elements referenced by \( uY \).


Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uX</td>
<td>Input reference to vector x</td>
<td>reference</td>
</tr>
<tr>
<td>uY</td>
<td>Input reference to vector y</td>
<td>reference</td>
</tr>
<tr>
<td>n</td>
<td>Number of processed vector elements</td>
<td>long</td>
</tr>
<tr>
<td>incx</td>
<td>Index increment of vector x</td>
<td>long</td>
</tr>
</tbody>
</table>
CHAPTER 14. MATRIX – BLOCKS FOR MATRIX AND VECTOR OPERATIONS

incy  Index increment of vector y  long
c      Scalar coefficient c       double
s      Scalar coefficient s      double
HLD    Hold                       bool

Outputs

yx     Output reference to vector x  reference
yx  Output reference to vector y  reference
E      Error indicator              bool
MB_DSCAL – Scales a vector by a constant

Block Symbol

Licence: STANDARD

Function Description

The output references $yX$ is always set to the corresponding input reference $uX$. If $HLD = \text{on}$ then nothing is computed otherwise the BLAS function $\text{DSCAL}$ is called internally:

$$\text{DSCAL}(N, a, uX, \text{INCX});$$

where parameters of $\text{DSCAL}$ are set in the following way:

- If the input $n > 0$ then $N$ is set to $n$ else $N$ is set to the current number of the vector or matrix elements $\text{CNT}$ referenced by $uX$.
- If the input $\text{INCX} \neq 0$ then $\text{INCX}$ is set to $\text{INCX}$ else $\text{INCX}$ is set to 1.

The error flag $E$ is set to $\text{on}$ if:

- the reference $uX$ is not defined (i.e. input $uX$ is not connected),
- $n < 0$ or $\text{INCX} < 0$,
- $(N - 1) \times \text{INCX} + 1 > \text{CNT}$.


Inputs

- $uX$ Input reference to vector $x$ reference
- $a$ Scalar coefficient $a$ double
- $n$ Number of processed vector elements long
- $\text{INCX}$ Index increment of vector $x$ long
- $\text{HLD}$ Hold bool

Outputs

- $yX$ Output reference to vector $x$ reference
- $E$ Error indicator bool
MB_DSWAP – Interchanges two vectors

Block Symbol 

License: STANDARD

Function Description

The output references yX and yY are always set to the corresponding input references uX and uY. If HLD = on then nothing is computed otherwise the BLAS function DSWAP is called internally:

DSWAP(N, uX, INCX, uY, INCY);

where the values N, INCX and INCY are set in the following way:

- If the input n > 0 then N is set to n else N is set to the current number of the vector or matrix elements CNTX referenced by uX.
- If the input incx ≠ 0 then INCX is set to incx else INCX is set to 1.
- If the input incy ≠ 0 then INCY is set to incy else INCY is set to 1.

The error flag E is set to on if:

- the reference uX or uY is not defined (i.e. input uX or uY is not connected),
- n < 0,
- (N – 1) * |INCX| + 1 > CNTX,
- (N – 1) * |INCY| + 1 > CNTY, where CNTY is a number of the vector or matrix elements referenced by uY.


Inputs

uX Input reference to vector x reference
uY Input reference to vector y reference
n Number of processed vector elements long
incx Index increment of vector x long
incy Index increment of vector y long
HLD Hold bool
**Outputs**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yX</td>
<td>Output reference to vector x</td>
<td>reference</td>
</tr>
<tr>
<td>yY</td>
<td>Output reference to vector y</td>
<td>reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
MB_DTRMM – Performs \( B := \alpha \cdot \text{op}(A) \cdot B \) or \( B := \alpha \cdot B \cdot \text{op}(A) \), where \( \text{op}(X) = X \) or \( \text{op}(X) = X^T \) for triangular matrix \( A \)

**Block Symbol**

![Block Symbol](image)

**Licence:** STANDARD

**Function Description**

The output references \( yA \) and \( yB \) are always set to the corresponding input references \( uA \) and \( uB \). If \( \text{HLD} = \text{on} \) then nothing is computed otherwise the BLAS function \( \text{DTRMM} \) is called internally:

\[
\text{DTRMM}(s\text{RSIDE}, s\text{LUPLO}, s\text{TRANSA}, s\text{NDIAG}, M, N, \alpha, uA, LDA, uB, LDB);
\]

where parameters of \( \text{DTRMM} \) are set in the following way:

- If \( \text{RSIDE} = \text{on} \) then the string \( s\text{RSIDE} \) is set to "R" else it is set to "L".
- If \( \text{LUPLO} = \text{on} \) then the string \( s\text{LUPLO} \) is set to "L" else it is set to "U".
- Integer input \( \text{transa} \) is mapped to the string \( s\text{TRANSA} \): \( 0, 1 \rightarrow \text{"N"}, 2 \rightarrow \text{"T"} \) and \( 3 \rightarrow \text{"C"} \).
- If \( \text{NDIAG} = \text{on} \) then the string \( s\text{NDIAG} \) is set to "N" else it is set to "U".
- \( M \) is number of rows of the matrix referenced by \( uB \).
- \( N \) is number of columns of the matrix referenced by \( uB \).
- \( LDA \) and \( LDB \) are leading dimensions of matrices referenced by \( uA \) and \( uB \).

The error flag \( E \) is set to \( \text{on} \) if:

- the reference \( uA \) or \( uB \) is not defined (i.e. input \( uA \) or \( uB \) is not connected),
- \( \text{transa} \) is less than 0 or greater than 3,
- matrix referenced by \( uA \) is not square or is not compatible with the matrix referenced by \( uB \),
- the call of the function \( \text{DTRMM} \) returns error using the function \( \text{XERBLA} \), see the system log.

**Inputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>uB</td>
<td>Input reference to matrix B</td>
<td>reference</td>
</tr>
<tr>
<td>RSIDE</td>
<td>Operation is applied from right side</td>
<td>bool</td>
</tr>
<tr>
<td>LUPL0</td>
<td>Matrix A is a lower triangular matrix</td>
<td>bool</td>
</tr>
<tr>
<td>transa</td>
<td>Transposition of matrix A</td>
<td>⌈0 3⌉ long</td>
</tr>
<tr>
<td>NDIAG</td>
<td>Matrix A is not assumed to be unit triangular</td>
<td>bool</td>
</tr>
<tr>
<td>alpha</td>
<td>Scalar coefficient alpha</td>
<td>double</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>yB</td>
<td>Output reference to matrix B</td>
<td>reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
MB_DTRMV – Performs \(x := A \cdot x\) or \(x := A^T \cdot x\) for triangular matrix \(A\)

Function Description

The output references \(yA\) and \(yX\) are always set to the corresponding input references \(uA\) and \(uX\). If \(HLD = \text{on}\) then nothing is computed otherwise the BLAS function DTRMV is called internally:

\[
\text{DTRMV}(sLUPLO, sTRANS, sNDIAG, N, uA, LDA, uX, INCX);
\]

where parameters of DTRMV are set in the following way:

- If \(LUPLO = \text{on}\) then the string \(sLUPLO\) is set to "L" else it is set to "U".
- Integer input \(\text{trans}\) is mapped to the string \(sTRANS\): \(\{0, 1\} \rightarrow \text{"N"}, \{2\} \rightarrow \text{"T"} \) and \(\{3\} \rightarrow \text{"C"}\).
- If \(NDIAG = \text{on}\) then the string \(sNDIAG\) is set to "N" else it is set to "U".
- \(N\) is number of rows and columns of the square matrix referenced by \(uA\).
- \(LDA\) is the leading dimension of matrix referenced by \(uA\).
- If the input \(\text{incx} \neq 0\) then \(\text{INCX}\) is set to \(\text{incx}\) else \(\text{INCX}\) is set to 1.

The error flag \(E\) is set to \(\text{on}\) if:

- the reference \(uA\) or \(uX\) is not defined (i.e. input \(uA\) or \(uX\) is not connected),
- \(\text{trans}\) is less than 0 or greater than 3,
- matrix referenced by \(uA\) is not square,
- \((N - 1) \cdot |\text{INCX}| + 1 > \text{CNTX}\), where \(\text{CNTX}\) is a number of the vector or matrix elements referenced by \(uX\).
- the call of the function DTRMV returns error using the function XERBLA, see the system log.
CHAPTER 14. MATRIX – BLOCKS FOR MATRIX AND VECTOR OPERATIONS

Inputs

- uA: Input reference to matrix A
- uX: Input reference to vector x
- LUPL0: Matrix A is a lower triangular matrix
- trans: Transposition of the input matrix
- NDIAG: Matrix A is not assumed to be unit triangular
- incx: Index increment of vector x
- HLD: Hold

Outputs

- yA: Output reference to matrix A
- yX: Output reference to vector x
- E: Error indicator
MB_DTRSV – Solves one of the system of equations $A^T \times x = b$ or $A^\top x = b$ for triangular matrix $A$

Block Symbol

Licence: STANDARD

Function Description

The output references $y_A$ and $y_X$ are always set to the corresponding input references $u_A$ and $u_X$. If $HLD = \text{on}$ then nothing is computed otherwise the BLAS function DTRSV is called internally:

$$\text{DTRSV}(s\text{LUPLO}, s\text{TRANS}, s\text{NDIAG}, N, u_A, LDA, u_X, INCX);$$

where parameters of DTRSV are set in the following way:

- If $\text{LUPLO} = \text{on}$ then the string $s\text{LUPLO}$ is set to "L" else it is set to "U".
- Integer input $\text{trans}$ is mapped to the string $s\text{TRANS}$: $\{0, 1\} \rightarrow "N", \{2\} \rightarrow "T"$ and $\{3\} \rightarrow "C"$.
- If $\text{NDIAG} = \text{on}$ then the string $s\text{NDIAG}$ is set to "N" else it is set to "U".
- $N$ is number of rows and columns of the square matrix referenced by $u_A$.
- $LDA$ is the leading dimension of matrix referenced by $u_A$.
- If the input $\text{incx} \neq 0$ then $\text{INCX}$ is set to $\text{incx}$ else $\text{INCX}$ is set to 1.

The error flag $E$ is set to $\text{on}$ if:

- the reference $u_A$ or $u_X$ is not defined (i.e. input $u_A$ or $u_X$ is not connected),
- $\text{trans}$ is less than 0 or greater than 3,
- matrix referenced by $u_A$ is not square,
- $(N - 1) \times |\text{INCX}| + 1 > \text{CNTX}$, where $\text{CNTX}$ is a number of the vector or matrix elements referenced by $u_X$.
- the call of the function DTRMV returns error using the function XERBLA, see the system log.
## Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>uX</td>
<td>Input reference to vector x</td>
<td>reference</td>
</tr>
<tr>
<td>LUPL0</td>
<td>Matrix A is a lower triangular matrix</td>
<td>bool</td>
</tr>
<tr>
<td>trans</td>
<td>Transposition of the input matrix</td>
<td>long</td>
</tr>
<tr>
<td>NDIAG</td>
<td>Matrix A is not assumed to be unit triangular</td>
<td>bool</td>
</tr>
<tr>
<td>incx</td>
<td>Index increment of vector x</td>
<td>long</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>yX</td>
<td>Output reference to vector x</td>
<td>reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
ML_DGEBAK – Backward transformation to ML_DGEBAL of left or right eigenvectors

Function Description

The output references ySCALE and yV are always set to the corresponding input references uSCALE and uV. If HLD = on then nothing is computed otherwise the LAPACK function DGEBAK is called internally:

\[ \text{DGEBAK}(sJOB, sRSIDE, N, ilo, IHI, uSCALE, M, uV, LDV, info); \]

where parameters of DGEBAK are set in the following way:

- Integer input job is mapped to the string sJOB: \( \{0,1\} \rightarrow "N", \{2\} \rightarrow "P", \{3\} \rightarrow "S" \) and \( \{4\} \rightarrow "B". \)
- If RSIDE = on then the string sRSIDE is set to "R" else it is set to "L".
- \( N \) is number of elements of the vector referenced by uSCALE.
- If the input ihi \( \neq 0 \) then IHI is set to ihi else IHI is set to \( N - 1 \).
- \( M \) is number of columns of the matrix referenced by uV.
- LDV is the leading dimension of the matrix referenced by uV.
- info is return code from the function DGEBAK.

The error flag E is set to on if:

- the reference uSCALE or uV is not defined (i.e. input uSCALE or uV is not connected),
- the call of the function DGEBAK returns error using the function XERBLA, see the return code info and system log.

Emphasize that the indices ilo and ihi start from zero unlike FORTRAN version where they start from one. See LAPACK documentation [8] for more details.
Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uSCALE</td>
<td>Input reference to vector SCALE</td>
<td>reference</td>
</tr>
<tr>
<td>uV</td>
<td>Reference to matrix of right or left eigenvectors to be transformed</td>
<td>reference</td>
</tr>
<tr>
<td>job</td>
<td>Type of backward transformation required</td>
<td>↓0 ↑4 long</td>
</tr>
<tr>
<td>RSIDE</td>
<td>Operation is applied from right side</td>
<td>bool</td>
</tr>
<tr>
<td>ilo</td>
<td>Zero based low row and column index of working submatrix</td>
<td>long</td>
</tr>
<tr>
<td>ihi</td>
<td>Zero based high row and column index of working submatrix</td>
<td>long</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ySCALE</td>
<td>Output reference to vector SCALE</td>
<td>reference</td>
</tr>
<tr>
<td>yV</td>
<td>Reference to matrix of transformed right or left eigenvectors</td>
<td>reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
<tr>
<td>info</td>
<td>LAPACK function result info. If info = -i, the i=th argument had an illegal value</td>
<td>long</td>
</tr>
</tbody>
</table>
ML_DGEBAL – Balancing of a general real matrix

Block Symbol

Function Description

The output references $yA$ and $ySCALE$ are always set to the corresponding input references $uA$ and $uSCALE$. If HLD = on then nothing is computed otherwise the LAPACK function DGEBAL is called internally:

$$DGEBAL(sJOB, N, uA, LDA, ilo, ihi, uSCALE, info);$$

where parameters of DGEBAL are set in the following way:

- Integer input $job$ is mapped to the string $sJOB$: $\{0, 1\} \rightarrow "N", \{2\} \rightarrow "P", \{3\} \rightarrow "S"$ and $\{4\} \rightarrow "B"$.
- $N$ is number of columns of the square matrix referenced by $uA$.
- $LDA$ is the leading dimension of the matrix referenced by $uA$.
- $ilo$ and $ihi$ are returned low and high row and column indices of the balanced submatrix of the matrix referenced by $uA$.
- $info$ is return code from the function DGEBAL.

The error flag $E$ is set to on if:

- the reference $uA$ or $uSCALE$ is not defined (i.e. input $uA$ or $uSCALE$ is not connected),
- the reference $uA$ or $uSCALE$ is not square,
- number of elements of the vector referenced by $uSCALE$ is less than $N$.
- the call of the function DGEBAL returns error using the function XERBLA, see the return code $info$ and system log.

Emphasize that the indices $ilo$ and $ihi$ start from zero unlike FORTRAN version where they start from one. See LAPACK documentation [8] for more details.
### Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>uSCALE</td>
<td>Input reference to vector SCALE</td>
<td>reference</td>
</tr>
<tr>
<td>job</td>
<td>Specifies the operations to be performed on matrix A</td>
<td>$\downarrow 0 \uparrow 4$ long</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>ySCALE</td>
<td>Output reference to vector SCALE</td>
<td>reference</td>
</tr>
<tr>
<td>ilo</td>
<td>Zero based low row and column index of working submatrix</td>
<td>long</td>
</tr>
<tr>
<td>ihi</td>
<td>Zero based high row and column index of working submatrix</td>
<td>long</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
<tr>
<td>info</td>
<td>LAPACK function result info. If info = -i, the i-th argument had an illegal value</td>
<td>long</td>
</tr>
</tbody>
</table>
ML_DGEBRD — Reduces a general real matrix to bidiagonal form by an orthogonal transformation

Block Symbol

Licence: MATRIX

Function Description

The output references \( yA \), \( yD \), \( yE \), \( yTAUQ \), \( yTAUP \) and \( yWORK \) are always set to the corresponding input references \( uA \), \( uD \), \( uE \), \( uTAUQ \), \( uTAUP \) and \( uWORK \). If \( HLD = \text{on} \) then nothing is computed otherwise the LAPACK function \( \text{DGEBRD} \) is called internally:

\[
\text{DGEBRD}(M, N, uA, LDA, uD, uE, uTAUQ, uTAUP, uWORK, info);
\]

where parameters of \( \text{DGEBRD} \) are set in the following way:

- \( M \) is number of rows of the matrix referenced by \( uA \).
- \( N \) is number of columns of the matrix referenced by \( uA \).
- \( LDA \) is the leading dimension of the matrix referenced by \( uA \).
- \( info \) is return code from the function \( \text{DGEBRD} \).

The error flag \( E \) is set to \text{on} if:

- the reference \( uA \) or \( uD \) or \( uE \) or \( uTAUQ \) or \( uTAUP \) or \( uWORK \) is not defined (i.e. input \( uA \) or \( uD \) or \( uE \) or \( uTAUQ \) or \( uTAUP \) or \( uWORK \) is not connected),
- number of elements of any vector referenced by \( uD \), \( uTAUQ \) and \( uTAUP \) is less than \( \text{MINMN} \), where \( \text{MINMN} \) is minimum from \( M \) and \( N \),
- number of elements of the vector referenced by \( uE \) is less than \( \text{MINMN} - 1 \),
- the call of the function \( \text{DGEBRD} \) returns error using the function \( \text{XERBLA} \), see the return code \( info \) and system log.

See LAPACK documentation [8] for more details.
Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
</tr>
<tr>
<td>uD</td>
<td>Diagonal elements of the bidiagonal matrix B</td>
</tr>
<tr>
<td>uE</td>
<td>Off-diagonal elements of the bidiagonal matrix B</td>
</tr>
<tr>
<td>uTAUQ</td>
<td>Reference to a vector of scalar factors of the elementary reflectors which represent the orthogonal matrix Q</td>
</tr>
<tr>
<td>uTAUP</td>
<td>Reference to a vector of scalar factors of the elementary reflectors which represent the orthogonal matrix P</td>
</tr>
<tr>
<td>uWORK</td>
<td>Input reference to working vector WORK</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
</tr>
<tr>
<td>yD</td>
<td>Output reference to D</td>
</tr>
<tr>
<td>yE</td>
<td>Output reference to E</td>
</tr>
<tr>
<td>yTAUQ</td>
<td>Output reference to TAUQ</td>
</tr>
<tr>
<td>yTAUP</td>
<td>Output reference to TAUP</td>
</tr>
<tr>
<td>yWORK</td>
<td>Output reference to working vector WORK</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
</tr>
<tr>
<td>info</td>
<td>LAPACK function result info. If info = -i, the i-th argument had an illegal value</td>
</tr>
</tbody>
</table>
**ML_DGECON** – Estimates the reciprocal of the condition number of a general real matrix

**Block Symbol**  

![Block Symbol](image)

**Licence:** MATRIX

**Function Description**

The output references \( yA \), \( yWORK \) and \( yIWORK \) are always set to the corresponding input references \( uA \), \( uWORK \) and \( uIWORK \). If \( HLD = \text{on} \) then nothing is computed otherwise the LAPACK function \texttt{DGECON} is called internally:

\[
\text{DGECON}(\text{sINORM}, N, uA, LDA, anorm, rcond, uWORK, uIWORK, info);
\]

where parameters of \texttt{DGECON} are set in the following way:

- If \( \text{INORM} = \text{on} \) then the string \( \text{sINORM} \) is set to "I" else it is set to "1".
- \( N \) is number of columns of the matrix referenced by \( uA \).
- \( LDA \) is the leading dimension of the matrix referenced by \( uA \).
- \( rcond \) is returned reciprocal value of the condition number of the matrix referenced by \( uA \).
- \( info \) is return code from the function \texttt{DGECON}.

The error flag \( E \) is set to \text{on} if:

- the reference \( uA \) or \( uWORK \) or \( uIWORK \) is not defined (i.e. input \( uA \) or \( uWORK \) or \( uIWORK \) is not connected),
- the matrix referenced by \( uA \) is not square,
- number of elements of the vector referenced by \( uWORK \) is less than \( 4 \times N \),
- number of elements of the integer vector referenced by \( uIWORK \) is less than \( N \),
- the call of the function \texttt{DGECON} returns error using the function \texttt{XERBLA}, see the return code \( info \) and system log.

See LAPACK documentation [8] for more details.
Inputs

<table>
<thead>
<tr>
<th></th>
<th>(uA) Input reference to matrix A</th>
<th>(uWORK) Input reference to working vector WORK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(uIWORK) Input reference to integer working vector WORK</td>
<td>(INORM) Use Infinity-norm</td>
</tr>
<tr>
<td></td>
<td>(HLD) Hold</td>
<td>(anorm) Norm of the original matrix A</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th></th>
<th>(yA) Output reference to matrix A</th>
<th>(yWORK) Output reference to working vector WORK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(yIWORK) Output reference to integer working vector WORK</td>
<td>(rcond) The reciprocal of the condition number of the matrix A</td>
</tr>
<tr>
<td></td>
<td>(E) Error indicator</td>
<td>(info) LAPACK function result info. If info = -i, the i=th argument had an illegal value</td>
</tr>
<tr>
<td></td>
<td>(info) LAPACK function result info. If info = -i, the i=th argument had an illegal value</td>
<td>(info) LAPACK function result info. If info = -i, the i=th argument had an illegal value</td>
</tr>
</tbody>
</table>
ML_DGEES – Computes the eigenvalues, the Schur form, and, optionally, the matrix of Schur vectors

Function Description

The output references yA, yWR, yWI, yVS, yWORK and yBWORK are always set to the corresponding input references uA, uWR, uWI, uVS, uWORK and uBWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGEES is called internally:

\[
\text{DGEES(sJOBVS, sSORT, SELECT, N, uA, LDA, sdim, uWR, uWI, uVS, LDVS, uWORK, LWORK, uBWORK, info);}
\]

where parameters of DGEES are set in the following way:

- If JOBVS = on then the string sJOBVS is set to "V" else it is set to "N".
- If SORT = on then the string sSORT is set to "S" else it is set to "N".
- SELECT is the reference to Boolean eigenvalues sorting function which in this function block returns always true (i.e. on).
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.
- sdim is returned number of eigenvalues for which the function SELECT is true.
- LDVS is the leading dimension of the matrix referenced by uVS.
- LWORK is number of elements in the vector referenced by uWORK.
- info is return code from the function DGEES.

The error flag E is set to on if:

- the reference uA or uWR or uWI or uVS or uWORK or uBWORK is not defined (i.e. input uA or uWR or uWI or uVS or uWORK or uBWORK is not connected),
the matrix referenced by \texttt{uA} is not square,

- number of elements of any vector referenced by \texttt{uWR}, \texttt{uWI} and \texttt{uBWORK} is less than \( N \),

- number of columns of the matrix referenced by \texttt{uVS} is not equal to \( N \),

- the call of the function \texttt{DGEES} returns error using the function \texttt{XERBLA}, see the return code \texttt{info} and system log.

See LAPACK documentation \[8\] for more details.

**Inputs**

| \texttt{uA}   | Input reference to matrix A | reference |
| \texttt{uWR}  | Input reference to vector of real parts of eigenvalues | reference |
| \texttt{uWI}  | Input reference to vector of imaginary parts of eigenvalues | reference |
| \texttt{uVS}  | Input reference to orthogonal matrix of Schur vectors | reference |
| \texttt{uWORK} | Input reference to working vector WORK | reference |
| \texttt{uBWORK} | Input reference to Boolean working vector WORK | reference |
| \texttt{JOBVS} | If true then Schur vectors are computed | bool |
| \texttt{SORT}  | If true then eigenvalues are sorted | bool |
| \texttt{HLD}   | Hold | bool |

**Outputs**

| \texttt{yA}   | Output reference to matrix A | reference |
| \texttt{yWR}  | Output reference to vector of real parts of eigenvalues | reference |
| \texttt{yWI}  | Output reference to vector of imaginary parts of eigenvalues | reference |
| \texttt{yVS}  | Output reference to VS | reference |
| \texttt{yWORK} | Output reference to working vector WORK | reference |
| \texttt{yBWORK} | Output reference to Boolean working vector WORK | reference |
| \texttt{sdim} | If SORT then number of eigenvalues for which SELECT is true else 0 | long |
| \texttt{E}    | Error indicator | bool |
| \texttt{info} | LAPACK function result info. If info = -i, the i-th argument had an illegal value | long |
ML_DGEEV – Computes the eigenvalues and, optionally, the left and/or right eigenvectors

Block Symbol License: MATRIX

Function Description

The output references yA, yWR, yWI, yVL, yVR and yWORK are always set to the corresponding input references uA, uWR, uWI, uVL, uVR and uWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGEEV is called internally:

```
DGEEV(sJOBVL, sJOBVR, N, uA, LDA, uWR, uWI, uVL, LDVL, uVR, LDVR, 
    uWORK, LWORK, info);
```

where parameters of DGEEV are set in the following way:

- If JOBVL = on then the string sJOBVL is set to "V" else it is set to "N".
- If JOBVR = on then the string sJOBVR is set to "V" else it is set to "N".
- N is number of columns of the matrix referenced by uA.
- LDA, LDVL and LDVR are leading dimensions of the matrices referenced by uA, uVL and uVR.
- LWORK is number of elements of the vector referenced by uWORK.
- info is return code from the function DGEEV.

The error flag E is set to on if:

- the reference uA or uWR or uWI or uVL or uVR or uWORK is not defined (i.e. input uA or uWR or uWI or uVL or uVR or uWORK is not connected),
- the matrix referenced by uA is not square,
- number of elements of vectors referenced by uWR or uWI is less than N,
- number of columns of matrices referenced by uVL or uVR is not equal to N,
the call of the function \texttt{DGEEV} returns error using the function \texttt{XERBLA}, see the return code \texttt{info} and system log.

See LAPACK documentation \cite{8} for more details.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{uA}</td>
<td>Input reference to matrix A</td>
</tr>
<tr>
<td>\texttt{uWR}</td>
<td>Input reference to vector of real parts of eigenvalues</td>
</tr>
<tr>
<td>\texttt{uWI}</td>
<td>Input reference to vector of imaginary parts of eigenvalues</td>
</tr>
<tr>
<td>\texttt{uVL}</td>
<td>Input reference to matrix of left eigenvectors</td>
</tr>
<tr>
<td>\texttt{uVR}</td>
<td>Input reference to matrix of right eigenvectors</td>
</tr>
<tr>
<td>\texttt{uWORK}</td>
<td>Input reference to working vector WORK</td>
</tr>
<tr>
<td>\texttt{JOBVL}</td>
<td>If true then left eigenvectors are computed</td>
</tr>
<tr>
<td>\texttt{JOBVR}</td>
<td>If true then right eigenvectors are computed</td>
</tr>
<tr>
<td>\texttt{HLD}</td>
<td>Hold</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{yA}</td>
<td>Output reference to matrix A</td>
</tr>
<tr>
<td>\texttt{yWR}</td>
<td>Output reference to vector of real parts of eigenvalues</td>
</tr>
<tr>
<td>\texttt{yWI}</td>
<td>Output reference to vector of imaginary parts of eigenvalues</td>
</tr>
<tr>
<td>\texttt{yVL}</td>
<td>Output reference to VL</td>
</tr>
<tr>
<td>\texttt{yVR}</td>
<td>Output reference to VR</td>
</tr>
<tr>
<td>\texttt{yWORK}</td>
<td>Output reference to working vector WORK</td>
</tr>
<tr>
<td>\texttt{E}</td>
<td>Error indicator</td>
</tr>
<tr>
<td>\texttt{info}</td>
<td>LAPACK function result info. If info = -i, the i-th argument had an illegal value</td>
</tr>
</tbody>
</table>

\texttt{info} \quad \text{LAPACK function result info. If info = -i, the i-th argument had an illegal value}
ML_DGEHRD – Reduces a real general matrix A to upper Hessenberg form

Block Symbol

Licence: MATRIX

Function Description

The output references yA, yTAU and yWORK are always set to the corresponding input references uA, uTAU and uWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGEHRD is called internally:

DGEHRD(N, ilo, IHI, uA, LDA, uTAU, uWORK, LWORK, info);

where parameters of DGEHRD are set in the following way:

- N is number of columns of the square matrix referenced by uA.
- If the input ihi ≠ 0 then IHI is set to ihi else IHI is set to N − 1.
- LDA is the leading dimension of the matrix referenced by uA.
- LWORK is number of elements of the vector referenced by uWORK.
- info is return code from the function DGEHRD.

The error flag E is set to on if:

- the reference uA or uTAU or uWORK is not defined (i.e. input uA or uTAU or uWORK is not connected),
- matrix referenced by uA is not square,
- number of elements of the vector referenced by uTAU is less than N − 1.
- the call of the function DGEHRD returns error using the function XERBLA, see the return code info and system log.

Emphasize that the indices ilo and ihi start from zero unlike FORTRAN version where they start from one. See LAPACK documentation [8] for more details.
Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>uTAU</td>
<td>Input reference to vector of scalar factors of the elementary reflectors</td>
<td>reference</td>
</tr>
<tr>
<td>uWORK</td>
<td>Input reference to working vector WORK</td>
<td>reference</td>
</tr>
<tr>
<td>ilo</td>
<td>Zero based low row and column index of working submatrix</td>
<td>long</td>
</tr>
<tr>
<td>ihi</td>
<td>Zero based high row and column index of working submatrix</td>
<td>long</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>yTAU</td>
<td>Output reference to vector of scalar factors of the elementary reflectors</td>
<td>reference</td>
</tr>
<tr>
<td>yWORK</td>
<td>Output reference to working vector WORK</td>
<td>reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
<tr>
<td>info</td>
<td>LAPACK function result info. If info = -i, the i-th argument had an illegal value</td>
<td>long</td>
</tr>
</tbody>
</table>
ML_DGELQF — Computes an LQ factorization of a real M-by-N matrix A

Function Description

The output references yA, yTAU and yWORK are always set to the corresponding input references uA, uTAU and uWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGELQF is called internally:

DGELQF(M, N, uA, LDA, uTAU, uWORK, LWORK, info);

where parameters of DGELQF are set in the following way:

- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.
- LWORK is number of elements of the vector referenced by uWORK.
- info is return code from the function DGELQF.

The error flag E is set to on if:

- the reference uA or uTAU or uWORK is not defined (i.e. input uA or uTAU or uWORK is not connected),
- number of elements of the vector referenced by uTAU is less than the minimum of number of rows and number of columns of the matrix referenced by uA.
- the call of the function DGELQF returns error using the function XERBLA, see the return code info and system log.

See LAPACK documentation [8] for more details.

Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
</tr>
<tr>
<td>Input</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>uTAU</td>
<td>Input reference to vector of scalar factors of the elementary reflectors</td>
</tr>
<tr>
<td>uWORK</td>
<td>Input reference to working vector WORK</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>yTAU</td>
<td>Output reference to vector of scalar factors of the elementary reflectors</td>
<td>reference</td>
</tr>
<tr>
<td>yWORK</td>
<td>Output reference to working vector WORK</td>
<td>reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
<tr>
<td>info</td>
<td>LAPACK function result info. If info = -i, the i-th argument had an illegal value</td>
<td>long</td>
</tr>
</tbody>
</table>
CHAPTER 14. MATRIX – BLOCKS FOR MATRIX AND VECTOR OPERATIONS

ML_DGELSD – Computes the minimum-norm solution to a real linear least squares problem

Block Symbol

Licence: MATRIX

Function Description

The output references yA, yB, yS, yWORK and yIWORK are always set to the corresponding input references uA, uB, uS, uWORK and uIWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGELSD is called internally:

\[
\text{DGELSD}(M, N, NRHS, uA, LDA, uB, LDB, uS, rcond, irank, uWORK, LWORK, uIWORK, info);
\]

where parameters of DGELSD are set in the following way:

- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- NRHS is number of columns of the matrix referenced by uB.
- LDA and LDB are leading dimensions of the matrices referenced by uA and uB.
- irank is returned effective rank of the matrix referenced by uA.
- LWORK is number of elements in the vector referenced by uWORK.
- info is return code from the function DGELSD.

The error flag E is set to on if:

- the reference uA or uB or uS or uWORK or uIWORK is not defined (i.e. input uA or uB or uS or uWORK or uIWORK is not connected),
- the number of rows of the matrix referenced by uB is not equal to M,
- number of elements of any vector referenced by uS is less than the minimum of M and N,
• number of elements of the integer vector referenced by $u_IWORK$ is not sufficient (see details in the LAPACK documentation of the function $DGELSD$),

• the call of the function $DGELSD$ returns error using the function $XERBLA$, see the return code $info$ and system log.

See LAPACK documentation [8] for more details.

**Inputs**

| uA          | Input reference to matrix A                  | reference |
| uB          | Input reference to matrix B                  | reference |
| uS          | Input reference to vector of singular values  | reference |
| uWORK       | Input reference to working vector WORK       | reference |
| uIWORK      | Input reference to integer working vector WORK | reference |
| rcond       | Used to determine the effective rank of A    | double    |
| HLD         | Hold                                          | bool      |

**Outputs**

| yA          | Output reference to matrix A                  | reference |
| yB          | Output reference to matrix B                  | reference |
| yS          | Output reference to vector of singular values  | reference |
| yWORK       | Output reference to working vector WORK       | reference |
| yIWORK      | Output reference to integer working vector WORK | reference |
| irank       | Effective rank of A                           | long      |
| E           | Error indicator                               | bool      |
| info        | LAPACK function result info. If $info = -i$, the $i$-th argument had an illegal value | long      |
ML_DGEQRF – Computes an QR factorization of a real M-by-N matrix A

Block Symbol

Function Description

The output references yA, yTAU and yWORK are always set to the corresponding input references uA, uTAU and uWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGEQRF is called internally:

\[ \text{DGEQRF}(M, N, uA, LDA, uTAU, uWORK, LWORK, info); \]

where parameters of DGEQRF are set in the following way:

- \( M \) is number of rows of the matrix referenced by uA.
- \( N \) is number of columns of the matrix referenced by uA.
- \( LDA \) is the leading dimension of the matrix referenced by uA.
- \( LWORK \) is number of elements of the vector referenced by uWORK.
- \( info \) is return code from the function DGEQRF.

The error flag \( E \) is set to on if:

- the reference uA or uTAU or uWORK is not defined (i.e. input uA or uTAU or uWORK is not connected),
- number of elements of the vector referenced by uTAU is less than the minimum of number of rows and number of columns of the matrix referenced by uA.
- the call of the function DGEQRF returns error using the function XERBLA, see the return code info and system log.

See LAPACK documentation [8] for more details.

Inputs

- \( uA \) Input reference to matrix A

- \( uTAU \) Reference to vector TAU

- \( uWORK \) Reference to vector WORK

- \( HLD \) Reference to flag HLD

- \( info \) Reference to return code info

- \( E \) Reference to error flag E
uTAU: Input reference to vector of scalar factors of the elementary reflectors
uWORK: Input reference to working vector WORK
HLD: Hold

Outputs

yA: Output reference to matrix A
yTAU: Output reference to vector of scalar factors of the elementary reflectors
yWORK: Output reference to working vector WORK
E: Error indicator
info: LAPACK function result info. If info = -i, the i=th argument had an illegal value
ML_DGESDD – Computes the singular value decomposition (SVD) of a real M-by-N matrix A

Function Description

The output references yA, yS, yU, yVT, yWORK and yIWORK are always set to the corresponding input references uA, uS, uU, uVT, uWORK and uIWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGESDD is called internally:

\[
\text{DGESDD}(\text{sJOBZ, M, N, uA, LDA, uS, uU, LDU, uVT, LDVT, uWORK, LWORK, uIWORK, info});
\]

where parameters of DGESDD are set in the following way:

- Integer input \( \text{jobz} \) is mapped to the string \( \text{sJOBZ} \): \( \{0,1\} \rightarrow "A", \{2\} \rightarrow "S", \{3\} \rightarrow "O" \) and \( \{4\} \rightarrow "N" \).
- \( M \) is number of rows of the matrix referenced by \( uA \).
- \( N \) is number of columns of the matrix referenced by \( uA \).
- \( \text{LDA} \), \( \text{LDU} \) and \( \text{LDVT} \) are leading dimensions of the matrices referenced by \( uA \), \( uU \) and \( uVT \).
- \( \text{LWORK} \) is number of elements of the vector referenced by \( uWORK \).
- \( \text{info} \) is return code from the function DGESDD.

The error flag \( \text{E} \) is set to on if:

- the reference \( uA \) or \( uS \) or \( uU \) or \( uVT \) or \( uWORK \) or \( uIWORK \) is not defined (i.e. input \( uA \) or \( uS \) or \( uU \) or \( uVT \) or \( uWORK \) or \( uIWORK \) is not connected),
- number of elements of the vector referenced by \( uS \) is less than \( \text{MINMN} \), the minimum of number of rows and number of columns of the matrix referenced by \( uA \),
- number of elements of the integer vector referenced by \( uIWORK \) is less than \( 8*\text{MINMN} \),
the call of the function DGESDD returns error using the function XERBLA, see the return code info and system log.

See LAPACK documentation [8] for more details.

Inputs

uA  Input reference to matrix A  reference
uS  Input reference to vector of singular values  reference
uU  Input reference to matrix containing left singular vectors of A  reference
uVT Input reference to matrix containing right singular vectors of A  reference
uWORK Input reference to working vector WORK  reference
uIWORK Input reference to integer working vector WORK  reference
jobz Specifies options for computing  long
HLD  Hold  bool

Outputs

yA  Output reference to matrix A  reference
yS  Output reference to vector of singular values  reference
yU  Output reference to matrix containing left singular vectors of A  reference
yVT Output reference to matrix containing right singular vectors of A  reference
yWORK Output reference to working vector WORK  reference
yIWORK Output reference to integer working vector WORK  reference
E    Error indicator  bool
info LAPACK function result info. If info = -i, the i-th argument had an illegal value  long
### ML_DLACPY – Copies all or part of one matrix to another matrix

**Block Symbol**

![Block Symbol]

**Licence:** STANDARD

### Function Description

The output references $y_A$ and $y_B$ are always set to the corresponding input references $u_A$ and $u_B$. If $HLD = \text{on}$ then nothing is computed otherwise the LAPACK function DLACPY is called internally:

$$DLACPY(sUPLO, M, N, uA, LDA, uB, LDA);$$

where parameters of DLACPY are set in the following way:

- Integer input $u_{\text{pl}}$ is mapped to the string $s_{\text{pl}}$: $\{0, 1\} \rightarrow "A", \{2\} \rightarrow "U"$ and $\{3\} \rightarrow "L"$.
- $M$ is number of rows of the matrix referenced by $u_A$.
- $N$ is number of columns of the matrix referenced by $u_A$.
- $LDA$ is the leading dimension of the matrix referenced by $u_A$.

The number of rows of the matrix referenced by $u_B$ is set to $M$ and the leading dimension of the matrix referenced by $u_B$ is set to $LDA$.

The error flag $E$ is set to $\text{on}$ if:

- the reference $u_A$ or $u_B$ is not defined (i.e. input $u_A$ or $u_B$ is not connected),
- the allocated number of elements of the matrix referenced by $u_A$ is different from the allocated number of elements of the matrix referenced by $u_B$.

See LAPACK documentation [8] for more details.

### Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_A$</td>
<td>Input reference to matrix A</td>
</tr>
<tr>
<td>$u_B$</td>
<td>Input reference to matrix B</td>
</tr>
<tr>
<td>$u_{\text{pl}}$</td>
<td>Part of the matrix to be copied</td>
</tr>
<tr>
<td>$HLD$</td>
<td>Hold</td>
</tr>
</tbody>
</table>

- $s_{\text{pl}}$: All
- $u_{\text{pl}}$: All
- $LDA_{\text{pl}}$: Upper
- $LDA_{\text{pl}}$: Lower

- $u_{\text{pl}}$ is long

- $HLD$ is bool
### Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_A$</td>
<td>Output reference to matrix A</td>
<td></td>
</tr>
<tr>
<td>$y_B$</td>
<td>Output reference to matrix B</td>
<td></td>
</tr>
<tr>
<td>$E$</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
ML_DLANGE – Computes one of the matrix norms of a general matrix

Input references yA and yWORK are always set to the corresponding input references uA and uWORK. If HLD = on then nothing is computed otherwise the LAPACK function DLANGE is called internally:

\[ \text{value} = \text{DLANGE} (\text{sNORM}, M, N, uA, LDA, uWORK); \]

where parameters of DLACPY are set in the following way:

- Integer input norm is mapped to the string sNORM: \{0, 1\} \rightarrow "F" (Frobenius norm), \{2\} \rightarrow "M" (\max (\abs (A(i,j))))), \{3\} \rightarrow "1" (one norm) and \{4\} \rightarrow "I" (infinity norm).
- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.
- uWORK is the working vector of dimension LWORK \geq M. uWORK is used only for infinity norm, otherwise it is not referenced.

The error flag E is set to on if:

- the reference uA is not defined (i.e. input uA is not connected),
- the reference uWORK is not defined for norm = 4 (i.e. input uWORK is not connected).

See LAPACK documentation [8] for more details.

Use the block MB_DNRM2 for computation of Frobenius norm of a vector.

Inputs

<table>
<thead>
<tr>
<th>uA</th>
<th>Input reference to matrix A</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uWORK</td>
<td>Input reference to working vector WORK</td>
<td>reference</td>
</tr>
<tr>
<td>norm</td>
<td>The selected matrix norm</td>
<td>\downarrow \uparrow 4 long</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>yWORK</td>
<td>Output reference to working vector WORK</td>
<td>reference</td>
</tr>
<tr>
<td>value</td>
<td>Return value of the function</td>
<td>double</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
ML_DLASET – Initializes the off-diagonal elements and the diagonal elements of a matrix to given values

Block Symbol Licence: **STANDARD**

Function Description

The output reference \( yA \) is always set to the corresponding input references \( uA \). If \( HLD = \text{on} \) then nothing is computed otherwise the LAPACK function \( \text{DLASET} \) is called internally:

\[
\text{DLASET}(s\text{UPLO}, M, N, \alpha, \beta, uA, LDA);
\]

where parameters of \( \text{DLACPY} \) are set in the following way:

- Integer input \( \text{uplo} \) is mapped to the string \( s\text{UPLO} \): \( \{0, 1\} \rightarrow "A", \{2\} \rightarrow "U" \) and \( \{3\} \rightarrow "L" \).
- \( M \) is number of rows of the matrix referenced by \( uA \).
- \( N \) is number of columns of the matrix referenced by \( uA \).
- \( \text{LDA} \) is the leading dimension of the matrix referenced by \( uA \).

The error flag \( E \) is set to \( \text{on} \) if:

- the reference \( uA \) is not defined (i.e. input \( uA \) is not connected),

See LAPACK documentation [8] for more details.

Inputs

- **\( uA \)** Input reference to matrix A
- **\( \text{uplo} \)** Part of the matrix to be set
  - 0 ..... All
  - 1 ..... All
  - 2 ..... Upper
  - 3 ..... Lower
- **\( \alpha \)** Scalar coefficient alpha
- **\( \beta \)** Scalar coefficient beta
- **\( \text{HLD} \)** Hold

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( uA )</td>
<td>Input reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>( \text{uplo} )</td>
<td>Part of the matrix to be set</td>
<td>long</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>Scalar coefficient alpha</td>
<td>double</td>
</tr>
<tr>
<td>( \beta )</td>
<td>Scalar coefficient beta</td>
<td>double</td>
</tr>
<tr>
<td>( \text{HLD} )</td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_A$</td>
<td>Output reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>$E$</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
ML_DTRSYL – Solves the real Sylvester matrix equation for quasi-triangular matrices A and B

Block Symbol Licence: MATRIX

Function Description
The output references yA, yB and yC are always set to the corresponding input references uA, uB and uC. If HLD = on then nothing is computed otherwise the LAPACK function DTRSYL is called internally:

DTRSYL(sTRANA, sTRANB, M, N, uA, LDA, uB, LDB, uC, LDC, scale, info);

where parameters of DTRSYL are set in the following way:

- Integer inputs trana and tranb are mapped to strings sTRANA and sTRANB: \(0, 1 \rightarrow "N", 2 \rightarrow "T" \) and \(3 \rightarrow "C"\).
- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uB.
- LDA, LDB and LDC are leading dimensions of matrices referenced by uA, uB and uC.
- scale is returned scaling factor to avoid overflow.
- info is return code from the function DTRSYL.

The error flag E is set to on if:

- the reference uA or uB or uC is not defined (i.e. input uA or uB or uC is not connected),
- trana or tranb is less than 0 or greater than 3
- number of columns of the matrix referenced by uA is not equal to M
- number of rows of the matrix referenced by uB is not equal to N
- number of rows of the matrix referenced by uC is not equal to N or number of columns of this matrix is not equal to M.
• the call of the function \texttt{DTRSYL} returns error using the function \texttt{XERBLA}, see the system log.

See LAPACK documentation \cite{8} for more details.

\textbf{Inputs}

\begin{tabular}{lll}
\textbf{uA} & Input reference to matrix A & \textbf{reference}
\textbf{uB} & Input reference to matrix B & \textbf{reference}
\textbf{uC} & Input reference to matrix C & \textbf{reference}
\textbf{trana} & Transposition of matrix A & \downarrow0 \uparrow3 \textbf{long}
\textbf{tranb} & Transposition of matrix B & \downarrow0 \uparrow3 \textbf{long}
\textbf{isgn} & Sign in the equation (1 or -1) & \downarrow-1 \uparrow1 \textbf{long}
\textbf{HLD} & Hold & \textbf{bool}
\end{tabular}

\textbf{Outputs}

\begin{tabular}{lll}
\textbf{yA} & Output reference to matrix A & \textbf{reference}
\textbf{yB} & Output reference to matrix B & \textbf{reference}
\textbf{yC} & Output reference to matrix C & \textbf{reference}
\textbf{scale} & Scale & \textbf{double}
\textbf{E} & Error indicator & \textbf{bool}
\textbf{info} & LAPACK function result info. If info = -i, the i-th argument had an illegal value & \textbf{long}
\end{tabular}
**MX_AT – Get Matrix/Vector element**

**Block Symbol**

**Licence:** STANDARD

**Function Description**

The function block MX_AT returns the value (output value) of the matrix element at the i-th row and j-th column or the i-th vector element.

The output reference yMV is always set to the corresponding input reference uMV to the connected matrix/vector.

The error flag E is set to on if:

- the reference uMV is not defined (i.e. input uMV is not connected),
- the zero based row index i < 0 or i ≥ m, where m is the number of rows,
- the zero based column index j < 0 or j ≥ n, where n is the number of columns.

Note that j must be 0 for a vector.

**Inputs**

<table>
<thead>
<tr>
<th>uMV</th>
<th>Input reference to a matrix or vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Row index of the element</td>
</tr>
<tr>
<td>j</td>
<td>Column index of the element</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>yMV</th>
<th>Output reference to a matrix or vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Value of element at position (i,j)</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
</tr>
</tbody>
</table>
MX_ATSET – Set Matrix/Vector element

Block Symbol  Licence: STANDARD

Function Description

The function block MX_ATSET sets the value (input value) to the matrix element at the
i-th row and j-th column or to the i-th vector element.

The output reference yMV is always set to the corresponding input reference uMV to
the connected matrix/vector.

The error flag E is set to on if:

- the reference uMV is not defined (i.e. input uMV is not connected),
- the zero based row index \( i < 0 \) or \( i \geq m \), where \( m \) is the number of rows,
- the zero based column index \( j < 0 \) or \( j \geq n \), where \( n \) is the number of columns.

Note that \( j \) must be 0 for a vector.

Inputs

<table>
<thead>
<tr>
<th>uMV</th>
<th>Input reference to a matrix or vector reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Row index of the element ( \downarrow 0 ) long</td>
</tr>
<tr>
<td>j</td>
<td>Column index of the element ( \downarrow 0 ) long</td>
</tr>
<tr>
<td>value</td>
<td>Value which should be set to the element at position ((i,j)) long</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>yMV</th>
<th>Output reference to a matrix or vector reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Error indicator bool</td>
</tr>
</tbody>
</table>

**MX_CNADD — Add scalar to each Matrix/Vector element**

**Block Symbol**

![Block Symbol](image)

**Licence:** STANDARD

**Function Description**

The function block **MX_CNADD** adds the value of the input **alpha** to each matrix/vector element referenced by **uAX** and the result is stored to the matrix/vector referenced by **uBY**. If **HLD** = on then nothing is computed.

The output references **yAX** and **yBY** are always set to the corresponding input references **uAX** and **uBY**. The dimensions of the matrix/vector referenced by **uBY** are set to the dimensions of the matrix/vector referenced by **uAX** if they are different.

The error flag **E** is set to on if:

- the reference **uAX** of **uBY** is not defined (i.e. input **uAX** or **uBY** is not connected),
- the count of allocated elements of the matrix/vector referenced by **uAX** is different from the count of allocated elements of the matrix/vector referenced by **uBY**.

**Inputs**

- **uAX**: Input reference to the matrix A or vector X
- **uBY**: Input reference to the matrix B or vector Y
- **alpha**: Scalar coefficient alpha
- **HLD**: Hold

**Outputs**

- **yAX**: Output reference to the matrix A or vector X
- **yBY**: Output reference to the matrix B or vector Y
- **E**: Error indicator
**MX_CNMUL – Multiply a Matrix/Vector by a scalar**

**Block Symbol**

![Block Symbol]

**Licence:** STANDARD

**Function Description**

The function block MX_CNMUL multiplies each matrix/vector element referenced by uAX by the value of the input alpha and the result is stored to the matrix/vector referenced by uBY. If HLD = on then nothing is computed.

The output references yAX and yBY are always set to the corresponding input references uAX and uBY. The dimensions of the matrix/vector referenced by uBY are set to the dimensions of the matrix/vector referenced by uAX if they are different.

The error flag E is set to on if:

- the reference uAX of uBY is not defined (i.e. input uAX or uBY is not connected),
- the count of allocated elements of the matrix/vector referenced by uAX is different from the count of allocated elements of the matrix/vector referenced by uBY.

**Inputs**

- **uAX**: Input reference to the matrix A or vector X  
  - Reference
- **uBY**: Input reference to the matrix B or vector Y  
  - Reference
- **alpha**: Scalar coefficient alpha  
  - Double
- **HLD**: Hold  
  - Bool

**Outputs**

- **yAX**: Output reference to the matrix A or vector X  
  - Reference
- **yBY**: Output reference to the matrix B or vector Y  
  - Reference
- **E**: Error indicator  
  - Bool
MX_CTODPA – Discretizes continuous model given by (A,B) to (Ad,Bd) using Pade approximations

Block Symbol

Function Description

This function block discretizes a continuous state space model using Padé approximations of matrix exponential and its integral and scaling technique ([6]). The used technique is similar to method 3 Scaling and squaring described in [9].

The output references \( y_A, y_B, y_{Ad}, y_{Bd}, y_P, y_Q \) and \( y_R \) are always set to the corresponding input references \( u_A, u_B, u_{Ad}, u_{Bd}, u_P, u_Q \) and \( u_R \). If \( \text{HLD} = \text{on} \) then nothing is computed otherwise the function \( \text{mCtoD} \) is called internally:

\[
\text{mCtoD}(\text{nRes}, u_{Ad}, u_{Bd}, u_A, u_B, N, M, \text{is}, Ts, \text{eps}, u_P, u_Q, u_R);
\]

where parameters of \( \text{mCtoD} \) are set in the following way:

- \( \text{nRes} \) is return code from the function \( \text{mCtoD} \).
- \( N \) is number of rows of the square system matrix referenced by \( u_A \).
- \( M \) is number of columns of the input matrix referenced by \( u_B \).
- \( Ts \) is sampling period for the discretization, which is equal to sampling period of the task containing this function block.

The error flag \( E \) is set to \text{on} if:

- the reference \( u_A \) or \( u_B \) or \( u_{Ad} \) or \( u_{Bd} \) or \( u_P \) or \( u_Q \) or \( u_R \) is not defined (i.e. input \( u_A \) or \( u_B \) or \( u_{Ad} \) or \( u_{Bd} \) or \( u_P \) or \( u_Q \) or \( u_R \) is not connected),
- number of columns of the matrix referenced by \( u_A \) is not equal to \( N \),
- number of rows of the matrix referenced by \( u_B \) is not equal to \( N \),
- number of elements of any matrix referenced by \( u_{Ad}, u_P, u_Q \) or \( u_R \) is less than \( N \cdot N \),
- number of elements of the matrix referenced by \( u_{Bd} \) is less than \( N \cdot M \),
the return code nRes of the function mCtoD is not equal to zero.

Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>uB</td>
<td>Input reference to matrix B</td>
<td>reference</td>
</tr>
<tr>
<td>uAd</td>
<td>Input reference to discretized matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>uBd</td>
<td>Input reference to discretized matrix B</td>
<td>reference</td>
</tr>
<tr>
<td>uP</td>
<td>Input reference to a helper matrix</td>
<td>reference</td>
</tr>
<tr>
<td>uQ</td>
<td>Input reference to a helper matrix</td>
<td>reference</td>
</tr>
<tr>
<td>uR</td>
<td>Input reference to a helper matrix</td>
<td>reference</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>is</td>
<td>Pade approximation order</td>
<td>long</td>
</tr>
<tr>
<td>eps</td>
<td>Approximation accuracy</td>
<td>double</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>yB</td>
<td>Output reference to matrix B</td>
<td>reference</td>
</tr>
<tr>
<td>yAd</td>
<td>Output reference to discretized matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>yBd</td>
<td>Output reference to discretized matrix B</td>
<td>reference</td>
</tr>
<tr>
<td>yP</td>
<td>Output reference to a helper matrix</td>
<td>reference</td>
</tr>
<tr>
<td>yQ</td>
<td>Output reference to a helper matrix</td>
<td>reference</td>
</tr>
<tr>
<td>yR</td>
<td>Output reference to a helper matrix</td>
<td>reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
**MX_DIM – Matrix/Vector dimensions**

**Block Symbol**

![Block Symbol Image]

**Licence:** STANDARD

**Function Description**

The function block **MX_DIM** sets its outputs to the dimensions of the matrix or vector referenced by **uMV**.

The output reference **yMV** is always set to the corresponding input reference **uMV**. The error flag **E** is set to on if the reference **uMV** is not defined (i.e. input **uMV** is not connected).

**Input**

- **uMV**: Input reference to a matrix or vector

**Outputs**

- **yMV**: Output reference to a matrix or vector
- **m**: Number of matrix rows
- **n**: Number of matrix columns
- **ld**: Leading dimension (>= number of rows)
- **cnt**: Count of allocated elements (>= number of rows * number of columns)

Reference: long
**MX_DIMSET — Set Matrix/Vector dimensions**

Block Symbol  

Licence: **STANDARD**

**Function Description**

The function block `MX_DIMSET` sets number rows `m` of the vector or number of number of rows `m` and the leading dimension `ld` of the matrix referenced by `uMV`. Outputs of the block shows the other matrix/vector dimensions – number of columns `n` and count of allocated elements `cnt`.

The output reference `yMV` is always set to the corresponding input reference `uMV`. The error flag `E` is set to `on` if:

- the reference `uMV` is not defined (i.e. input `uMV` is not connected),
- the number of rows `m < 1` or `m > ld`,
- the leading dimension `ld > cnt`.

**Inputs**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uMV</code></td>
<td>Input reference to a matrix or vector</td>
<td>reference</td>
</tr>
<tr>
<td><code>m</code></td>
<td>Number of matrix rows</td>
<td>long</td>
</tr>
<tr>
<td><code>ld</code></td>
<td>Leading dimension (<code>&gt;= number of rows</code>)</td>
<td>long</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>yMV</code></td>
<td>Output reference to a matrix or vector</td>
<td>reference</td>
</tr>
<tr>
<td><code>n</code></td>
<td>Number of matrix columns</td>
<td>long</td>
</tr>
<tr>
<td><code>cnt</code></td>
<td>Count of allocated elements (<code>&gt;= number of rows * number of columns</code>)</td>
<td>long</td>
</tr>
<tr>
<td><code>E</code></td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
MX_DSAGET – Set subarray of A into B

Block Symbol

Licence: STANDARD

Function Description

Generally, the function block MX_DSAGET copies the subarray (submatrix) of matrix referenced by \( uA \) into the matrix referenced by \( uB \).

The output references \( yA \) and \( yB \) are always set to the corresponding input references \( uA \) and \( uB \). If \( HLD = \text{on} \) then nothing is copied otherwise the submatrix of matrix referenced by \( uA \) starting the row with zero based index \( I \) and the column with zero based index \( J \) containing \( M \) rows and \( N \) columns is copied (with respect to the value of the input \( \text{uplo} \)) to the matrix referenced by \( uB \). The mentioned variables have the following meanings:

- If the input \( i \leq 0 \) then \( I \) is set to 0 else if \( i \geq MA \) then \( I \) is set to \( MA - 1 \) else \( I \) is set to \( i \), where \( MA \) is the number of rows of the matrix referenced by \( uA \).

- If the input \( j \leq 0 \) then \( J \) is set to 0 else if \( j \geq NA \) then \( J \) is set to \( NA - 1 \) else \( J \) is set to \( j \), where \( NA \) is the number of columns of the matrix referenced by \( uA \).

- Number of copied rows \( M \) is set in two stages. First, \( M \) is set to minimum of \( MA - I \) and number of rows of the matrix referenced by \( uB \). Second, if \( m > 0 \) then \( M \) is set to the minimum of \( m \) and \( M \).

- Number of copied columns \( N \) is set in two stages. First, \( N \) is set to minimum of \( NA - J \) and number of columns of the matrix referenced by \( uB \). Second, if \( n > 0 \) then \( N \) is set to the minimum of \( n \) and \( N \).

The error flag \( E \) is set to \text{on} if:

- the reference \( uA \) or \( uB \) is not defined (i.e. input \( uA \) or \( uB \) is not connected),

- \( \text{uplo} \) is less than 0 or greater than 3,

- the number of elements of the matrix referenced by \( uB \) is less than \( M \times N \).
Inputs

- **uA**: Input reference to matrix A  
- **uB**: Input reference to matrix B  
- **uplo**: Part of the matrix to be copied  
  - 0 ··· All  
  - 1 ··· All  
  - 2 ··· Upper  
  - 3 ··· Lower  
- **i**: Index of the subarray first row  
- **j**: Index of the subarray first column  
- **m**: Number of matrix rows  
- **n**: Number of matrix columns  
- **HLD**: Hold

Outputs

- **yA**: Output reference to matrix A  
- **yB**: Output reference to matrix B  
- **E**: Error indicator
**MX_DSAREF** – Set reference to subarray of A into B

**Block Symbol**

![Block Symbol](image)

**Licence:** STANDARD

**Function Description**

The function block `MX_DSAREF` creates a reference `yB` to the subarray (submatrix) of matrix referenced by `uA`. This operation is very fast because no matrix element is copied.

The output reference `yA` is always set to the corresponding input reference `uA`, the output reference `yB` is created inside each instance of this function block. If `HLD = on` then no other operation is performed otherwise the reference to the matrix `yB` is created with the following properties:

- Number of rows of the submatrix is set to `M - i`, where `M` is number of rows of the matrix referenced by `uA`.

- Number of columns of the submatrix is set to `N - j`, where `N` is number of columns of the matrix referenced by `uA`.

- The first element in position `(0, 0)` of the submatrix is the element of the matrix referenced by `uA` in position `(i, j)`, all indices are zero based.

- The matrix referenced by `yB` has the same leading dimension as the matrix referenced by `uA`.

The error flag `E` is set to `on` if:

- the reference `uA` is not defined (i.e. input `uA` is not connected),

- `0 > i ≥ M`.

- `0 > j ≥ N`.

**Inputs**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>i</td>
<td>Index of the subarray first row</td>
<td>long</td>
</tr>
<tr>
<td>j</td>
<td>Index of the subarray first column</td>
<td>long</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>
Parameter

ay  Output reference of the subarray  \[0 \ 0\]  double

Outputs

yA  Output reference to matrix A  reference
yB  Output reference to matrix B  reference
E   Error indicator  bool
CHAPTER 14. MATRIX – BLOCKS FOR MATRIX AND VECTOR OPERATIONS

MX_DSASET — Set A into subarray of B

Block Symbol

Function Description

Generally, the function block MX_DSASET copies the matrix referenced by uA into the subarray (submatrix) of the matrix referenced by uB.

The output references yA and yB are always set to the corresponding input references uA and uB. If HLD = on then nothing is copied otherwise the matrix referenced by uA is copied (with respect to the value of the input uplo) to the submatrix of the matrix referenced by uB to the row with zero based index I and the column with zero based index J containing M rows and N columns. The mentioned variables have the following meanings:

- If the input i \leq 0 then I is set to 0 else if i \geq MB then I is set to MB − 1 else I is set to i, where MB is the number of rows of the matrix referenced by uB.

- If the input j \leq 0 then J is set to 0 else if j \geq NB then J is set to NB − 1 else J is set to j, where NB is the number of columns of the matrix referenced by uB.

- Number of copied rows M is set in two stages. First, M is set to minimum of MB − I and number of rows of the matrix referenced by uA. Second, if m > 0 then M is set to the minimum of m and M.

- Number of copied columns N is set in two stages. First, N is set to minimum of NB − J and number of columns of the matrix referenced by uA. Second, if n > 0 then N is set to the minimum of n and N.

The error flag E is set to on if:

- the reference uA or uB is not defined (i.e. input uA or uB is not connected),
- uplo is less than 0 or greater than 3,
- the number of elements of the matrix referenced by uB is less than M \times N.
Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uA</td>
<td>Input reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>uB</td>
<td>Input reference to matrix B</td>
<td>reference</td>
</tr>
<tr>
<td>uplo</td>
<td>Part of the matrix to be copied</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>0 ...... All</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 ...... All</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>Index of the subarray first row</td>
<td>long</td>
</tr>
<tr>
<td>j</td>
<td>Index of the subarray first column</td>
<td>long</td>
</tr>
<tr>
<td>m</td>
<td>Number of matrix rows</td>
<td>long</td>
</tr>
<tr>
<td>n</td>
<td>Number of matrix columns</td>
<td>long</td>
</tr>
<tr>
<td>HLD</td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yA</td>
<td>Output reference to matrix A</td>
<td>reference</td>
</tr>
<tr>
<td>yB</td>
<td>Output reference to matrix B</td>
<td>reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
MX_DTRNSP – General matrix transposition: \( B := \alpha A^T \)

### Function Description

The function block \texttt{MX_DTRNSP} stores the scalar multiple of the general (i.e. rectangular) matrix referenced by \( uA \) into the matrix referenced by \( uB \).

The output references \( yA \) and \( yB \) are always set to the corresponding input references \( uA \) and \( uB \). If \( HLD = \text{on} \) then nothing else is done otherwise the BLAS-like function \texttt{X_DTRNSP} is called internally:

\[
\texttt{X_DTRNSP}(M, N, ALPH, uA, LDA, uB, LDB);
\]

where parameters of \texttt{X_DTRNSP} are set in the following way:

- \( M \) is number of rows of the matrix referenced by \( uA \).
- \( N \) is number of columns of the matrix referenced by \( uA \).
- If the input \( \alpha \) is equal to 0 then \( ALPH \) is set to 1 else \( ALPH \) is set to \( \alpha \).
- \( LDA \) and \( LDB \) are leading dimensions of matrices referenced by \( uA \) and \( uB \).

The error flag \( E \) is set to \( \text{on} \) if:

- the reference \( uA \) or \( uB \) is not defined (i.e. input \( uA \) or \( uB \) is not connected),
- the call of the function \texttt{X_DTRNSP} returns error using the function \texttt{XERBLA}, see the system log.

### Inputs

<table>
<thead>
<tr>
<th>( uA )</th>
<th>Input reference to matrix A</th>
<th>( uB )</th>
<th>Input reference to matrix B</th>
<th>( \alpha )</th>
<th>Scalar coefficient alpha</th>
<th>( HLD )</th>
<th>Hold</th>
</tr>
</thead>
</table>

### Outputs

<p>| ( yA ) | Output reference to matrix A | | | | | | |</p>
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>yB</td>
<td>Output reference to matrix B</td>
<td>reference</td>
</tr>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
MX_DTRNSQ – Square matrix in-place transposition: \( A := \alpha A^T \)

Function Description

The function block MX_DTRNSQ transpose the scalar multiple of the square matrix referenced by \( uA \) in-place.

The output reference \( yA \) is always set to the corresponding input references \( uA \). If \( HLD = \text{on} \) then nothing else is done otherwise the BLAS-like function X_DTRNSQ is called internally:

\[
X_DTRNSQ(N, \alpha, uA, LDA);
\]

where parameters of \( X_DTRNSQ \) are set in the following way:

- \( N \) is number of rows and columns of the matrix referenced by \( uA \).
- If the input \( \alpha \) is equal to 0 then \( \text{ALPHA} \) is set to 1 else \( \text{ALPHA} \) is set to \( \alpha \).
- \( \text{LDA} \) is the leading dimension of the matrix referenced by \( uA \).

The error flag \( E \) is set to \( \text{on} \) if:

- the reference \( uA \) is not defined (i.e. input \( uA \) is not connected),
- the matrix referenced by \( uA \) is not square,
- the call of the function \( X_DTRNSQ \) returns error using the function \( \text{XERBLA} \), see the system log.

Inputs

- \( uA \) Input reference to matrix A reference
- \( \alpha \) Scalar coefficient alpha double
- \( HLD \) Hold bool

Outputs

- \( yA \) Output reference to matrix A reference
- \( E \) Error indicator bool
MX_FILL – Fill real matrix or vector

Block Symbol

Licence: STANDARD

Function Description

The function block MX_FILL fills elements of the matrix or vector referenced by uMV according to the input mode.

The output reference yMV is always set to the corresponding input references uMV. If HLD = on then nothing else is done.
The error flag E is set to on if:

- the reference uMV is not defined (i.e. input uMV is not connected),
- $0 > \text{mode} > 4$.

Inputs

- **uMV**: Input reference to a matrix or vector
- **value**: Fill value of matrix/vector
- **mode**: Fill mode
  - $0, 1 \ldots$ Value – All elements are set to \texttt{value}
  - $2 \ldots \ldots$ Ones – All elements are set to 1
  - $3 \ldots \ldots$ Diagonal value – Diagonal is set to \texttt{value}, the other elements to 0
  - $4 \ldots \ldots$ Diagonal ones – Initializes identity matrix (\texttt{eye})
- **HLD**: Hold

Outputs

- **yMV**: Output reference to a matrix or vector
- **E**: Error indicator
CHAPTER 14. MATRIX – BLOCKS FOR MATRIX AND VECTOR OPERATIONS

**MX_MAT** – Matrix data storage block

**Block Symbol**

**Function Description**

The function block **MX_MAT** allocates memory (during the block initialization) for \( m \times n \) elements of the type determined by the parameter **etype** of the matrix referenced by the output **yMat**. Also matrix leading dimension can be set by the parameter **ld**. If \( ld < m \) then the leading dimension is set to \( m \).

Note that the present version of the **MATRIX** function block set supports only matrices with the **etype** equal to 8: **Double**.

**Parameters**

- **m**: Number of matrix rows
  - 1 \( \uparrow 1000000000 \) \( \odot 10 \) long
- **n**: Number of matrix columns
  - 1 \( \uparrow 1000000000 \) \( \odot 10 \) long
- **ld**: Leading dimension (\( \geq \) number of rows)
  - 0 \( \uparrow 1000000000 \) \( \odot 10 \) long
- **etype**: Type of elements
  - 1 .... Bool
  - 2 .... Byte
  - 3 .... Short
  - 4 .... Long
  - 5 .... Word
  - 6 .... DWord
  - 7 .... Float
  - 8 .... Double
  - 9 .... --
  - 10 .... Large

**Output**

- **yMat**: Output reference to a matrix
**MX_RAND** – Randomly generated matrix or vector

**Block Symbol**

Licence: **STANDARD**

**Function Description**

The function block **MX_RAND** generates random elements of the matrix or vector referenced by **uMV**.

The output reference **yMV** is always set to the corresponding input references **uMV**. If **HLD = on** then nothing is generated otherwise pseudo-random values of the matrix or vector elements referenced by **uMV** are generated using these rules:

- If the parameter **BIP** is **on** then the generated elements are inside the interval \([-\text{scale}; \text{scale}]\) else they are inside the interval \([0; \text{scale}]\).

- Elements are internally generated using the standard C language function **rand()** which generates pseudo-random numbers in the range from 0 to **RAND_MAX**. Note, that the value of **RAND_MAX** can be platform dependent (and it should be at least 32767).

- The rising edge on the input **SET** causes that the standard C language function **srand(nseed)** (initializes the pseudo-random generator with the value of **nseed**) is called before the generation of random elements. The same sequences of pseudo-random numbers are generated after calls of **srand(nseed)** for the same values of **nseed**.

The error flag **E** is set to **on** if the reference **uMV** is not defined (i.e. input **uMV** is not connected).

**Inputs**

<table>
<thead>
<tr>
<th><strong>Input</strong></th>
<th><strong>Description</strong></th>
<th><strong>Type</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uMV</strong></td>
<td>Input reference to a matrix or vector</td>
<td>reference</td>
</tr>
<tr>
<td><strong>nseed</strong></td>
<td>Random number seed</td>
<td>long</td>
</tr>
<tr>
<td><strong>SET</strong></td>
<td>Set initial value of random number generator to <strong>nseed</strong> on rising edge</td>
<td>bool</td>
</tr>
<tr>
<td><strong>HLD</strong></td>
<td>Hold</td>
<td>bool</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th><strong>Parameter</strong></th>
<th><strong>Description</strong></th>
<th><strong>Type</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BIP</strong></td>
<td>Bipolar random values flag</td>
<td>bool</td>
</tr>
<tr>
<td>scale</td>
<td>Random values multiplication factor</td>
<td>( \odot 1.0 )</td>
</tr>
</tbody>
</table>

**Outputs**

| yMV | Output reference to a matrix or vector | reference |
| E   | Error indicator                     | bool      |
**MX_REFCOPY** – Copies input references of matrices A and B to their output references

**Block Symbol**

![Block Symbol](image)

**Licence:** STANDARD

**Function Description**

The function block **MX_REFCOPY** is an administrative block of the MATRIX blockset. It does nothing else than copying the input references \( u_A \) and \( u_B \) to the corresponding output references \( y_A \) and \( y_B \).

But suitable insertion of this block to the function block scheme can substantially influence (change) the execution order of blocks which can be very advantageous especially in combination with such blocks as e.g. **MX_DSAREF**.

**Inputs**

<table>
<thead>
<tr>
<th>( u_A )</th>
<th>Input reference to matrix A</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u_B )</td>
<td>Input reference to matrix B</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>( y_A )</th>
<th>Output reference to matrix A</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y_B )</td>
<td>Output reference to matrix B</td>
</tr>
</tbody>
</table>
**MX_VEC – Vector data storage block**

**Block Symbol**

[MX_VEC]  

**Licence:** STANDARD

**Function Description**

The function block **MX_VEC** allocates memory (during the block initialization) for n elements of the type determined by the parameter **etype** of the vector referenced by the output **yVec**.

Note that the present version of the **MATRIX** function block set supports only vectors with the **etype** equal to **8**: Double.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Number of vector elements</td>
</tr>
<tr>
<td>etype</td>
<td>Type of elements</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>etype</th>
<th>Type</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>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Large</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yVec</td>
<td>Output reference to a vector</td>
</tr>
</tbody>
</table>

**Output**

*yVec**  

**Output reference**
**MX_WRITE – Write a Matrix/Vector to the console/system log**

**Block Symbol**

![Block Symbol]

**Licence:** STANDARD

**Function Description**

This function block can write a vector or matrix to the console or the system log. The severity of the console/system log output is set by the parameter `mode` in combination with settings of system log from REXYGEN Studio, menu Target/Configure System Log. Written data can be viewed in REXYGEN Studio, after opening the system log window by the command Target/Show System Log. The function block is very useful for debugging purposes of matrix/vector algorithms.

The output references `yMV` is always set to the input reference `uMV`. If `RUN = off` then nothing else is done otherwise matrix or vector is written to the system log if the configured target logging level for function blocks contains the configured `mode`. Format of each matrix/vector element is determined by parameters `mchars` and `mdec`. The error flag `E` is set to `on` if:

- the reference `uMV` is not defined (i.e. input `uMV` is not connected),
- $3 > mchars > 25$,
- $0 > mdec > mchars - 2$.

**Inputs**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uMV</code></td>
<td>Input reference to a matrix or vector</td>
</tr>
<tr>
<td><code>RUN</code></td>
<td>Enable execution</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Symbol</code></td>
<td>Matrix/vector symbolic name for console or log output</td>
</tr>
<tr>
<td><code>mchars</code></td>
<td>Number of characters per single element</td>
</tr>
<tr>
<td><code>mdec</code></td>
<td>Number of decimal digits per single element</td>
</tr>
<tr>
<td><code>mode</code></td>
<td>Severity mode of writing</td>
</tr>
</tbody>
</table>

<p>| 1 | None |
| 2 | Verbose |
| 3 | Information |
| 4 | Warning |
| 5 | Error |</p>
<table>
<thead>
<tr>
<th>yMV</th>
<th>Output reference to a matrix or vector</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
</tbody>
</table>
RTOV – Vector multiplexer

Block Symbol 

License: STANDARD

Function Description

The RTOV block can be used to create vector signals in the REXYGEN system. It combines the scalar input signals into one vector output signal.

It is also possible to chain the RTOV blocks to create signals with more than 8 items.

The $n_{\text{max}}$ parameter defines the maximal number of items in the vector (in other words, the size of memory allocated for the signal). The $\text{offset}$ 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
- $u1$: Analog input of the block
- $u2$: Analog input of the block
- $u3$: Analog input of the block
- $u4$: Analog input of the block
- $u5$: Analog input of the block
- $u6$: Analog input of the block
- $u7$: Analog input of the block
- $u8$: Analog input of the block

Parameters

- $n_{\text{max}}$: Allocated size of vector
- $\text{offset}$: Index of the first input in vector
- $n$: Number of valid inputs

Output

- $yVec$: Vector signal
**SWVMR – Vector/matrix/reference signal switch**

**Function Description**

The SWVMR allows switching of vector or matrix signals. It also allow switching of motion axes in motion control algorithms (see the RM_Axis block).

Use the SSW block or its alternatives SWR and SELU for switching simple signals.

**Inputs**

<table>
<thead>
<tr>
<th>uRef0</th>
<th>Vector signal reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>uRef1</td>
<td>Vector signal reference</td>
</tr>
<tr>
<td>uRef2</td>
<td>Vector signal reference</td>
</tr>
<tr>
<td>uRef3</td>
<td>Vector signal reference</td>
</tr>
<tr>
<td>uRef4</td>
<td>Vector signal reference</td>
</tr>
<tr>
<td>uRef5</td>
<td>Vector signal reference</td>
</tr>
<tr>
<td>uRef6</td>
<td>Vector signal reference</td>
</tr>
<tr>
<td>uRef7</td>
<td>Vector signal reference</td>
</tr>
<tr>
<td>iSW</td>
<td>Active signal selector</td>
</tr>
</tbody>
</table>

**Output**

<table>
<thead>
<tr>
<th>yRef</th>
<th>Vector signal reference</th>
</tr>
</thead>
</table>
VTOR – Vector demultiplexer

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

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPC</td>
<td>External program call</td>
<td>426</td>
</tr>
<tr>
<td>HTTP</td>
<td>HTTP GET or POST request (obsolete)</td>
<td>429</td>
</tr>
<tr>
<td>HTTP2</td>
<td>Block for generating HTTP GET or POST requests</td>
<td>431</td>
</tr>
<tr>
<td>SMTP</td>
<td>Send e-mail message via SMTP</td>
<td>433</td>
</tr>
<tr>
<td>RDC</td>
<td>Remote data connection</td>
<td>435</td>
</tr>
<tr>
<td>REXLANG</td>
<td>User programmable block</td>
<td>439</td>
</tr>
</tbody>
</table>
CHAPTER 15. SPEC – SPECIAL BLOCKS

EPC – External program call

Block Symbol Licence: ADVANCED

Function Description

The EPC block executes an external program upon a rising edge (off→on) occurring at the EXEC input. The name and options of the program are defined by the cmd parameter. The format is the same as if the program was executed from the command line of the operating system.

It is possible to pass data from the REXYGEN system to the external program via files. The formatting of the files is defined by the format parameter. All the currently supported formats are textual and simple, which allows straightforward processing of the data in arbitrary program. Use e.g.

values=load('-ASCII', 'epc_inputVec1');

for loading the data in MATLAB or

values=read('epc_inputVec1',-1,32);

in SCILAB. The filename and number of columns must be adjusted for the given project. Data exchange in the opposite direction is naturally also supported, the REXYGEN system can read the files in the same format.

The block works in two modes. In basic mode, the rising edge on the EXEC input triggers reading the data on inputs and storing them in the ifns file. The values of the i-th input vector uVec<i> are stored in the i-th file from the ifns list. In sampling mode, the data from the input vectors are stored in each period of the control algorithm. In both cases the values from one time instant form one line in the file.

Analogically, the data from output files are copied to the outputs of the block (one line from the i-th file in the ofns list to the i-th output vector yVec<i>).

The inputs working in the sampling mode are defined by the 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 RT0V block to combine individual signals into a vector one for the uVec input.
The **RTOV** blocks can be chained, therefore it is possible to create a vector of arbitrary dimension. Similarly, use the **VTOR** block to demultiplex a vector signal to individual signals.

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uVec1..uVec8</td>
<td>Input vector signal</td>
<td>reference</td>
</tr>
<tr>
<td>EXEC</td>
<td>External program is called on rising edge</td>
<td>bool</td>
</tr>
<tr>
<td>RESET</td>
<td>Block reset (deletes the input and output files and terminates the external program)</td>
<td>bool</td>
</tr>
<tr>
<td>DSI</td>
<td>Disable inputs sampling</td>
<td>bool</td>
</tr>
<tr>
<td>DSO</td>
<td>Disable outputs sampling</td>
<td>bool</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yVec1..yVec8</td>
<td>Output vector signal</td>
<td>reference</td>
</tr>
<tr>
<td>DONE</td>
<td>External program finished</td>
<td>bool</td>
</tr>
<tr>
<td>BUSY</td>
<td>External program running</td>
<td>bool</td>
</tr>
<tr>
<td>ERR</td>
<td>Error flag</td>
<td>bool</td>
</tr>
<tr>
<td>errID</td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td>res</td>
<td>External program return code</td>
<td>long</td>
</tr>
<tr>
<td>icnt</td>
<td>Current input sample</td>
<td>long</td>
</tr>
<tr>
<td>ocnt</td>
<td>Current output sample</td>
<td>long</td>
</tr>
</tbody>
</table>

### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd</td>
<td>Operating system command to execute</td>
<td>string</td>
</tr>
<tr>
<td>ifns</td>
<td>Input filenames (separated by semicolon)</td>
<td>string</td>
</tr>
<tr>
<td>ofns</td>
<td>Output filenames (separated by semicolon)</td>
<td>string</td>
</tr>
<tr>
<td>sl</td>
<td>List of inputs working in the <em>sampling mode</em>. 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>
</tr>
<tr>
<td>ifm</td>
<td>Maximum number of input samples</td>
<td>long</td>
</tr>
<tr>
<td>format</td>
<td>Format of input and output files</td>
<td>string</td>
</tr>
<tr>
<td>nmax</td>
<td>Maximum output vectors length</td>
<td>long</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:
\[
\text{chrt -o 0 extprg.sh,}
\]
where \texttt{extprg.sh} is the original external program.

- The size of signals is limited by parameter \texttt{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 \texttt{ifns} and \texttt{ofns} files for implementation reasons.
The names of these files are extended by the underscore character.

- The \texttt{ifns} and \texttt{ofns} paths are relative to the folder where the archives of the
\texttt{REXYGEN} system are stored. It is recommended to define a symbolic link to a
RAM-drive inside this folder for improved performance. On the other hand, for
long series of data it is better to store the data on a permanent storage medium
because the data can be appended e.g. after a power-failure recovery.

- The \texttt{OSCALL} block can be used for execution of some operating system functions.
HTTP – HTTP GET or POST request (obsolete)

Function Description

The HTTP block performs a single HTTP GET or POST request. Target address (URL) is defined by url parameter and urldata input. A final URL is formed in the way so that urldata input is simply added to url parameter.

HTTP request is started by the TRG parameter. Then the BUSY output is set until a request is finished, which is signaled by the DONE output. In case of an error, the ERROR output is set. The errId output carries last error identified by REXYGEN system error code. The hterror carries a HTTP status code. All data sent back by server to client is stored in the data output.

The block may be run in blocking or non-blocking mode which is specified by the BLOCKING parameter. In blocking mode, execution of a task is suspended until a request is finished. In non-blocking mode, the block performs only single operation depending on available data and execution of a task is not blocked. It is advised to always run HTTP block in non-blocking mode. It is however necessary to mention that on various operating systems some operations can not be performed in the non-blocking mode, so be careful and do not use this block in quick tasks or in tasks with short execution period. The non-blocking operation is best supported on GNU/Linux operating system. The maximal duration of a request performed by the HTTP block is specified by the timeout parameter.

The block supports user authentication using basic HTTP authentication method. User name and password may be specified by user and password parameters. The block also supports secure HTTP (HTTPS). It is also possible to let the block verify server’s certificate by setting the VERIFY parameter. SSL certificate of a server or server’s trusted certificate authority must be stored in the certificate parameter in a PEM format. The block does not support any certificate storage.

Parameters postmime and acceptmime specify MIME encoding of data being sent to server or expected encoding of a HTTP response.

Parameters nmax, postmax, and datamax specify maximum sizes of buffers allocated by the block. The nmax parameter is maximal size of any string parameter. The postmax parameter specifies a maximal size of postdata. The datamax parameter specifies a maximal size of data.
## Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>postdata</td>
<td>Data to put in HTTP POST request</td>
<td>string</td>
</tr>
<tr>
<td>urldata</td>
<td>Data to append to URL address</td>
<td>string</td>
</tr>
<tr>
<td>TRG</td>
<td>Trigger of the selected action</td>
<td>bool</td>
</tr>
</tbody>
</table>

## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>URL address to send the HTTP request to</td>
<td>string</td>
</tr>
<tr>
<td>method</td>
<td>HTTP request type</td>
<td>long</td>
</tr>
<tr>
<td>user</td>
<td>User name</td>
<td>string</td>
</tr>
<tr>
<td>password</td>
<td>Password</td>
<td>string</td>
</tr>
<tr>
<td>certificate</td>
<td>Authentication certificate</td>
<td>string</td>
</tr>
<tr>
<td>VERIFY</td>
<td>Enable server verification (valid certificate)</td>
<td>bool</td>
</tr>
<tr>
<td>postmime</td>
<td>MIME encoding for POST request</td>
<td>string</td>
</tr>
<tr>
<td>acceptmime</td>
<td>MIME encoding of HTTP response</td>
<td>string</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>long</td>
</tr>
<tr>
<td>postmax</td>
<td>Allocated memory for POST request data</td>
<td>long</td>
</tr>
<tr>
<td>datamax</td>
<td>Allocated memory for HTTP response</td>
<td>long</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Response data</td>
<td>string</td>
</tr>
<tr>
<td>BUSY</td>
<td>Sending HTTP request</td>
<td>bool</td>
</tr>
<tr>
<td>DONE</td>
<td>HTTP request processed</td>
<td>bool</td>
</tr>
<tr>
<td>ERROR</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
<tr>
<td>errId</td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td>hterror</td>
<td>HTTP response</td>
<td>long</td>
</tr>
</tbody>
</table>
HTTP2 – Block for generating HTTP GET or POST requests

Block Symbol

Function Description

The HTTP block performs a single HTTP GET or POST request. Target address (URL) is defined by url parameter and urldata input. A final URL is formed in the way so that urldata input is appended to the url parameter. The header input can be used for declaration of additional header fields.

A HTTP request is started by the TRG input. Then the BUSY output is set until the request is finished, which is signaled by a pulse at the DONE output. In case of an error, the pulse is generated at the ERROR output. The errId output carries information about the last error identified by REXYGEN system error code. The hterror carries a HTTP status code. All data received from server is published via the data output. All error outputs are reset when a new HTTP request is triggered by the TRG input.

The block may be run in blocking or non-blocking mode which is specified by the BLOCKING parameter. In blocking mode, execution of a task is suspended until the request is finished. In non-blocking mode, the block performs only single operation depending on available data and execution of a task is not blocked. It is advised to always run HTTP block in non-blocking mode. It is however necessary to mention that on various operating systems some operations cannot be performed in the non-blocking mode, so be careful and do not use this block in quick tasks (QTASK) or in tasks with short execution period. The non-blocking operation is best supported on GNU/Linux operating system. The maximal duration of a request performed by the HTTP block is specified by the timeout parameter.

The block supports user authentication using basic HTTP authentication method. User name and password may be specified by user and password parameters. The block also supports secure HTTP (HTTPS). It is also possible to let the block verify server’s certificate by setting the VERIFY parameter. SSL certificate of a server or server’s trusted certificate authority must be stored in the certificate parameter in a PEM format. The block does not support any certificate storage.

Parameters postmime and acceptmime specify MIME encoding of data being sent to server and expected encoding of the HTTP response.

Parameters nmax, postmax, and datamax specify maximum sizes of buffers allocated by the block. The nmax parameter is maximal size of any string parameter. The postmax parameter specifies a maximal size of postdata. The datamax parameter specifies a
maximal size of data.

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>postdata</td>
<td>Data to put in HTTP POST request</td>
<td>string</td>
</tr>
<tr>
<td>urldata</td>
<td>Data to append to URL address</td>
<td>string</td>
</tr>
<tr>
<td>header</td>
<td>Additional header fields</td>
<td>string</td>
</tr>
<tr>
<td>TRG</td>
<td>Trigger of the selected action</td>
<td>bool</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>URL address to send the HTTP request to</td>
<td>string</td>
</tr>
<tr>
<td>method</td>
<td>HTTP request type</td>
<td>long</td>
</tr>
<tr>
<td>1</td>
<td>GET</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>POST</td>
<td></td>
</tr>
<tr>
<td>user</td>
<td>User name</td>
<td>string</td>
</tr>
<tr>
<td>password</td>
<td>Password</td>
<td>string</td>
</tr>
<tr>
<td>certificate</td>
<td>Authentication certificate</td>
<td>string</td>
</tr>
<tr>
<td>VERIFY</td>
<td>Enable server verification (valid certificate)</td>
<td>bool</td>
</tr>
<tr>
<td>postmime</td>
<td>MIME encoding for POST request</td>
<td>string</td>
</tr>
<tr>
<td>acceptmime</td>
<td>MIME encoding for GET request</td>
<td>string</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>long</td>
</tr>
<tr>
<td>postmax</td>
<td>Allocated memory for POST request data</td>
<td>long</td>
</tr>
<tr>
<td>datamax</td>
<td>Allocated memory for HTTP response</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>data</td>
<td>Response data</td>
<td>string</td>
</tr>
<tr>
<td>BUSY</td>
<td>Sending HTTP request</td>
<td>bool</td>
</tr>
<tr>
<td>DONE</td>
<td>HTTP request processed</td>
<td>bool</td>
</tr>
<tr>
<td>ERROR</td>
<td>Error indicator</td>
<td>bool</td>
</tr>
<tr>
<td>errId</td>
<td>Error code</td>
<td>error</td>
</tr>
<tr>
<td>hterror</td>
<td>HTTP response</td>
<td>long</td>
</tr>
</tbody>
</table>
SMTP – Send e-mail message via SMTP

Block Symbol

Function Description

The SMTP block sends a single e-mail message via standard SMTP protocol. The block acts as a simple e-mail client. It does not implement a mail server.

The contents of a message is defined by the inputs \texttt{subj} and \texttt{body}. Parameters \texttt{from} and \texttt{to} specify sender and receiver of a message. A message is sent when the \texttt{TRG} parameter is set. Then the \texttt{BUSY} output is set until the 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 the last error identified by REXYGEN system error code. The \texttt{domain} parameter must always be set to identify the target device. The default value should work in most cases. There can be multiple recipients of the message. In such a case, the individual e-mail addresses must be comma-separated and no space character may be present.

The block may be run in non-blocking or blocking mode, which is specified by the \texttt{BLOCKING} parameter. In the blocking mode, the execution of a task is suspended until a request is finished. In the 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 \texttt{timeout} parameter.

The block supports user authentication using standard SMTP authentication method. User name and password may be specified by the \texttt{user} and \texttt{password} parameters. The block also supports secure connection. The encryption method is selected by the \texttt{tls} parameter. 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.

The length of the whole message (subject, body and headers) is limited to a maximum of 1024 characters.

Inputs

\begin{verbatim}
subj Subject of the e-mail message string
\end{verbatim}
CHAPTER 15. SPEC – SPECIAL BLOCKS

body Body of the e-mail message string
TRG Trigger of the selected action bool

Parameters

server SMTP server address string
to E-mail of the recipient string
from E-mail of the sender string
tls Encryption method long
  1 ...... None
  2 ...... StartTLS
  3 ...... TLS
user User name string
password Password string
domain Domain name or identification of the target device string
auth Authentication method long
  1 ...... Login
  2 ...... Plain
certificate Authentication certificate string
VERIFY Enable server verification (valid certificate) bool
timeout Timeout interval double
BLOCKING Wait for the operation to finish bool

Outputs

BUSY Sending e-mail bool
DONE E-mail has been sent bool
ERROR Error indicator bool
errId Error code error
**RDC – Remote data connection**

**Block Symbol**

![Block Symbol Image]

**Licence:** ADVANCED

**Function Description**

The RDC block is a special input-output block. The values are transferred between two blocks on different computers, eventually two different Simulinks on the same computer or Simulink and the REXYGEN system on the same computer. In order to communicate, the two RDC blocks must have the same id number. The communication is based on UDP/IP protocol. This protocol is used as commonly as the more known TCP/IP, i.e. it works over all LAN networks and the Internet. The algorithm performs the following operations in each step:

- If HLD = on, the block execution is terminated.

- If the period parameter is a positive number, the difference between the system timer and the time of the last packet sending is evaluated. The block execution is stopped if the difference does not exceed the period parameter. If the period parameter is zero or negative, the time difference is not checked.

- A data packet is created. The packet includes block number, the so-called invoke number (serial number of the packet) and the values u0 to u15. All values are stored in the commonly used so-called network byte order, therefore the application is computer and/or processor independent.

- The packet is sent to the specified IP address and port.

- The invoke number is increased by 1.

- It is checked whether any incoming packets have been received.

- If so, the packet validity is checked (size, id number, invoke number).
• If the data is valid, all outputs $y_0$ to $y_{15}$ 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**

<table>
<thead>
<tr>
<th>HLD</th>
<th>Input for disabling the execution of the block. No packets are received nor transmitted when HLD = on.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_0..u_{15}$</td>
<td>Values which are sent/written to the output values $y_0$ to $y_{15}$ of the paired block</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>iE</th>
<th>Displays the code of the last error. The error codes are listed below:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
</tbody>
</table>

*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 REXYGEN system, the REXYGEN Diagnostics diagnostic program and the TRND block must be used.
The simplicity of the example is intentional. The goal is to demonstrate the functionality of the block, not the complexity of the system. In reality, the RDC block is used in more complex tasks, e.g. for remote tuning of the PID controller as shown below. The PID control algorithm is running on Computer1 while the tuning algorithm is executed by Computer2. See the PIDU, PIDMA and SSW blocks for more details.

OPC server of the RDC block

There is also an OPC server embedded in the RDC block. Detailed description will be available soon.
REXLANG – User programmable block

Function Description

The standard function blocks of the REXYGEN system cover the most typical needs in control applications. But there still exist situations where it is necessary (or more convenient) to implement an user-defined function. The REXLANG block covers this case. It implements an user-defined algorithm written in a scripting language very similar to the C language (or Java).

Scripting language

As mentioned, the scripting language is similar to the C language. Nevertheless, there are some differences and limitations:

- Only the `double` and `long` data types are supported (it is possible to use `int`, `short`, `bool` as well, but these are internally converted to `long`. The `float` type can be used but it is converted internally to `double`. The `typedef` type is not defined.

- Pointers and structures are not implemented. However, it is possible to define arrays and use the indexes (the `[ ]` operator).

- The ``, `operator is not implemented.

- The preprocessor supports the following commands: `#include`, `#define`, `#ifdef .. | #else .. | #endif`, `#ifndef .. | #else .. | #endif` (i.e. `#pragma` and `#if .. | #else .. | #endif` are not supported).

- The standard ANSI C libraries are not implemented, however the majority of mathematic functions from `math.h` and some other functions are implemented (see the text below).
• The input, output and parameter keywords are defined for referencing the REXLANG block inputs, outputs and parameters. System functions for controlling the execution and diagnostics are implemented (see the text below).

• The main() function is executed periodically during runtime. Alongside the main() function the init() (executed once at startup), exit() (executed once when the control algorithm is stopped) and the parchange() (executed on parameters change in REXYGEN).

• The functions and procedures without parameters must be explicitly declared void.

• The identifiers cannot be overloaded, i.e. the keywords and built-in functions cannot share the name with an identifier. The local and global variables cannot share the same name.

• Array initializers are not supported. Neither in local arrays nor the global ones.

Scripting language syntax

The scripting language syntax is based on the C language, but pointers are not supported and the data types are limited to long and double. Moreover the input, output and parameter keywords are defined for referencing the REXLANG block inputs, outputs and parameters. The syntax is as follows:

• <type> input(<input number>) <variable name>;

• <type> output(<outpt number>) <variable name>;

• <type> parameter(<parameter number>) <variable name>;

The input and parameter variables are read-only while the output variables are write-only. For example:

double input(1) input_signal; /* declaration of a variable of type double, which corresponds with the u1 input of the block */

long output(2) output_signal; /* declaration of a variable of type long, which corresponds with the y2 output of the block */

input_signal = 3; //not allowed, inputs are read-only
sum = output_signal + 1; //not allowed, outputs are write-only
if (input_signal>1) output_signal = 3 + input_signal; //correct

Available functions

The following functions are available in the scripting language:
**Mathematic functions** (see ANSI C, math.h):
atan, sin, cos, exp, log, sqrt, tan, asin, acos, fabs, fmod, sinh, cosh, tanh, pow, atan2, ceil, floor and abs. Please note that the abs function works with integer numbers. All the other functions work with variables of type double.

**Vector functions** (not part of ANSI C)

```plaintext
double max([n,]val1,...,valn)
    Returns the maximum value. The first parameter defining the number of items is optional.
double max(n,vec)
    Returns the value of maximal item in the vec vector.
double min([n,]val1,...,valn)
    Returns the minimum value. The first parameter defining the number of items is optional.
double min(n,vec)
    Returns the value of minimal item in the vec vector.
double poly([n,]x,an,...,a1,a0)
    Evaluates the polynomial \( y = a_n x^n + \ldots + a_1 x + a_0 \). The first parameter defining the number of items is optional.
double poly(n,x,vec)
    Evaluates the polynomial \( y = \text{vec}[n] x^n + \ldots + \text{vec}[1] x + \text{vec}[0] \).
double scal(n,vec1,vec2)
    Evaluates the scalar product \( y = \text{vec1}[0] \text{vec2}[0] + \ldots + \text{vec1}[n-1] \text{vec2}[n-1] \).
double scal(n,vec1,vec2,skip1,skip2)
    Evaluates the scalar product \( y = \text{vec1}[0] \text{vec2}[n-1] + \text{vec1}[1] \text{vec2}[n-1] + \ldots + \text{vec1}[(n-1)*skip1] \text{vec2}[(n-1)*skip2] \). This is well suited for multiplication of matrices, which are stored as vectors (line by line or column by column).
double conv(n,vec1,vec2)
    Evaluates the convolutory product \( y = \text{vec1}[0] \text{vec2}[n-1] + \text{vec1}[1] \text{vec2}[n-1] + \ldots + \text{vec1}[n-1] \text{vec2}[0] \).
double sum(n,vec)
    Sums the items in a vector, i.e. \( y = \text{vec}[0] + \text{vec}[1] + \ldots + \text{vec}[n-1] \).
double sum([n,]val1,...,valn)
    Sums the items, i.e. \( y = \text{val1} + \text{val2} + \ldots + \text{valn} \). The first parameter defining the number of items is optional.
[]array([n,]an-1,...,a1,a0)
    Returns an array/vector with the given values. The first parameter defining the number of items is optional. The type of the returned value is chosen automatically to fit the type of parameters (all must be of the same type).
```
CHAPTER 15. SPEC – SPECIAL BLOCKS

[]subarray(idx, vec)
   Returns a subarray/subvector of the vec array, starting at the idx index.
   The type of the returned value is chosen automatically according to the vec array.

copyarray(count, vecSource, idxSource, vecTarget, idxTarget)
   Copies count items of the vecSource array, starting at idxSource index,
   to the vecTarget array, starting at idxTarget index. Both arrays must
   be of the same type.

void fillarray(vector, value, count)
   Copies value to count items of the vector array (always starting from
   index 0).

• String functions (ANSI C contains analogous functions in the string.h file)

string strsub(str, idx, len)
   Returns a substring of length len starting at index idx.

long strlen(str)
   Returns string length (number of characters).

long strfind(str, substr)
   Returns the position of first occurrence of substr in str.

long strrfind(str, substr)
   Returns the position of last occurrence of substr in str.

strreplace(str, pattern, substr)
   Find all occurrences of pattern in str and replace it with substr (in-place
   replacement, so new string is stored into str).

strupr(str)
   Converts a string to uppercase.

strlwr(str)
   Converts a string to lowercase.

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 [, radix])
   Converts an integer number num to text. The optional parameter radix
   specifies the numerical system in which the conversion is to be performed
   (typically 10 or 16). If radix is not specified, default value is radix = 10.
   The output string does not contain any identification of the numerical
   system used (e.g. the 0x prefix for the hexadecimal system).
string double2str(num)
    Converts a decimal number num to text.

strncpy(dest,src)
    Function copies the src string to the dest string. Implemented for compatibility with ANSI C. The construction dest=src yields the same result.

strcat(dest,src)
    Function appends a copy of the src string to the dest string. Implemented for compatibility with ANSI C. The construction dest=dest+src yields the same result.

strcmp(str1,str2)
    Function compares strings str1 and str2. Implemented for compatibility with ANSI C. The construction str1==str2 yields the same result.

float2buf(buf,x[,endian])
    Function for converting real number x into 4 elements of array buf. Each element represents an octet (byte) of number in single precision representation according to IEEE 754 (known as float). The function is useful for filling communication buffers. Optional 3rd parameter has the following meaning: 0 (default) = processor native endian, 1 = little endian, 2 = big endian.

double2buf(buf,x[,endian])
    Similar function to float2buf, but stores 8 elements in double precision format.

double buf2float(buf[,endian])
    Inverse function to float2buf

double buf2double(buf[,endian])
    Inverse function to double2buf

long RegExp(str,regexp,capture[])
    Compares the str string with regular expression regexp. When the string matches the pattern, the capture array contains individual sections of the regular expression. capture[0] is always the complete regular expression. The function return the number of captured strings or a negative value in case of an error. The regular expression may contain the following:

    (?i) ... Must be at the beginning of the regular expression. Makes the matching case-insensitive.
    ^ ... Match beginning of a string
    $ ... Match end of a string
    () ... Grouping and substring capturing
    \s ... Match whitespace
    \S ... Match non-whitespace
    \d ... Match decimal digit
    \n ... Match new line character
    \r ... Match line feed character
    \f ... Match vertical tab character
CHAPTER 15. SPEC – SPECIAL BLOCKS

\v ... Match horizontal tab character
\t ... Match horizontal tab character
\b ... Match backspace character
+ ... Match one or more times (greedy)
+? ... Match one or more times (non-greedy)
* ... Match zero or more times (greedy)
*? ... Match zero or more times (non-greedy)
? ... Match zero or once (non-greedy)
x|y ... Match x or y (alternation operator)
\meta ... Match one of the meta characters: ^$().\[\]*+?|\xHH ...
Match byte with hex value 0xHH, e.g. \x4a.
[... ] ... Match any character from the set. Ranges like [a-z are
supported.
[^... ] ... Match any character but the ones from the set.

long ParseJson (json,cnt,names[],values[])
The json string is supposed to contain text in JSON format. The names array
contain the requested objects (subitems are accessed via ., index of the
array via []). The values array then contains values of the requested ob-
jects. 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 REXY-
GEN.

In order to view these debugging messages in System log it is necessary to
enable them. Go to the menu
Target → Diagnostic messages and tick the Information checkbox in the
Function block messages box. Logging has to be also enabled for the par-
ticular block by ticking the Enable logging checkbox in the Runtime tab of
the block parameters dialog. By default, this is enabled after placing a new
block from library. Only then are the messages displayed in the System
log.
TraceError(id,val) TraceWarning(id,val) TraceVerbose(id,val)

These commands are similar to the Trace command, only the output is routed to the corresponding logging group (Error, Warning, Verbose). Messages with the Error level are always written to the log. To view the Warning andVerbose messages, enable the corresponding message group. Go to the menu Target→Diagnostic messages and tick the corresponding checkbox in the Function block messages box.

Suspend(sec)

The script is suspended if its execution within the given sampling period takes more seconds than specified by the sec parameter. At the next start of the block the script continues from the point where it was suspended. Use Suspend(0) to suspend the code immediately.

double GetPeriod()

Returns the sampling period of the block in seconds.

double CurrentTime()

Returns the current time (in internal format). Intended for use with the ElapsedTime() function.

double ElapsedTime(new_time, old_time)

Returns the elapsed time in seconds (decimal number), i.e. the difference between the two time values new_time and old_time. The CurrentTime() function is typically used in place of the new_time parameter.

double Random()

Returns a pseudo-random number from the ⟨0,1⟩ interval. The pseudo-random number generator is initialized prior to calling the init() function so the sequence is always the same.

long QGet(var)

Returns the quality of the var variable (see the QFC, QFD, VIN, VOUT blocks). The function is intended for use with the inputs, outputs and parameters. It always returns 0 for internal variables.

void QSet(var, value)

Sets the quality of the var variable (see the QFC, QFD, VIN, VOUT blocks). The function is intended for use with the inputs, outputs and parameters. It has no meaning for internal variables.

long QPropag([n,]val1,...,valn)

Returns the quality resulting from merging of qualities of val1,...,valn. The basic rule for merging is that the resulting quality correspond with the worst quality of val1,...,valn. To obtain the same behavior as in other blocks of the REXYGEN system, use this function to set the quality of output, use all the signals influencing the output as parameters.

double LoadValue(fileid, idx)

Reads a value from a file. A binary file with double values or a text file with values on individual lines is supposed. The idx index (binary file)
or line number (text file) starts at 0. The file is identified by `fileid`. At present the following values are supported:

0 ... file on a disk identified by the `p0` parameter
1 ... file on disk identified by name of the REXLANG block and extension `.dat`
2 ... file on a disk identified by the `srcname` parameter, but the extension is changed to `.dat`
3 ... `rexlang.dat` file in the current directory
4-7 ... same like 0-3, but format is text file. Each line contains one number. The index `idx` is the line number and starts at zero. Value `idx=-1` means next line (e.g. sequential writing).

```c
void SaveValue(fileid, idx, value)
```
Stores the `value` to a file. The meaning of parameters is the same as in the `LoadValue` function.

```c
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).

```c
void Sleep(seconds)
```
Stop execution of the block’s algorithm (and whole task) for defined time. Use this block with extreme caution and only if there is no other possibility to achieve the desired behaviour of your algorithm. The sleep interval should not exceed 900 milliseconds. The shortest interval is about 0.01s, the precise value depends on the target platform.

```c
long GetExtInt(ItemID)
```
Returns the value of input/output/parameter of arbitrary block in REXYGEN algorithm. Such an external data item is referenced by the `ItemID` parameter. The structure of the string parameter `ItemID` is the same as in e.g. the `sc` parameter of the `GETPI` function block. If the value cannot be obtained (e.g. invalid or non-existing `ItemID`, data type conflict, etc.), the REXLANG block issues an error and must be reset.

```c
long GetExtLong(ItemID)
```
See `GetExtInt(ItemID)`.

```c
double GetExtReal(ItemID)
```
Similar to `GetExtInt(ItemID)` but for decimal numbers.

```c
double GetExtDouble(ItemID)
```
See `GetExtReal(ItemID)`.

```c
string GetExtString(ItemID)
```
Similar to `GetExtInt(ItemID)` but for strings.
void SetExt(ItemID, value)
Sets the input/output/parameter of arbitrary block in REXYGEN algorithm to value. Such an external data item is referenced by the ItemID parameter. The structure of the string parameter ItemID is the same as in e.g. the sc parameter of the SETPI function block. The type of the external data item (long/double/string) must correspond with the type of the value parameter. If the value cannot be set (e.g. invalid or non-existing ItemID, data type conflict, etc.), the REXLANG block issues an error and must be reset.

long memrd32(hMem, offset)
Reading physical memory. Get the handle by Open(72,"/dev/mem",<physical address>,<area size>).

long memwr32(hMem, offset, value)
Writing to physical memory. Get the handle by OpenMemory("/dev/mem",<physical address>,<area size>).

- Communication functions (not part of ANSI C)
This set of functions is intended for communication over TCP/IP, UDP/IP, serial line (RS-232 or RS-485), SPI bus and I2C bus. Only a brief list of available functions is given below, see the example projects of the REXYGEN system for more details.

long OpenFile(string filename)
Function for opening a file. Identification number (the so-called handle) of the file is returned. If a negative value is returned, the opening was not successful.

long OpenCom(string comname, long baudrate, long parity)
Function for opening a serial line. Identification number (the so-called handle) of serial port is returned. If a negative value is returned, the opening was not successful. Parity setting: 0=none, 1=odd, 2=even.

long OpenUDP(string localname, long lclPort, string remotename, long remPort)
Function for opening a UDP socket. Identification number (the so-called handle) of the socket is returned. If a negative value is returned, the opening was not successful.

long OpenTCPsvr(string localname, long lclPort)
Function for opening a TCP socket (server, listening). Identification number (the so-called handle) of the socket is returned. If a negative value is returned, the opening was not successful.

long OpenTCPcli(string remotename, long remPort)
Function for opening a TCP socket (client). Identification number (the so-called handle) of the socket is returned. If a negative value is returned, the opening was not successful.

long OpenI2C(string devicename)
Function for opening the I2C bus. Identification number (the so-called handle) of the bus is returned. If a negative value is returned, the opening was not successful.
long OpenSPI(string devicename)
Function for opening the SPI bus. Identification number (the so-called handle) of the bus is returned. If a negative value is returned, the opening was not successful.

long OpenMemory(string devicename, long baseaddr, long size)
Function for mapping physical memory. Identification number (the so-called handle) of the memory address is returned. If a negative value is returned, the opening was not successful.

long Close(long handle)
Closes the socket, serial line, file or any device opened by the Open...

functions.

void SetOptions(long handle, long params[])
Sets the parameters of a socket or serial line. The array size must be at least:
- 22 for serial line,
- 2 for file (1st item is mode: 1=seek begin, 2=seek current, 3=seek end, 4=set file end, 2nd item is offset for seek),
- 3 for SPI (1st item is SPI mode, 2nd item is bits per word, 3rd item is max speed in Hz),
- 5 for I2C (1st item is slave address, 2nd item is 10-bit address flag, 3rd item is Packet Error Checking flag, 4th item is number of retries, 5th item is timeout)

void GetOptions(long handle, long params[])
Reads parameters of a socket or serial line to the params array. The array size must be big enough, at least 2 for files, 2 for a socket and 22 for serial line (see SetOptions).

long Accept(long hListen)
Accepts the connection to listening socket hListen invoked by the client. A communication socket handle or an error is returned.

long Read(long handle, long buffer[], long count)
Receives data from a serial line or socket. The count parameter defines the maximum number of bytes to read. The count of bytes read or an error code is returned. Each byte of incoming data is put to the buffer array of type long in the corresponding order.
It is also possible to use the form
long Read(long handle, string data[], long count) (i.e. a string is used instead of a data array; one byte in the input file corresponds to one character; not applicable to binary files).

The error codes are:
-1 it is necessary to wait for the operation to finish (the function is "non-blocking")
-309 reading failed; the operating system error code appears in the log (when function block logging is enabled)
-307 file/socket is not open
long Write(long handle, long buffer[], long count)
Sends the data to a serial line or socket. The count parameter defines the number of bytes to send. The count of bytes or an error code sent is returned. Each byte of outgoing data is read from the buffer array of type long in the corresponding order.
It is also possible to use the form
long Write(long handle, string data) (i.e. a string is used instead of a data array; one byte in the output file corresponds to one character; not applicable to binary files).
The error codes are:
-1 it is necessary to wait for the operation to finish (the function is "non-blocking")
-310 write failed; the operating system error code appears in the log (when function block logging is enabled)
-307 file/socket is not open

long ReadLine(long handle, string data)
Read one line from (text) file, serial line or socket; read characters are in the variable data up to allocated size of the string; the function return real size (number of bytes) of line or error code.

long DeleteFile(string filename)
Delete file. Return 0 if success; negative value is error code.

long RenameFile(string filename, string newfilename)
Rename file. Return 0 if success; negative value is error code.

long I2C(long handle, long addr, long bufW[], long cntW, long bufR[], long cntR)
Communication over the I2C bus. Works only in Linux operating system on devices with the I2C bus (e.g. Raspberry Pi). Sends and receives data to/from the slave device with address addr. The parameter handle is returned by the OpenI2C function, whose parameter defines the device name (according to the operating system). The parameter bufW is a buffer (an array) for the data which is sent out, cntW is the number of bytes to send out, bufR is a buffer (an array) for the data which comes in and cntR is the number of bytes to receive. The function returns 0 or an error code.

long SPI(long handle, 0, long bufW[], long cntW, long bufR[], long cntR)
Execution of one transaction over the SPI bus. Works only in Linux operating system on devices with the SPI bus (e.g. Raspberry Pi). The parameter handle is returned by the OpenSPI function, whose parameter defines the device name (according to the operating system). The second parameter is always 0 (reserved for internal use). The parameter bufW is a buffer (an array) for the data which is sent out, cntW is the number of bytes to send out, bufR is a buffer (an array) for the data which comes in and cntR is the number of bytes to receive. Note that SPI communication is full-duplex, therefore the resulting length of the SPI transaction is given by maximum of the cntW and cntR parameters, not their sum. The function returns 0 or an error code.
long Seek(long handle, long mode[], long offset)
Set position for Read/Write command. Parameter mode means: 1=offset from begin of the file, 2= offset from current position, 3=offset from end of the file.

long Recv(long handle, long buffer[], long count)
Obsolete function. Use Read instead.

long Send(long handle, long buffer[], long count)
Obsolete function. Use Write instead.

Remarks

- The data type of inputs u0..u15, outputs y0..y15 and parameters p0..p15 is determined during compilation of the source code according to the input, output and parameter definitions.

- All error codes < 0 require restarting of the REXYGEN executive. Of course it is necessary to remove the cause of the error first.

- WARNING! - The inputs and outputs of the block cannot be accessed within the init() function (the values of inputs are 0, outputs are not set).

- It is possible to include path in the srcname parameter. Otherwise the file is expected directly in the project directory or in the directories specified by the -I command line option of the REXYGEN Compiler compiler.

- All parameters of the vector functions are of type double (or array of type double). The only exception is the n parameter of type long. Note that the functions with one vector parameter exist in three variants:

double function(val1,...,valn)
Vector is defined as a sequence of values of type double.

double function(n,val1,...,valn)
Vector is defined as in the first case, only the first parameter defines the number of values – the size of the vector. This variant is compatible with the C compiler. The n parameter must be a number, not the so-called const variable and it must correspond with the number of the following elements defining the vector.

double function(n,vec)
The n parameter is an arbitrary expression of type long and defines the number of elements the function takes into account.

- The optional parameter n of the vector functions must be specified if the compatibility with C/C++ compiler is required. In such a case all the nonstandard functions must be implemented as well and the functions with variable number of parameters need to know the parameter count.
In all cases it is important to keep in mind that the vectors start at index 0 and that the array limits are not checked (just like in the C language). E.g. if double vec[10], x; is defined, the elements have indexes 0 to 9. The expression x=vec[10]; is neither a syntax nor runtime error, the value is not defined. More importantly, it is possible to write vec[11]=x, which poses a threat, because some other variable might be overwritten and the program works unexpectedly or even crashes.

- Only the parser error and line number are reported during compilation. This means a syntax error. If everything seems fine, the problem can be caused by identifier/keyword/function name conflict.

- All jumps are translated as relative, i.e. the corresponding code is restricted to 32767 instructions (in portable format for various platforms).

- All valid variables and temporary results are stored in the stack, namely:
  - Global variables and local static variables (permanently at the beginning of the stack)
  - Return addresses of functions
  - Parameters of functions
  - Local function variables
  - Return value of function
  - Temporary results of operations (i.e. the expression a=b+c; is evaluated in the following manner: b is stored in the stack, c is stored in the stack (it follows after b), the sum is evaluated, both values are removed from the stack and the result is stored in the stack)

Each simple variable (long or double) thus counts as one item in the stack. For arrays, only the size is important, not the type.

- The arrays are passed to the functions as a reference. This means that the parameter counts as one item in the stack and that the function works directly with the referenced array, not its local copy.

- If the stack size is not sufficient (less than space required for global variables plus 10), the stack size is automatically set to twice the size of the space required for the global variables plus 100 (for computations, function parameters and local variables in the case that only a few global variables are present).

- If basic debug level is selected, several checks are performed during the execution of the script, namely initialization of the values which are read and array index limits. Also a couple of uninitialized values are inserted in front of and at the back of each declared array. The NOP instructions with line number of the source file are added to the *.ill file.
CHAPTER 15. SPEC – SPECIAL BLOCKS

- If full debug is selected, additional check is engaged – the attempts to access invalid data range are monitored (e.g. stack overflow).

- The term instruction in the context of this block refers to an instruction of a processor-independent mnemocode. The mnemocode is stored in the *.ill file.

- The `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

<table>
<thead>
<tr>
<th>HLD</th>
<th>Hold – the block code is not executed if the input is set to on</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET</td>
<td>Rising edge resets the block. The block gets initialized again (all global variables are cleared and the <code>Init()</code> function is called).</td>
<td>bool</td>
</tr>
<tr>
<td>u0..u15</td>
<td>Input signals which are accessible from the script</td>
<td>unknown</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>iE</th>
<th>Runtime error code. Unless iE = 0 or iE = -1 the algorithm is stopped until it is reinitialized by the <code>RESET</code> input or by restarting the executive</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error occurred, the whole <code>main()</code> function was executed (also the <code>init()</code> function).</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>The execution was suspended using the <code>Suspend()</code> command, i.e. the execution will resume as soon as the <code>REXLANG</code> block is executed again</td>
<td></td>
</tr>
<tr>
<td>xxx</td>
<td>Error code of the <code>REXYGEN</code> system, see Appendix C</td>
<td></td>
</tr>
<tr>
<td>y0..y15</td>
<td>Output signals which can be set from within the script</td>
<td>unknown</td>
</tr>
</tbody>
</table>

Parameters

| srcname | Source file name | srcfile.c string |
srcType: Coding of source file

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.

stack: Stack size defined as number of variables. Default and recommended value is 0, which enables automatic estimation of the necessary stack size.

dbg: Debug level – checking is safer but slows down the execution of the algorithm. Option No check can crash REXYGEN application on target platform if code is incorrect.

strs: Total size of buffer for strings. Enter the maximum number of characters to allocate memory for. The default value 0 means that the buffer size is determined automatically.

p0..p15: Parameters which are accessible from the script

Example C-like

The following example shows a simple code to sum two input signals and also sum two user-defined parameters.

double input(0) input_u0;
double input(2) input_u2;

double parameter(0) param_p0;
double parameter(1) param_p1;

double output(0) output_y0;
double output(1) output_y1;

double my_value;

long init(void)
CHAPTER 15. SPEC – SPECIAL BLOCKS

{  
    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

MQTT – Communication via MQTT protocol

Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MqttPublish – Publish MQTT message</td>
<td>458</td>
</tr>
<tr>
<td>MqttSubscribe – Subscribe to MQTT topic</td>
<td>460</td>
</tr>
</tbody>
</table>
MqttPublish – Publish MQTT message

Function Description

This function block depends on the MQTT driver. Please read the MQTTDrv manual \[10\] before use.

The MqttPublish block publishes messages to an MQTT broker through the connection established by the MQTTDrv driver.

The first parameter is the topic the block will publish the messages to. MQTT delivers Application Messages according to the Quality of Service (QoS) levels. Use the QoS parameter to set a different Quality of Service level. See the MQTT specification \[11\] for more details.

If the RETAIN parameter is set a RETAIN flag will be set on the outgoing PUBLISH Control Packet. See the MQTT specification \[11\] for more details.

The defBuffSize parameter can be used to optimize the memory usage of the block. It states the amount of the statically allocated memory for the inner buffer for the outgoing messages. If the value is unnecessarily large the memory is being wasted. On the other hand if the value of the parameter is too small it leads to frequent dynamic memory allocations which can be time consuming.

The message to be published is constructed from the value input signal. The value input signal is expected to be a string. If it is not a string it will be converted automatically. To request a message to be published in the current period set the RUN flag to on. The BUSY flag is on if the block has a pending request and waits for a response from a broker. When the response is received in the current cycle the DONE flag is set to on.

Inputs

- value: Input signal, string
- RUN: Enable execution, bool

Parameters

- topic: MQTT topic, string
QoS | Quality of Service | 1 long
---|-------------------|-----------------
1   | QoS0 (At most once) |  
2   | QoS1 (At least once) |  
3   | QoS2 (Exactly once) |  
RETAIN | Retain last message | ⊠on bool
defBuffSize | Default buffer size | ↓1 ▽2048 long

Outputs

<table>
<thead>
<tr>
<th>BUSY</th>
<th>Busy flag</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>DONE</td>
<td>Indicator of finished transaction</td>
<td>bool</td>
</tr>
<tr>
<td>errId</td>
<td>Error code</td>
<td>error</td>
</tr>
</tbody>
</table>
MqttSubscribe – Subscribe to MQTT topic

Block Symbol

License: MQTT

Function Description

This function block depends on the MQTT driver. Please read the MQTTDrv manual [10] before use.

The MqttSubscribe block subscribes to a topic on an MQTT broker and receives Publish messages on that topic through the connection established by the MQTTDrv driver.

The first parameter is the topic the block will subscribe to. MQTT protocol delivers Application Messages according to the Quality of Service (QoS) levels. Use the QoS parameter to set a different Quality of Service level. See the MQTT specification [11] for more details.

By setting the type parameter of the block it can be specified the expected data type of the incoming message. The block converts the incoming message to the specified type and sets the value output signal in case of success or it sets the errId to the resulting error code.

The mode parameter has two available options: Last value and Buffered values. If Last value mode is used the block will always output only the last message received even if multiple messages were received in the last period. If the mode is set to Buffered values than the block buffers the incoming messages and outputs one by one in consecutive ticks of the task.

The defBuffSize parameter can be used to optimize the memory usage of the block. It states the amount of the statically allocated memory in the inner buffer for the incoming messages. If the value is unnecessarily large the memory is being wasted. On the other hand if the value of the parameter is too small it leads to frequent dynamic memory allocations which can be time consuming.

A Subscribe action is performed upon a rising edge (off→on) and an Unsubscribe action is performed upon a falling edge (on→off) at the RUN input.

The nDRDY output specifies how many messages were received and are available in the inner buffer. If the mode of the block is set to Last value the nDRDY output can only have value 0 or 1.

The RETAIN output flag is set if the received Publish packet had the RETAIN flag set. See the MQTT specification [11] for more details.
Note that subscribing to topics containing wildcards is not supported.

**Input**

<table>
<thead>
<tr>
<th><strong>RUN</strong></th>
<th>Enable execution</th>
<th><strong>bool</strong></th>
</tr>
</thead>
</table>

**Parameters**

<table>
<thead>
<tr>
<th><strong>topic</strong></th>
<th>MQTT topic</th>
<th><strong>string</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>QoS</strong></td>
<td>Quality of Service</td>
<td><strong>⊙1 long</strong></td>
</tr>
<tr>
<td>1</td>
<td>QoS0 (At most once)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>QoS1 (At least once)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>QoS2 (Exactly once)</td>
<td></td>
</tr>
<tr>
<td><strong>type</strong></td>
<td>Expected type of incoming data</td>
<td><strong>⊙1 long</strong></td>
</tr>
<tr>
<td>1</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>bool</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>byte vector/blob</td>
<td></td>
</tr>
<tr>
<td><strong>mode</strong></td>
<td>Incoming messages buffering mode</td>
<td><strong>⊙1 long</strong></td>
</tr>
<tr>
<td>1</td>
<td>Last value</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Buffered values</td>
<td></td>
</tr>
<tr>
<td><strong>defBuffSize</strong></td>
<td>Default buffer size</td>
<td><strong>1 ⊙2048 long</strong></td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th><strong>value</strong></th>
<th>Output signal</th>
<th><strong>unknown</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>nDRDY</strong></td>
<td>Number of received messages</td>
<td><strong>0 ↑10 long</strong></td>
</tr>
<tr>
<td><strong>errId</strong></td>
<td>Error code</td>
<td><strong>error</strong></td>
</tr>
</tbody>
</table>
Chapter 17

CanDrv – Communication via CAN bus

Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanItem</td>
<td>Secondary received CAN message</td>
<td>464</td>
</tr>
<tr>
<td>CanRecv</td>
<td>Receive CAN message</td>
<td>465</td>
</tr>
<tr>
<td>CanSend</td>
<td>Send CAN message</td>
<td>467</td>
</tr>
</tbody>
</table>
CanItem – Secondary received CAN message

Block Symbol

Licence: CANDRV

Function Description

The block is used with the CanRecv block. The uRef input of the CanItem block must be connected to the itemRef output of some CanRecv block or to the yRef output of another CanItem block.

This block shows the previous message that has passed the filter in the CanRecv block.

If more than one CanItem block is connected (directly or indirectly through the yRef output of the CanItem block already connected to the CanRecv block) then the first executed CanItem block shows the first message before the last received message (which is shown by the CanRecv block), the second executed CanItem block shows the second message before the last received message (which is shown by the CanRecv block) etc. It is strongly recommended to connect the CanItem blocks in a daisy chain. Unexpected ordering of messages may occur if the blocks are connected in a tree-like structure.

If no message has been received since start of the CAN driver, the data outputs have fallback values msgId = -1 and length = -1.

The DRDY output is set to DRDY = on if the message has been received during the last period, i.e. after previous execution of the CanItem block. At the same moment, the outputs msgId, data and length are updated. If there is no new data, DRDY output is set to DRDY = off and the data values are kept on the other outputs (msgId, data and length).

Input

<table>
<thead>
<tr>
<th>uRef</th>
<th>Secondary received packet reference</th>
</tr>
</thead>
</table>

Outputs

<table>
<thead>
<tr>
<th>yRef</th>
<th>Secondary received packet reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgId</td>
<td>CAN message ID (COB-ID)</td>
</tr>
<tr>
<td>data</td>
<td>Message data (8 bytes maximum, LSB first)</td>
</tr>
<tr>
<td>length</td>
<td>Message length (number of bytes of data)</td>
</tr>
<tr>
<td>DRDY</td>
<td>Received message in the last period flag</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{msgId} & = -1 \\
\text{length} & = -1 \\
\text{DRDY} & = \text{on} \\
\end{align*}
\]

\[
\begin{align*}
\text{data} & = -9,22337E+18 \\
\text{length} & = 0 \text{ to } 8 \\
\end{align*}
\]
CanRecv – Receive CAN message

Function Description

The CanRecv block receives message via CAN bus. The message is defined by the msgId, data and length inputs and the RTR and EXT parameters.

Number of messages received in the current task period (i.e. since the previous execution) is indicated by the nDRDY output.

The data from the last received message is available at the msgId, data and length outputs. Previous messages (with respect to the nmax parameter) are available using the CanItem block(s) linked to the itemRef output.

The block must be linked with the CanDrv driver. The driver must be configured to use the simple CAN mode (i.e. the parameter NodeMode = 256).

The block’s name must be in the form <DRV>_<blkname> (see e.g. OUTQUAD or OUTOCT blocks for details about referencing data from I/O drivers). The <blkname> part of the name has no special meaning in this case and it is recommended to keep the original CanRecv.

The block supports short (11-bit) and long (29-bit) message IDs (see the EXT parameter) and RequestToReceive messages (see the RTR parameter). FD mode which allows up to 64 data bytes in a single message is not supported.

Outputs

| itemRef | Secondary received packet reference | reference |
| msgId   | CAN message ID (COB-ID)            | long      |
| data    | Message data (8 bytes maximum, LSB first) | large |
| length  | Message length (number of bytes of data) | ↓9,22337E+18 ↑9,22337E+18 |
| nDRDY   | Number of received messages in the last period | ↑255 word |
| iErr    | Error code                        | error     |
| age     | Elapsed time since the last received message [s] | ↓0.0 double |
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filterId</code></td>
<td>Message ID of packets to receive by this block</td>
<td>long</td>
</tr>
<tr>
<td><code>filterIdMask</code></td>
<td>Mask for the <code>filterId</code> parameter (marks valid bits)</td>
<td>long</td>
</tr>
<tr>
<td><code>filterLength</code></td>
<td>Data length of packets to receive by this block (-1 allows all lengths)</td>
<td>long</td>
</tr>
<tr>
<td><code>RTR</code></td>
<td>Request To Receive flag</td>
<td>bool</td>
</tr>
<tr>
<td><code>EXT</code></td>
<td>Extended message ID (29 bits)</td>
<td>bool</td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>Error is indicated if no packet is received within the timeout interval</td>
<td>double</td>
</tr>
<tr>
<td><code>nmax</code></td>
<td>Maximum number of received messages in one period</td>
<td>long</td>
</tr>
</tbody>
</table>
CanSend – Send CAN message

Block Symbol

\[
\begin{array}{c}
\text{msgId} \\
\text{len} \\
\text{data} \\
\text{RUN} \\
\text{iErr}
\end{array}
\]

Licence: CANDRV

Function Description

The CanSend block sends message via CAN bus. The message content is defined by the msgId, data and length inputs and the RTR and EXT parameters. Message is sent only if the input RUN is set to RUN = on.

The block must be linked with the CanDrv driver. The driver must be configured to use the simple CAN mode (i.e. the parameter NodeMode = 256).

The block’s name must be in the form <DRV>__<blkname> (see e.g. OUTQUAD or OUTOCT blocks for details about referencing data from I/O drivers). The <blkname> part of the name has no special meaning in this case and it is recommended to keep the original CanSend.

The block supports short (11-bit) and long (29-bit) message IDs (see the EXT parameter) and RequestToReceive messages (see the RTR parameter). FD mode which allows up to 64 data bytes in a single message is not supported.

Inputs

<table>
<thead>
<tr>
<th></th>
<th>CAN message ID (COB-ID)</th>
<th>(0 \uparrow536870911) long</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgId</td>
<td>Message length (number of bytes of data)</td>
<td>(0 \uparrow8) long</td>
</tr>
<tr>
<td>length</td>
<td>Message data (8 bytes maximum, LSB first)</td>
<td>(-9,22337E+18 \uparrow9,22337E+18) large</td>
</tr>
<tr>
<td>data</td>
<td>Sending message is enabled</td>
<td>bool</td>
</tr>
<tr>
<td>RUN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th></th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>iErr</td>
<td>error</td>
</tr>
</tbody>
</table>

Parameters

|   | Request To Receive flag | \(\circ \text{on} \) bool |
| RTR | Extended message ID (29bits) | \(\circ \text{on} \) bool |
| EXT |
Chapter 18

MC_SINGLE – Motion control - single axis blocks

Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM_Axis</td>
<td>Motion control axis</td>
<td>472</td>
</tr>
<tr>
<td>MC_AccelerationProfile, MCP_AccelerationProfile</td>
<td>Acceleration profile</td>
<td>478</td>
</tr>
<tr>
<td>MC_Halt, MCP_Halt</td>
<td>Stopping a movement (interruptible)</td>
<td>482</td>
</tr>
<tr>
<td>MC_HaltSuperimposed, MCP_HaltSuperimposed</td>
<td>Stopping a movement (superimposed and interruptible)</td>
<td>483</td>
</tr>
<tr>
<td>MC_Home, MCP_Home</td>
<td>Homing</td>
<td>484</td>
</tr>
<tr>
<td>MC_MoveAbsolute, MCP_MoveAbsolute</td>
<td>Move to position (absolute coordinate)</td>
<td>486</td>
</tr>
<tr>
<td>MC_MoveAdditive, MCP_MoveAdditive</td>
<td>Move to position (relative to previous motion)</td>
<td>489</td>
</tr>
<tr>
<td>MC_MoveRelative, MCP_MoveRelative</td>
<td>Move to position (relative to execution point)</td>
<td>492</td>
</tr>
<tr>
<td>MC_MoveSuperimposed, MCP_MoveSuperimposed</td>
<td>Superimposed move</td>
<td>495</td>
</tr>
<tr>
<td>MC_MoveContinuousAbsolute, MCP_MoveContinuousAbsolute</td>
<td>Move to position (absolute coordinate)</td>
<td>498</td>
</tr>
<tr>
<td>MC_MoveContinuousRelative, MCP_MoveContinuousRelative</td>
<td>Move to position (relative to previous motion)</td>
<td>501</td>
</tr>
<tr>
<td>MC_MoveVelocity, MCP_MoveVelocity</td>
<td>Move with constant velocity</td>
<td>505</td>
</tr>
<tr>
<td>MC_PositionProfile, MCP_PositionProfile</td>
<td>Position profile</td>
<td>509</td>
</tr>
<tr>
<td>MC_Power</td>
<td>Axis activation (power on/off)</td>
<td>513</td>
</tr>
<tr>
<td>MC_ReadActualPosition</td>
<td>Read actual position</td>
<td>514</td>
</tr>
<tr>
<td>MC_ReadAxisError</td>
<td>Read axis error</td>
<td>515</td>
</tr>
<tr>
<td>MC_ReadBoolParameter, MCP_ReadBoolParameter</td>
<td>Read axis parameter (bool)</td>
<td>516</td>
</tr>
<tr>
<td>MC_ReadParameter</td>
<td>Read axis parameter</td>
<td>517</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 **REXYGEN**, block parameters are used to simplify usage of the blocks. To keep compatibility with **PLCopen** and improve usability of the blocks, almost all of them are implemented twice: with prefix **MC_** without parameters (parameters are inputs) and with prefix **MCP_** with parameters. Some blocks require additional **vendor specific** parameters. In such a case even the **MC_**-prefixed blocks contain parameters.

**PLCopen** specifies that all inputs/parameters are sampled at rising-edge of the **Execute** input. In **REXYGEN** block parameters are usually changed very rare. Therefore the parameters of the activated block have not be changed until block is finished (e.g. while output **Busy** is on).

The **REXYGEN** system does not allow input-output signals and all signals must have different name. For these reasons the **Axis** input-output signal, which is used in all blocks, is divided into input **uAxis** and output **yAxis**. The block algorithm copies the input **uAxis** to the output **yAxis**. The **yAxis** output is not necessary for the function of motion control blocks, but "chaining" the axis references makes it possible to order the blocks and define priorities. Other reference signals are either defined as input-only or use this mechanism as well.

**PLCopen** defines the outputs **Busy**, **Active**, **CommandAborted** as optional in almost all blocks. In **REXYGEN**, some of them are never set, but the outputs are defined to simplify future extensions and/or changes in the implementation.

Units used for position and distance of axis are implementation specific. It can be meters, millimeters, encoder ticks, angular degrees (for rotational axis) or any others.

<table>
<thead>
<tr>
<th>Function Block</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MC_ReadStatus</strong></td>
<td>Read axis status</td>
<td>519</td>
</tr>
<tr>
<td><strong>MC_Reset</strong></td>
<td>Reset axis errors</td>
<td>521</td>
</tr>
<tr>
<td><strong>MC_SetOverride</strong>, <strong>MCP_SetOverride</strong></td>
<td>Set override factors</td>
<td>522</td>
</tr>
<tr>
<td><strong>MC_Stop</strong>, <strong>MCP_Stop</strong></td>
<td>Stopping a movement</td>
<td>524</td>
</tr>
<tr>
<td><strong>MC_TorqueControl</strong>, <strong>MCP_TorqueControl</strong></td>
<td>Torque/force control</td>
<td>526</td>
</tr>
<tr>
<td><strong>MC_VelocityProfile</strong>, <strong>MCP_VelocityProfile</strong></td>
<td>Velocity profile</td>
<td>529</td>
</tr>
<tr>
<td><strong>MC_WriteBoolParameter</strong></td>
<td>Write axis parameter (bool)</td>
<td>533</td>
</tr>
<tr>
<td><strong>MC_WriteParameter</strong></td>
<td>Write axis parameter</td>
<td>534</td>
</tr>
<tr>
<td><strong>RM_AxisOut</strong></td>
<td>Axis output</td>
<td>535</td>
</tr>
<tr>
<td><strong>RM_AxisSpline</strong></td>
<td>Commanded values interpolation</td>
<td>537</td>
</tr>
<tr>
<td><strong>RM_Track</strong></td>
<td>Tracking and inching</td>
<td>539</td>
</tr>
</tbody>
</table>
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 REXGEN 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

Function Description

The RM_AXIS block is a cornerstone of the motion control solution within the REXYGEN system. This base block keeps all status values and implements basic algorithm for one motion control axis (one motor), which includes limits checking, emergency stop, etc. The block is used for both real and virtual axes. The real axis must have a position feedback controller, which is out of this block’s scope. The key status values are commanded position, velocity, acceleration and torque, as well as state of the axis, axis error code and a reference to the block, which controls the axis.

This block (like all blocks in the motion control library) does not implement a feedback controller which would keep the actual position as near to the commanded position as possible. Such a controller must be provided by using other blocks (e.g. PIDU) or external (hardware) controller. The feedback signals are used for lag checking, homing and could be used in special motion control blocks.

The parameters of this block correspond with the requirements of the PLCopen standard for an axis. If improper parameters are set, the errorID output is set to -700 (invalid parameter) and all motion blocks fail with the -720 error code (general failure).

Note that the default values for position, velocity and acceleration limits are intentionally set to 0, which makes them invalid. Limits must always be set by the user according to the real axis and the axis actuator.

Inputs

<table>
<thead>
<tr>
<th>HLD</th>
<th>Hold</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>...</td>
<td>Motion is allowed</td>
</tr>
<tr>
<td>on</td>
<td>.....</td>
<td>Axis is halted and no motion is possible</td>
</tr>
</tbody>
</table>

| ActualPos | Current position of the axis (feedback) | double |
| ActualVelocity | Current velocity of the axis (feedback) | double |
| ActualTorque | Current torque in the axis (feedback) | double |
| LIMN       | Limit switch in negative direction | bool |
| LIMZ       | Absolute switch or reference pulse for homing | bool |
LIMP

Limit switch in positive direction

bool

Outputs

axisRef Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)

CommandedPosition Requested (commanded) position of the axis
double

CommandedVelocity Requested (commanded) velocity of the axis
double

CommandedAcceleration Requested (commanded) acceleration of the axis
double

CommandedTorque Requested (commanded) torque in the axis
double

iState State of the axis
long

0 .... Disabled
1 .... Stand still
2 .... Homing
3 .... Discrete motion
4 .... Continuous motion
5 .... Synchronized motion
6 .... Coordinated motion
7 .... Stopping
8 .... Error stop

ErrorID Result of the last operation
error

1 .... REXYGEN general error

Parameters

ReturnType 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

MaxPosSystem Positive position limit for system
double

EnableLimitNeg Enable negative position limit checking
bool

MaxNegLimit Neg Negative position limit for application (MC blocks)
double

MinPosNeg System 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)
double

MaxJerk Maximal allowed jerk [unit/s³]
double
MaxTorque Maximal motor torque/force (0=not used) double
kta Torque-Acceleration ratio double
ReverseLimit Invert meaning of LIMN, LIMZ and LIMP inputs bool

Example
Following example illustrates basic principle of use of motion control blocks. It presents the minimal configuration which is needed for operation of a physical or virtual axis. The axis is represented by RM_Axis block. The limitations imposed on the motion trajectory in form of maximum velocity, acceleration, jerk and position have to be set in parameters of the RM_Axis block. The inputs can be connected to supply the values of actual position, speed and torque (feedback for slip monitoring) or logical limit switch signals for homing procedure. The axisRef output signal needs to be connected to any motion control block related to the corresponding axis. The axis has to be activated by enabling the MC_Power block. The state of the axis changes from Disabled to Standstill (see the following state transition diagram) and any discrete, continuous or synchronized motion can be started by executing a proper functional block (e.g. MC_MoveAbsolute). The trajectory of motion in form of desired position, velocity and acceleration is generated in output signals of the RM_Axis block. The reference values are provided to an actuator control loop which is implemented locally in REXYGEN system in the same or different task or they are transmitted via a serial communication interface to end device which controls the motor motion (servo amplifier, frequency inverter etc.). In case of any error, the axis performs an emergency stop and indicates the error ID. The error has to be confirmed by executing the MC_Reset block prior to any subsequent motion command. The following state diagram demonstrates the state transitions of an axis.
Axis state transition diagram

Note 1: From any state. An error in the axis occurred.
Note 2: From any state. MC_Power.Enable = FALSE and there is no error in the axis.
Note 3: MC_Reset AND MC_Power.Status = FALSE
Note 6: MC_Stop.Done = TRUE AND MC_Stop.Execute = FALSE
Motion blending

According to **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><strong>Aborting</strong></td>
<td>The new motion is executed immediately</td>
</tr>
<tr>
<td><strong>Buffered</strong></td>
<td>the new motion is executed immediately after finishing the previous one, there is no blending</td>
</tr>
<tr>
<td><strong>Blending low</strong></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><strong>Blending high</strong></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><strong>Blending previous</strong></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><strong>Blending next</strong></td>
<td>the new motion is executed immediately after finishing the previous one, but the axis will not stop between the movements, the first motion ends with the limit for maximum velocity of second block at the first end-position</td>
</tr>
</tbody>
</table>
Illustration of blending modes

Aborting

Buffered

Blending low

Blending high

Blending next

Blending previous

Commanded velocity

Active block 1 = false/Active block 2 = true

Value of the maximum velocity $v_1 = 30$ (block 1)

Value of the maximum velocity $v_2 = 15$ (block 2)
MC_AccelerationProfile, MCP_AccelerationProfile – Acceleration profile

Block Symbols

Function Description

The MC_AccelerationProfile and MCP_AccelerationProfile blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The MC_PositionProfile block commands a time-position locked motion profile. Block implements two possibilities for definition of time-acceleration function:

1. sequence of values: the user defines a sequence of time-acceleration pairs. In each time interval, the values of velocity are interpolated. Times sequence is in array \textit{times}, position sequence is in array \textit{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 \( p(x) = a_5 x^5 + a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0 \) 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 \textit{values} in ascending order (e.g. array \textit{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 \textit{times} includes only the first (usually zero) and last point.

Note 1: input \textit{TimePosition} is missing, because all path data are in parameters of the block.

Note 2: parameter \textit{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 reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>TimeScale</td>
<td>Overall scale factor in time</td>
</tr>
<tr>
<td>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>
</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)

Outputs

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM_Axis.axisRef−uAxis or yAxis−uAxis reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
</tbody>
</table>

1 .... REXYGEN general error
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 . . . . . . . . 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>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>
Example
MC_Halt, MCP_Halt — Stopping a movement (interruptible)

Block Symbols

Function Description

The MC_Halt and MCP_Halt blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The MC_Halt block commands a controlled motion stop and transfers the axis to the state DiscreteMotion. After the axis has reached zero velocity, the Done output is set to true immediately and the axis state is changed to Standstill.

Note 1: Block MC_Halt is intended for temporary stop of an axis under normal working conditions. Any next motion command which cancels the MC_Halt can be executed in nonbuffered mode (opposite to MC_Stop, which cannot be interrupted). The new command can start even before the stopping sequence was finished.

Inputs

| uAxis       | Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed) |
| Execute     | The block is activated on rising edge                                              |
| Deceleration| Maximal allowed deceleration [unit/s^2]                                           |
| Jerk        | Maximal allowed jerk [unit/s^3]                                                   |

Outputs

| yAxis       | Axis reference (only RM_Axis.axisRef-yAxis or yAxis-yAxis connections are allowed) |
| Done        | Algorithm finished                                                               |
| CommandAborted | Algorithm was aborted                                                           |
| Busy        | Algorithm not finished yet                                                       |
| Active      | The block is controlling the axis                                               |
| Error       | Error occurred                                                                   |
| ErrorID     | Result of the last operation                                                    |

i ..... REXYGEN general error
MC_HaltSuperimposed, MCP_HaltSuperimposed – Stopping a movement (superimposed and interruptible)

Block Symbols

<table>
<thead>
<tr>
<th>MC_HaltSuperimposed</th>
<th>MCP_HaltSuperimposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>yAxis</td>
</tr>
<tr>
<td>Execute</td>
<td>CommandAborted</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Busy</td>
</tr>
<tr>
<td>Jerk</td>
<td>Error</td>
</tr>
<tr>
<td>ErrorID</td>
<td></td>
</tr>
<tr>
<td>uAxis</td>
<td>yAxis</td>
</tr>
<tr>
<td>Execute</td>
<td>CommandAborted</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Busy</td>
</tr>
<tr>
<td>Jerk</td>
<td>Error</td>
</tr>
<tr>
<td>ErrorID</td>
<td></td>
</tr>
</tbody>
</table>

Licence: MOTION CONTROL

Function Description

*The MC_HaltSuperimposed and MCP_HaltSuperimposed blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.*

Block MC_HaltSuperimposed commands a halt to all superimposed motions of the axis. The underlying motion is not interrupted.

Inputs

- **uAxis** Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis reference connections are allowed)
- **Execute** The block is activated on rising edge
- **Deceleration** Maximal allowed deceleration [unit/s²]
- **Jerk** Maximal allowed jerk [unit/s³]

Outputs

- **yAxis** Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis reference connections are allowed)
- **Done** Algorithm finished
- **CommandAborted** Algorithm was aborted
- **Busy** Algorithm not finished yet
- **Active** The block is controlling the axis
- **Error** Error occurred
- **ErrorID** Result of the last operation
  - i . . . . . REXYGEN general error

ErrorID
MC_Home, MCP_Home — Homing

Block Symbols

Function Description

The MC_Home and MCP_Home blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The MC_Home block commands the axis to perform the "search home" sequence. The details of this sequence are described in PLCopen and can be set by parameters of the block. The "Position" input is used to set the absolute position when reference signal is detected. This Function Block completes at "StandStill".

Note 1: Parameter/input BufferMode is not supported. Mode is always Aborting. It is not limitation, because homing is typically done once in initialization sequence before some regular movement is proceeded.

Note 2: Homing procedure requires some of RM_Axis block input connected. Depending on homing mode, ActualPos, ActualTorque, LimP, LimZ, LimN can be required. It is expected that only one method is used. Therefore, there are no separate inputs for zero switch and encoder reference pulse (both must be connected to LimZ).

Note 3: HomingMode=4(Direct) only sets the actual position. Therefore, the MC_SetPosition block is not implemented. HomingMode=5(Absolute) only switches the axis from state Homing to state StandStill.

Note 4: Motion trajectory for homing procedure is implemented in simpler way than for regular motion commands - acceleration and deceleration is same (only one parameter) and jerk is not used. For extremely precise homing (position set), it is recommended to run homing procedure twice. First, homing procedure is run with "high" velocity to move near zero switch, then small movement (out of zero switch) follows and finally second homing procedure with "small" velocity is performed.

Note 5: HomingMode=6(Block) detect home-position when the actual torque reach value in parameter Torque Limit or position lag reach value in parameter MaxPositionLag.
in attached RM_Axis block (only if the parameter has positive value).

**Inputs**

- **uAxis**: Axis reference (only RM_Axis.axisRef→uAxis or yAxis→uAxis reference connections are allowed)
- **Execute**: The block is activated on rising edge
- **Velocity**: Maximal allowed velocity [unit/s]
- **Acceleration**: Maximal allowed acceleration [unit/s²]
- **TorqueLimit**: Maximal allowed torque/force
- **TimeLimit**: Maximal allowed time for the whole algorithm [s]
- **DistanceLimit**: Maximal allowed distance for the whole algorithm [unit]
- **Position**: Requested target position (absolute) [unit]
- **Direction**: Direction of movement (cyclic axis or special case only)
  - 1 .... Positive
  - 2 .... Shortest
  - 3 .... Negative
  - 4 .... Current
- **HomingMode**: Homing mode algorithm
  - 1 .... Absolute switch
  - 2 .... Limit switch
  - 3 .... Reference pulse
  - 4 .... Direct (user reference)
  - 5 .... Absolute encoder
  - 6 .... Block

**Outputs**

- **yAxis**: Axis reference (only RM_Axis.axisRef→uAxis or yAxis→uAxis reference connections are allowed)
- **Done**: Algorithm finished
- **CommandAborted**: Algorithm was aborted
- **Busy**: Algorithm not finished yet
- **Active**: The block is controlling the axis
- **Error**: Error occurred
- **ErrorID**: Result of the last operation
  - i .... REXYGEN general error
MC_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 act almost same as `MC_MoveRelative`. The only difference is the final position is computed adding input `Distance` to final position of the previous block.

The `MC_MoveSuperimposed` block acts almost the same as the `MC_MoveRelative`
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>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>AAxis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis reference 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>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/s^2]</td>
<td>double</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s^2]</td>
<td>double</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s^3]</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>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>reference</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
<td>bool</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
<td>bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>error</td>
</tr>
<tr>
<td></td>
<td>i . . . . . . REXYGEN general error</td>
<td></td>
</tr>
</tbody>
</table>
Example

```
MC_MoveAbsolute
block 1
uAxis
Execute
Position
Velocity
Acceleration
Deceleration
Jerk
BufferMode
Direction
 Done
CommandAborted
Busy
Active
Error
ErrorID

MC_MoveAbsolute
block 2
yAxis
Execute
Position
Velocity
Acceleration
Deceleration
Jerk
BufferMode
Direction
 Done
CommandAborted
Busy
Active
Error
ErrorID
```

![Diagram of motion control blocks with parameters](image-url)
**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
- **Distance** Requested target distance (relative to start point) [unit]
- **Jerk** Maximum jerk
- **BufferMode** Buffer mode
- **ErrorID** Error ID
- **yAxis** Axis reference (only **RM_Axis.axisRef-yAxis** or **uAxis-yAxis** reference connections are allowed)
- **Done** Completion flag
- **CommandAborted** Command aborted flag
- **Busy** Busy flag
## 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

<table>
<thead>
<tr>
<th><code>yAxis</code></th>
<th>Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Done</code></td>
<td>Algorithm finished                                                                            bool</td>
</tr>
<tr>
<td><code>CommandAborted</code></td>
<td>Algorithm was aborted                                                                       bool</td>
</tr>
<tr>
<td><code>Busy</code></td>
<td>Algorithm not finished yet                                                                    bool</td>
</tr>
<tr>
<td><code>Active</code></td>
<td>The block is controlling the axis                                                             bool</td>
</tr>
<tr>
<td><code>Error</code></td>
<td>Error occurred                                                                                bool</td>
</tr>
<tr>
<td><code>ErrorID</code></td>
<td>Result of the last operation                                                                 error</td>
</tr>
<tr>
<td>i . . . .</td>
<td>REXYGEN general error</td>
</tr>
</tbody>
</table>
Example
MC_MoveRelative, MCP_MoveRelative — Move to position (relative to execution point)

Function Description

The MC_MoveRelative and MCP_MoveRelative blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_version of the block.

The MC_MoveRelative block moves an axis to specified position as fast as possible. The final position is determined by adding the value of Distance parameter to the actual position at the moment of triggering the Execute input. If no further action is pending, final velocity is zero (axis moves to position and stops) otherwise it depends on blending mode. For blending purposes, start and stop velocity of this block is maximum velocity with direction respecting current and final position. If start velocity of next pending block is in opposite direction, then blending velocity is always zero.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

Inputs

- **uAxis** Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)
- **Execute** The block is activated on rising edge [bool]
- **Distance** Requested target distance (relative to execution point) [unit] [double]
<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 (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 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 commands)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 . . . . . Blending previous (start after finishing the previous motion,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>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 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 RM_Axis.axisRef–uAxis or yAxis–uAxis reference</td>
<td></td>
</tr>
<tr>
<td></td>
<td>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>Result of the last operation</td>
<td>error</td>
</tr>
<tr>
<td></td>
<td>i . . . . . REXYGEN general error</td>
<td></td>
</tr>
</tbody>
</table>
Example

The diagrams illustrate the execution of MC_MoveRelative blocks for single-axis motion control.

- **Execute Block 1**: Execution and status changes over time, showing the active and done states.
- **Execute Block 2**: Similar execution and status changes.
- **Commanded Velocity**: Graph showing velocity changes over time.
- **Commanded Position**: Graph showing position changes over time.

The parameters for each block are as follows:

- **Block 1**:
  - Velocity: 30, 15
  - Position: 100, 150
  - Mode: 2
  - Deceleration: 20, 10
  - Acceleration: 50, 25

- **Block 2**:
  - Velocity: 15, 30
  - Position: 150
  - Mode: 2
  - Deceleration: 10
  - Acceleration: 25

The graphs depict the execution, active, done, and command states, along with the commanded velocity and position over time.
MC_MoveSuperimposed, MCP_MoveSuperimposed — Superimposed move

Block Symbols

Function Description

The MC_MoveSuperimposed and MCP_MoveSuperimposed blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The MC_MoveSuperimposed block moves an axis to specified position as fast as possible (with respect to set limitations). Final position is specified by input parameter Distance. In case that the axis is already in motion at the moment of execution of the MC_MoveSuperimposed block, the generated values of position, velocity and acceleration are added to the values provided by the previous motion block. If there is no previous motion, the block behaves in the same way as the MC_MoveRelative command.

Note: There is no BufferMode parameter which is irrelevant in the superimposed mode. If there is already an superimposed motion running at the moment of execution, the new block is started immediately (analogous to aborting mode).

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>Distance</td>
<td>Requested target distance (relative to execution point) [unit]</td>
</tr>
<tr>
<td>VelocityDiff</td>
<td>Maximal allowed velocity [unit/s]</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s²]</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s²]</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s³]</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td>i .....</td>
<td>REXYGEN general error</td>
</tr>
</tbody>
</table>
Example

![Diagram of motion control parameters and graphs showing executed commands, active status, and completion status for blocks 1 and 2. The graphs display time in seconds on the x-axis and velocity or position on the y-axis. The diagrams illustrate the commanded velocity and position profiles over time for each block.]

- **Execute**
  - Block 1
  - Block 2

- **Active**
  - Block 1
  - Block 2

- **Done**
  - Block 1
  - Block 2

- **Commanded velocity**
- **Commanded position**
MC_MoveContinuousAbsolute, MCP_MoveContinuousAbsolute – Move to position (absolute coordinate)

Function Description

The MC_MoveContinuousAbsolute and MCP_MoveContinuousAbsolute blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The MC_MoveContinuousAbsolute block moves an axis to specified position as fast as possible. If no further action is pending, final velocity is specified by parameter EndVelocity. For blending purposes, start and stop velocity of this block is maximum velocity with direction respecting current and final position. If start velocity of next pending block is in opposite direction, then blending velocity is always zero.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

Note 1: If the EndVelocity is set to zero value, the block behaves in the same way as MC_MoveAbsolute.

Note 2: If next motion command is executed before the final position is reached, the block behaves in the same way as MC_MoveAbsolute.
Inputs

**uAxis**  Axis reference (only `RM_Axis.axisRef-uAxis` or `yAxis-uAxis` reference connections are allowed)

**Execute**  The block is activated on rising edge  bool

**Position**  Requested target position (absolute) [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)

**Direction**  Direction of movement (cyclic axis or special case only)  long

1  Positive
2  Shortest
3  Negative
4  Current

**EndVelocity**  End velocity  double

Outputs

**yAxis**  Axis reference (only `RM_Axis.axisRef-uAxis` or `yAxis-uAxis` reference connections are allowed)

**InEndVelocity**  Algorithm finished  bool

**CommandAborted**  Algorithm was aborted  bool

**Busy**  Algorithm not finished yet  bool

**Active**  The block is controlling the axis  bool

**Error**  Error occurred  bool

**ErrorID**  Result of the last operation  error

i  REXYGEN general error
Example

```
<table>
<thead>
<tr>
<th>Commanded velocity</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>velocity1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Commanded position</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>position1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>execute1</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
</tr>
<tr>
<td>Execute</td>
</tr>
<tr>
<td>Position</td>
</tr>
<tr>
<td>Blocks</td>
</tr>
<tr>
<td>IfEndVelocity</td>
</tr>
<tr>
<td>KC1</td>
</tr>
<tr>
<td>Velocity</td>
</tr>
<tr>
<td>Acceleration</td>
</tr>
<tr>
<td>Deceleration</td>
</tr>
<tr>
<td>BufferMode</td>
</tr>
<tr>
<td>Direction</td>
</tr>
<tr>
<td>EndVelocity</td>
</tr>
<tr>
<td>Active</td>
</tr>
<tr>
<td>ErrorID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>execute2</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
</tr>
<tr>
<td>Execute</td>
</tr>
<tr>
<td>Position</td>
</tr>
<tr>
<td>Blocks</td>
</tr>
<tr>
<td>IfEndVelocity</td>
</tr>
<tr>
<td>KC1</td>
</tr>
<tr>
<td>Velocity</td>
</tr>
<tr>
<td>Acceleration</td>
</tr>
<tr>
<td>Deceleration</td>
</tr>
<tr>
<td>BufferMode</td>
</tr>
<tr>
<td>Direction</td>
</tr>
<tr>
<td>EndVelocity</td>
</tr>
<tr>
<td>Active</td>
</tr>
<tr>
<td>ErrorID</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Axis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
</tr>
<tr>
<td>Position</td>
</tr>
<tr>
<td>Blocks</td>
</tr>
<tr>
<td>IfEndVelocity</td>
</tr>
<tr>
<td>KC1</td>
</tr>
<tr>
<td>Velocity</td>
</tr>
<tr>
<td>Acceleration</td>
</tr>
<tr>
<td>Deceleration</td>
</tr>
<tr>
<td>BufferMode</td>
</tr>
<tr>
<td>Direction</td>
</tr>
<tr>
<td>EndVelocity</td>
</tr>
<tr>
<td>Active</td>
</tr>
<tr>
<td>ErrorID</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>MC_MoveAbsolute</th>
</tr>
</thead>
<tbody>
<tr>
<td>block 1</td>
</tr>
<tr>
<td>uAxis</td>
</tr>
<tr>
<td>Execute</td>
</tr>
<tr>
<td>Position</td>
</tr>
<tr>
<td>Blocks</td>
</tr>
<tr>
<td>IfEndVelocity</td>
</tr>
<tr>
<td>KC1</td>
</tr>
<tr>
<td>Velocity</td>
</tr>
<tr>
<td>Acceleration</td>
</tr>
<tr>
<td>Deceleration</td>
</tr>
<tr>
<td>BufferMode</td>
</tr>
<tr>
<td>Direction</td>
</tr>
<tr>
<td>EndVelocity</td>
</tr>
<tr>
<td>Active</td>
</tr>
<tr>
<td>ErrorID</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>MC_MoveContinuousAbsolute</th>
</tr>
</thead>
<tbody>
<tr>
<td>block 2</td>
</tr>
<tr>
<td>uAxis</td>
</tr>
<tr>
<td>Execute</td>
</tr>
<tr>
<td>Position</td>
</tr>
<tr>
<td>Blocks</td>
</tr>
<tr>
<td>IfEndVelocity</td>
</tr>
<tr>
<td>KC1</td>
</tr>
<tr>
<td>Velocity</td>
</tr>
<tr>
<td>Acceleration</td>
</tr>
<tr>
<td>Deceleration</td>
</tr>
<tr>
<td>BufferMode</td>
</tr>
<tr>
<td>Direction</td>
</tr>
<tr>
<td>EndVelocity</td>
</tr>
<tr>
<td>Active</td>
</tr>
<tr>
<td>ErrorID</td>
</tr>
</tbody>
</table>
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```

```
| position2             |
| 150                   |
| position1             |
| 100                   |
| mode2                 |
| 2                     |
| direction2            |
| 1                     |
| end_velocity          |
| 10                    |
```
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>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>Distance</td>
<td>Requested target distance (relative to execution point) [unit]</td>
</tr>
<tr>
<td>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>
</tbody>
</table>
**BufferMode** Buffering mode

1. Aborting (start immediately)
2. Buffered (start after finish of previous motion)
3. Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
4. Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
5. Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)
6. Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)

**EndVelocity** End velocity

**Outputs**

- **yAxis** Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis 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** Result of the last operation
  
i. REXYGEN general error
Example

![Diagram of MC_SINGLE - MOTION CONTROL - SINGLE AXIS BLOCKS](image)

**Commanded velocity**

**Commanded position**

Time [s]
**MC_MoveVelocity, MCP_MoveVelocity** — Move with constant velocity

**Block Symbols**

![Block Symbols](image)

**Licence:** MOTION CONTROL

**Function Description**

The MC_MoveVelocity and MCP_MoveVelocity blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP version of the block.

The MC_MoveVelocity block changes axis velocity to specified value as fast as possible and keeps the specified velocity until the command is aborted by another block or event.

Note: parameter Direction enumerate also shortest_way although for this block it is not valid value.

**Inputs**

- **uAxis**: Axis reference (only RM_Axis.axisRef→uAxis or yAxis→uAxis reference connections are allowed)
- **Execute**: The block is activated on rising edge
- **Velocity**: Maximal allowed velocity [unit/s]
- **Acceleration**: Maximal allowed acceleration unit/s²]
- **Deceleration**: Maximal allowed deceleration [unit/s²]
- **Jerk**: Maximal allowed jerk [unit/s³]
- **Direction**: Direction of movement (cyclic axis or special case only)
  - 1 ..... Positive
  - 2 ..... Shortest
  - 3 ..... Negative
  - 4 ..... Current
  - 5 ..... Current
  - 6 ..... Current
  - 7 ..... Current
  - 8 ..... Current
  - 9 ..... Current
  - 10 ..... Current
  - 11 ..... Current
  - 12 ..... Current

- **BufferMode**: Buffer mode
- **ErrorID**: Error ID
- **Busy**: Busy status
- **Active**: Active status
- **CommandAborted**: Command aborted
- **InVelocity**: In velocity
- **yAxis**: y Axis reference
CHAPTER 18. 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)

Outputs

<table>
<thead>
<tr>
<th>Variable</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>reference</td>
</tr>
<tr>
<td>InVelocity</td>
<td>Requested velocity reached</td>
<td>bool</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
<td>bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>error</td>
</tr>
<tr>
<td>i ....</td>
<td>REXYGEN general error</td>
<td></td>
</tr>
</tbody>
</table>
Example
MC_PositionProfile, MCP_PositionProfile – Position profile

Block Symbols

Licence: MOTION CONTROL

Function Description

The **MC_PositionProfile** and **MCP_PositionProfile** blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the **MCP_** version of the block.

The **MC_PositionProfile** block commands a time-position locked motion profile. Block implements two possibilities for definition of time-position function:

1. sequence of values: the user defines a sequence of time-position pairs. In each time interval, the values of position are interpolated. Times sequence is in array **times**, position sequence is in array **values**. Time sequence must be increasing and must start with zero or zero must be between the first and last point. Execution always starts from zero time, so if the sequence start with negative time, part of the profile is not executed (could be used for debugging or time shift). For **MC_VelocityProfile** and **MC_AccelerationProfile** interpolation is linear, but for **MC_PositionProfile**, 3rd order polynomial is used in order to avoid steps in velocity.

2. spline: time sequence is the same as in previous case. Each interval is interpolate by 5th order polynomial \( p(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \) where beginning of the time-interval is for \( x = 0 \), end of time-interval is for \( x = 1 \) and factors \( a_i \) are put in array **values** in ascending order (e.g. array **values** contains 6 values for each interval).

For both case, the time sequence could be equally spaced and then array **times** includes only the first (usually zero) and last point.

Note 1: input **TimePosition** is missing, because all path data are in parameters of the block.

Note 2: parameter **values** must be set as vector in all cases, e.g. text string must not include semicolon.
Note 3: incorrect parameter `cSeg` (higher than real size of arrays `times` and/or `values`) leads to unpredictable result and in some cases crashes whole runtime execution (the problem is platform dependent and currently it is known only for SIMULINK - crash of whole MATLAB).

Note 4: in the spline mode, polynomial is always 5th order and always in position (also for sibling block `MC_VelocityProfile` and `MC_AccelerationProfile`) and it couldn’t be changed. As the special editor exists, this is not important limitation.

Note 5: The block does not include ramp-in mode. If start position and/or velocity of profile is different from actual (commanded) position of axis, block fails with error -707 (step). It is recommended to use `BufferMode=BlendingNext` to eliminate the problem with start velocity.

**Inputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uAxis</code></td>
<td>Axis reference (only <code>RM_Axis.axisRef-uAxis</code> or <code>yAxis-uAxis</code> connections are allowed)</td>
<td></td>
</tr>
<tr>
<td><code>Execute</code></td>
<td>The block is activated on rising edge</td>
<td>bool</td>
</tr>
<tr>
<td><code>TimeScale</code></td>
<td>Overall scale factor in time</td>
<td>double</td>
</tr>
<tr>
<td><code>PositionScale</code></td>
<td>Overall scale factor in value</td>
<td>double</td>
</tr>
<tr>
<td><code>Offset</code></td>
<td>Overall profile offset in value</td>
<td>double</td>
</tr>
<tr>
<td><code>BufferMode</code></td>
<td>Buffering mode</td>
<td>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)

**Outputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>yAxis</code></td>
<td>Axis reference (only <code>RM_Axis.axisRef-yAxis</code> or <code>yAxis-yAxis</code> connections are allowed)</td>
<td></td>
</tr>
<tr>
<td><code>Done</code></td>
<td>Algorithm finished</td>
<td>bool</td>
</tr>
<tr>
<td><code>CommandAborted</code></td>
<td>Algorithm was aborted</td>
<td>bool</td>
</tr>
<tr>
<td><code>Busy</code></td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td><code>Active</code></td>
<td>The block is controlling the axis</td>
<td>bool</td>
</tr>
<tr>
<td><code>Error</code></td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td><code>ErrorID</code></td>
<td>Result of the last operation</td>
<td>error</td>
</tr>
</tbody>
</table>

1. ...  REXYGEN general error
Parameters

alg  Algorithm for interpolation
     1 . . . . Sequence of time/value pairs
     2 . . . . Sequence of equidistant values
     3 . . . . Spline
     4 . . . . Equidistant spline

cSeg  Number of profile segments

times  Times when segments are switched

values  Values or interpolating polynomial coefficients (a0, a1, a2, ...)

<table>
<thead>
<tr>
<th>Values</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0 30]</td>
<td>double</td>
</tr>
<tr>
<td>[0 100 100 50]</td>
<td>double</td>
</tr>
</tbody>
</table>
Example
**MC_Power** – **Axis activation (power on/off)**

**Block Symbol**

![Block Symbol Image]

**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-uAxis** or **yAxis-uAxis** reference connections are allowed)
- **Enable**: Block function is enabled \( \text{bool} \)

**Outputs**

- **yAxis**: Axis reference (only **RM_Axis.axisRef-uAxis** or **yAxis-uAxis** reference connections are allowed)
- **Status**: Effective state of the power stage \( \text{bool} \)
- **Busy**: Algorithm not finished yet \( \text{bool} \)
- **Active**: The block is controlling the axis \( \text{bool} \)
- **Error**: Error occurred \( \text{bool} \)
- **ErrorID**: Result of the last operation \( i \ldots \text{REXYGEN general error} \)
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

- **uAxis**: Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)
- **Enable**: Block function is enabled

Outputs

- **yAxis**: Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)
- **Valid**: Output value is valid
- **Busy**: Algorithm not finished yet
- **Error**: Error occurred
- **ErrorID**: Result of the last operation
  - i ..... REXYGEN general error
- **Position**: Actual absolute position
MC_ReadAxisError – Read axis error

Block Symbol

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**  
  Result of the last operation
- **AxisErrorID**  
  Result of the last operation read from axis

Licence: MOTION CONTROL
MC_ReadBoolParameter — Read axis parameter (bool)

Block Symbol Licence: MOTION CONTROL

Function Description

The block MC_ReadBoolParameter displays actual value of various signals related to the connected axis on its Value output. The user chooses from a set of accessible logical variables by setting the ParameterNumber input. The output value is valid only while the block is activated by the logical Enable input.

The block displays the parameters and outputs of RM_Axis block and is implemented for sake of compatibility with the PLCOpen specification.

Inputs

- **uAxis** Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)
- **Enable** Block function is enabled
- **ParameterNumber** Parameter ID
  - 4 ...... Enable sw positive limit
  - 5 ...... Enable sw negative limit
  - 6 ...... Enable position lag monitoring

Outputs

- **yAxis** Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)
- **Valid** Output value is valid
- **Busy** Algorithm not finished yet
- **Error** Error occurred
- **ErrorID** Result of the last operation
  - i ...... REXYGEN general error
- **Value** Parameter value
**MC_ReadParameter** – Read axis parameter

**Function Description**

The block **MC_ReadParameter** displays actual value of various system variables of the connected axis on its **Value** output. The user chooses from a set of accessible variables by setting the **ParameterNumber** input. The output value is valid only while the block is activated by the logical **Enable** input.

The block displays the parameters and outputs of **RM_Axis** block and is implemented for sake of compatibility with the **PLCOpen** specification.

**Inputs**

- **uAxis**  
  Axis reference (only **RM_Axis.axisRef-uAxis** or **yAxis-uAxis** reference connections are allowed)
- **Enable**  
  Block function is enabled
- **ParameterNumber**  
  Parameter ID
  
<table>
<thead>
<tr>
<th>ParameterNumber</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Commanded position</td>
</tr>
<tr>
<td>2</td>
<td>Positive sw limit switch</td>
</tr>
<tr>
<td>3</td>
<td>Negative sw limit switch</td>
</tr>
<tr>
<td>7</td>
<td>Maximal position 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>10</td>
<td>Actual velocity</td>
</tr>
<tr>
<td>11</td>
<td>Commanded velocity</td>
</tr>
<tr>
<td>12</td>
<td>Maximal acceleration (system)</td>
</tr>
<tr>
<td>13</td>
<td>Maximal acceleration (appl.)</td>
</tr>
<tr>
<td>14</td>
<td>Maximal deceleration (system)</td>
</tr>
<tr>
<td>15</td>
<td>Maximal deceleration (appl.)</td>
</tr>
<tr>
<td>16</td>
<td>Maximal jerk</td>
</tr>
<tr>
<td>1000</td>
<td>Actual position</td>
</tr>
<tr>
<td>1001</td>
<td>Maximal torque/force</td>
</tr>
<tr>
<td>1003</td>
<td>Actual torque/force</td>
</tr>
<tr>
<td>1004</td>
<td>Commanded torque/force</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>Variable</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>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>Result of the last operation</td>
<td>error</td>
</tr>
<tr>
<td>Value</td>
<td>Parameter value</td>
<td>double</td>
</tr>
</tbody>
</table>

*REXYGEN general error*
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 Result of the last operation
   i ..... REXYGEN 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
SynchronizedMotion Axis is in the SynchronizedMotion state
Homing Axis is in the Homing state
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**

![MC_Reset Block Symbol](image)

**Licence:** MOTION CONTROL

**Function Description**

The **MC_Reset** block makes the transition from the state ErrorStop to StandStill by resetting all internal axis-related errors.

**Inputs**

- **uAxis** (Axis reference only `RM_Axis.axisRef-uAxis` or `yAxis-uAxis` 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** (Result of the last operation)

i ..... REXYGEN general error
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 recalculation of movement’s path if a block like MC_MoveAbsolute commands the axis. This recalculation needs lot of CPU time and also numerical problem could appear. For this reasons, a deadband (parameter diff) is established. The movement’s path recalculation is proceeded only if one of the factors is changed more then the deadband.

Note: all factor must be positive. Factor greater then 1.0 are possible, but often lead to overshooting of axis limits and failure of movement (with errorID=-700 - invalid parameter; if factor is set before start of block) or error stop of axis (with errorID=-701 - out of range; if factor is changed within movement and actual value overshoot limit).

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis 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>
<tr>
<td>JerkFactor</td>
<td>Jerk multiplication factor</td>
</tr>
<tr>
<td></td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>double</td>
</tr>
<tr>
<td></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>yAxis</td>
<td>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)</td>
<td>bool</td>
</tr>
<tr>
<td>Enabled</td>
<td>Block function is enabled</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>error</td>
</tr>
</tbody>
</table>

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>diff</td>
<td>Deadband (difference for recalculation)</td>
<td>double</td>
</tr>
</tbody>
</table>
MC_Stop, MCP_Stop – Stopping a movement

Block Symbols

Function Description

The MC_Stop and MCP_Stop blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The MC_Stop block commands a controlled motion stop and transfers the axis to the state Stopping. It aborts any ongoing Function Block execution. While the axis is in state Stopping, no other FB can perform any motion on the same axis. After the axis has reached velocity zero, the Done output is set to true immediately. The axis remains in the state Stopping as long as Execute is still true or velocity zero is not yet reached. As soon as Done=true and Execute=false the axis goes to state StandStill.

Note 1: parameter/input BufferMode is not supported. Mode is always Aborting.

Note 2: Failing stop-command could be dangerous. This block does not generate invalid-parameter-error but tries to stop the axis anyway (e.g. uses parameters from RM_Axis or generates error-stop-sequence).

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxis</td>
<td>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)</td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>bool</td>
</tr>
<tr>
<td>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>yAxis</td>
<td>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis 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>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
</tbody>
</table>
ErrorID  Result of the last operation  error
  i ..... REXYGEN general error
MC_TorqueControl, MCP_TorqueControl – Torque/force control

Block Symbols

Function Description

The MC_TorqueControl and MCP_TorqueControl blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP version of the block.

The MCP_TorqueControl block generates constant slope torque/force ramp until maximum requested value has been reached. Similar profile is generated for velocity. The motion trajectory is limited by maximum velocity, acceleration/deceleration, and jerk, or by the value of the torque, depending on the mechanical circumstances.

Inputs

- **uAxis**: Axis reference (only `RM_Axis.axisRef`-`uAxis` or `yAxis`-`uAxis` connections are allowed)
- **Execute**: The block is activated on rising edge 
- **Torque**: Maximal allowed torque/force
- **TorqueRamp**: Maximal allowed torque/force ramp
- **Velocity**: Maximal allowed velocity [unit/s]
- **Acceleration**: Maximal allowed acceleration [unit/s²]
- **Deceleration**: Maximal allowed deceleration [unit/s²]
- **Jerk**: Maximal allowed jerk [unit/s³]
- **Direction**: Direction of movement (cyclic axis or special case only)
  - 1 . . . Positive
  - 2 . . . Shortest
  - 3 . . . Negative
  - 4 . . . Current

Licence: MOTION CONTROL
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 RMAxis.axisRef-uAxis or yAxis-uAxis reference connections are allowed) Reference
InTorque Requested torque/force is reached bool
CommandAborted Algorithm was aborted bool
Busy Algorithm not finished yet bool
Active The block is controlling the axis bool
Error Error occurred bool
ErrorID Result of the last operation error
  i . . . . REXYGEN general error

Parameter
kma Torque/force to acceleration ratio double
Example

```
velocity 30
torque_ramp 50
torque 100
mode 1
direction 1
deceleration 20
acceleration 50
MC_TorqueControl
 Execute
```
MC_VelocityProfile, MCP_VelocityProfile – Velocity profile

Function Description

The MC_PositionProfile block commands a time-position locked motion profile. Block implements two possibilities for definition of time-velocity function:

1. sequence of values: the user defines a sequence of time-velocity pairs. In each time interval, the values of velocity are interpolated. Times sequence is in array times, position sequence is in array values. Time sequence must be increasing and must start with zero or zero must be between the first and last point. Execution always starts from zero time, so if the sequence start with negative time, part of the profile is not executed (could be used for debugging or time shift). For MC_VelocityProfile and MC_AccelerationProfile interpolation is linear, but for MC_PositionProfile, 3rd order polynomial is used in order to avoid steps in velocity.

2. spline: time sequence is the same as in previous case. Each interval is interpolated by 5th order polynomial \( p(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \) where beginning of the time-interval is for \( x = 0 \), end of time-interval is for \( x = 1 \) and factors \( a_i \) are put in array values in ascending order (e.g. array values contains 6 values for each interval). This method allows smaller number of intervals and there is special editor for synthesis of the interpolating spline function.

For both case, the time sequence could be equally spaced and then array times includes only the first (usually zero) and last point.

Note 1: input TimePosition is missing, because all path data are in parameters of the block.

Note 2: parameter values must be set as vector in all cases, e.g. text string must not include semicolon.

Note 3: incorrect parameter cSeg (higher then real size of arrays times and/or values) leads to unpredictable result and in some cases crashes whole runtime execution (The problem is platform dependent and currently it is known only for SIMULINK - crash of whole MATLAB).
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><code>uAxis</code></th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Execute</code></td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td><code>TimeScale</code></td>
<td>Overall scale factor in time</td>
</tr>
<tr>
<td><code>VelocityScale</code></td>
<td>Overall scale factor in value</td>
</tr>
<tr>
<td><code>Offset</code></td>
<td>Overall profile offset in value</td>
</tr>
<tr>
<td><code>BufferMode</code></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><code>yAxis</code></th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Done</code></td>
<td>Algorithm finished</td>
</tr>
<tr>
<td><code>CommandAborted</code></td>
<td>Algorithm was aborted</td>
</tr>
<tr>
<td><code>Busy</code></td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td><code>Active</code></td>
<td>The block is controlling the axis</td>
</tr>
<tr>
<td><code>Error</code></td>
<td>Error occurred</td>
</tr>
<tr>
<td><code>ErrorID</code></td>
<td>Result of the last operation</td>
</tr>
<tr>
<td>i . . . .</td>
<td>REXYGEN general error</td>
</tr>
</tbody>
</table>
Parameters

\textbf{alg} \quad \text{Algorithm for interpolation} \quad \odot 1 \quad \text{long}
1 \quad \text{Sequence of time/value pairs}
2 \quad \text{Sequence of equidistant values}
3 \quad \text{Spline}
4 \quad \text{Equidistant spline}

\textbf{cSeg} \quad \text{Number of profile segments} \quad \odot 3 \quad \text{long}

\textbf{times} \quad \text{Times when segments are switched} \quad \odot [0 \ 15 \ 25 \ 30] \quad \text{double}

\textbf{values} \quad \text{Values or interpolating polynomial coefficients (a0, a1, a2, ...)} \quad \odot [0 \ 100 \ 100 \ 50] \quad \text{double}
Example
MC_WriteBoolParameter – Write axis parameter (bool)

Function Description

The block MC_WriteBoolParameter writes desired value of various system parameters entered on its Value input to the connected axis. The user chooses from a set of accessible logical variables by setting the ParameterNumber input.

The block is implemented for sake of compatibility with the PLCOpen specification as the parameters can be written by the SETPB block.

Inputs

<table>
<thead>
<tr>
<th>uAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>ParameterNumber</td>
<td>Parameter ID</td>
</tr>
<tr>
<td>Value</td>
<td>Parameter value</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>yAxis</th>
<th>Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td></td>
<td>i . . . . . REXYGEN general error</td>
</tr>
</tbody>
</table>
MC_WriteParameter – Write axis parameter

Block Symbol

Function Description

The block MC_WriteParameter writes desired value of various system parameters entered on its Value input to the connected axis. The user chooses from a set of accessible variables by setting the ParameterNumber input.

The block is implemented for sake of compatibility with the PLCOpen specification as the parameters can be written by the SETPR block.

Inputs

- **uAxis** Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)
- **Execute** The block is activated on rising edge
- **ParameterNumber** Parameter ID
  - 2 .... Positive sw limit switch
  - 3 .... Negative sw limit switch
  - 7 .... Maximal position lag
  - 8 .... Maximal velocity (system)
  - 9 .... Maximal velocity (appl.)
  - 13 .... Maximal acceleration (appl.)
  - 15 .... Maximal deceleration (appl.)
  - 16 .... Maximal jerk
  - 1001 ... Maximal torque/force
- **Value** Parameter value

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** Result of the last operation
  - i .... REXYGEN general error
- double

Licence: MOTION CONTROL
RM_AxisOut – Axis output

Function Description

The RM_AxisOut block allows an access to important states of block RM_Axis. Same outputs are also available directly on RM_Axis (some of them), but this direct output is one step delayed. Blocks are ordered for execution by flow of a signal, so RM_Axis is first then all motion blocks (that actualize RM_Axis state), then RM_AxisOut (should be last) and finally waiting for next period.

Note 1: Control system REXYGEN orders blocks primary by flow of signal, secondarily by name of block (ascendent in alphabetical order), so name like "zzz" is good choice. For checking the order, you can use REXYGEN Diagnostics tool where the blocks are sorted by execution order.

Note 2: almost all blocks do not work with torque so commanded torque is 0. Commanded acceleration and torque should be used as feed-forward value for position/velocity controller so this value does not make any problem.

Inputs

uAxis axis reference that must be connected to axisRef of the RM_Axis block reference (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)

Outputs

pos0 Current commanded position [unit] double
vel0 Current commanded velocity [unit/s] double
acc0 Current commanded acceleration [unit/s²] double
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<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></td>
<td>0 . . . . . . Disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 . . . . . . Stand still</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 . . . . . . Homing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 . . . . . . Discrete motion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 . . . . . . Continuous motion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 . . . . . . Synchronized motion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6 . . . . . . Coordinated motion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7 . . . . . . Stopping</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 . . . . . . Error stop</td>
<td></td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>error</td>
</tr>
<tr>
<td></td>
<td>1 . . . . . . REXYGEN 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

Block Symbol

Licence: MOTION CONTROL

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 possibilites of connection to RM_Axis block: connect all necessary values (outputs of the block RM_AxisOut) as input of interpolating block or use only axis reference and read the state directly. This block uses axis reference. For correct synchronization between two tasks, the block RM_Axis must be executed first followed by all axis related motion control blocks and finally by block RM_AxisOut at the end.

Note 1: For interpolation of position signal, 3rd order polynomial \( p(t) \) is used, where \( p_s(0) = pos0, p_s(t_s) = pos1, \frac{dp_s(t)}{dt} \bigg|_{t=0} = vel0, \frac{dp_s(t)}{dt} \bigg|_{t=t_s} = vel1 \). To interpolate velocity, also an 3rd order polynomial \( p_v(t) \) is used, where \( p_v(0) = vel0, p_v(t_s) = vel1, \frac{dp_v(t)}{dt} \bigg|_{t=0} = acc0, \frac{dp_v(t)}{dt} \bigg|_{t=t_s} = acc1 \). Torque is interpolated by linear function.

Note 2: Because the time of execution of motion blocks is varying in time, the block uses one or two step prediction for interpolation depending on actual conditions and timing of the motion blocks in slower tasks. The use of predicted values is signalized by states RUN0, RUN1, RUN2.

Note 3: Control system REXYGEN orders the blocks primarily by flow of signal, secondarily by name of block (ascendent in alphabetical order), so name like "zzz" is good choice for the block RM_AxisOut. For checking the order, you can use REXYGEN Diagnostics tool where the blocks are sorted by execution order.

Input

uAxis Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis reference connections are allowed)
## Outputs

<table>
<thead>
<tr>
<th>Variable</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>

0 .... Off
1 .... Run0
2 .... Run1
3 .... Run2
5 .... Change1
-1 .... Change0
-2 .... Late
-3 .... Busy
-4 .... Slow
RM_Track – Tracking and inching

Block Symbol

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-&gt;uAxis or yAxis-&gt;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>Moving negative direction mode</td>
</tr>
</tbody>
</table>

** Licence: MOTION CONTROL**
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**: Result of the last operation
  - i ...... REXYGEN general error (error)
Chapter 19

MC_MULTI – Motion control -
multi axis blocks

Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC_CamIn, MCP_CamIn</td>
<td>Engage the cam</td>
<td>542</td>
</tr>
<tr>
<td>MC_CamOut</td>
<td>Disengage the cam</td>
<td>545</td>
</tr>
<tr>
<td>MCP_CamTableSelect</td>
<td>Cam definition</td>
<td>547</td>
</tr>
<tr>
<td>MC_CombineAxes, MCP_CombineAxes</td>
<td>Combine the motion of 2 axes into a third axis</td>
<td>549</td>
</tr>
<tr>
<td>MC_GearIn, MCP_GearIn</td>
<td>Engage the master/slave velocity ratio</td>
<td>552</td>
</tr>
<tr>
<td>MC_GearInPos, MCP_GearInPos</td>
<td>Engage the master/slave velocity ratio in defined position</td>
<td>555</td>
</tr>
<tr>
<td>MC_GearOut</td>
<td>Disengage the master/slave velocity ratio</td>
<td>560</td>
</tr>
<tr>
<td>MC_PhasingAbsolute, MCP_PhasingAbsolute</td>
<td>Phase shift in synchronized motion (absolute coordinates)</td>
<td>562</td>
</tr>
<tr>
<td>MC_PhasingRelative, MCP_PhasingRelative</td>
<td>Phase shift in synchronized motion (relative coordinates)</td>
<td>565</td>
</tr>
</tbody>
</table>

This block set is the second part of motion control blocks library according to the PLCopen standard for multi axis control. General vendor specific rules are the same as described in chapter 18 (the MC_SINGLE library, blocks for single axis motion control).
MC_CamIn, MCP_CamIn – Engage the cam

Function Description

The MC_CamIn and MCP_CamIn blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.

The MC_CamIn block switches on a mode in which the slave axis is commanded to position which corresponds to the position of master axis transformed with a function defined by the MCP_CamTableSelect block (connected to CamTableID input). Denoting the transformation as \( \text{Cam}(x) \), master axis position \( \text{Pos}_M \) and slave axis position \( \text{Pos}_S \), we obtain (for absolute relationship, without phasing): \( \text{Pos}_S = \text{Cam}((\text{Pos}_M - \text{MasterOffset})/\text{MasterScaling}) \cdot \text{SlaveScaling} + \text{SlaveOffset} \). This form of synchronized motion of the slave axis is called electronic cam.

The cam mode is switched off by executing other motion block on slave axis with mode aborting or by executing a MC_CamOut block. The cam mode is also finished when the master axis leaves a non-periodic cam profile. This situation is indicated by the EndOfProfile output.

In case of a difference between real position and/or velocity of slave axis and cam-profile slave axis position and velocity, some transient trajectory must be generated to cancel this offset. This mode is called ramp-in. The ramp-in function is added to the cam profile to eliminate the difference in start position. The RampIn parametr is an average velocity of the ramp-in function. Ramp-in path is not generated for \( \text{RampIn}=0 \) and error \(-707\) (position or velocity step) is invoked if some difference is detected. Recommended value for the RampIn parametr is 0.1 to 0.5 of maximal slave axis velocity. The parameter has to be lowered if maximal velocity or acceleration error is detected.
Inputs

- **uMaster** Master axis reference
- **uSlave** Slave axis reference
- **CamTableID** Cam table reference (connect to MCP_CamTableSelect.CamTableID)
- **Execute** The block is activated on rising edge
- **MasterOffset** Offset in cam table on master side [unit]
- **SlaveOffset** Offset in cam table on slave side [unit]
- **MasterScaling** Overall scaling factor in cam table on master side
- **SlaveScaling** Overall scaling factor in cam table on slave side
- **StartMode** Select relative or absolute cam table
  - 1 .... Master relative
  - 2 .... Slave relative
  - 3 .... Both relative
  - 4 .... Both absolute
- **BufferMode** Buffering mode
  - 1 .... Aborting (start immediately)
  - 2 .... Buffered (start after finish of previous motion)
  - 3 .... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
  - 4 .... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
  - 5 .... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)
  - 6 .... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)
- **RampIn** RampIn factor (0 = RampIn mode not used); average additive velocity (absolute value) during ramp-in process

Outputs

- **yMaster** Master axis reference
- **ySlave** Slave axis reference
- **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** Result of the last operation
  - i .... REXYGEN general error
- **EndOfProfile** Indicate end of cam profile (not periodic cam only)
- **SyncDistance** Position deviation of the slave axis from synchronized position
Example

Block 0 execute (CamTableSelect)
Block 1 execute (CamIn)
Block 2 execute (MoveVelocity)

Block 1 active (CamIn)
Block 2 active (MoveVelocity)

Acceleration

Velocity

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**  
  Result of the last operation
    - i . . . . .  
    - REXYGEN general error
Example

![Diagram showing MC Multi - Motion Control - Multi Axis Blocks example with various blocks and parameters.]

**Block 0 execute (CamTableSelect)**
- **Execute** block 0
- **Velocity** block 0
- **Acceleration** block 0
- **Position** block 0

**Block 1 execute (CamIn)**
- **Active** block 1
- **Acceleration** block 1
- **Velocity** block 1
- **Position** block 1

**Block 2 execute (MoveVelocity)**
- **Active** block 2
- **Acceleration** block 2
- **Velocity** block 2
- **Position** block 2

**Block 3 execute (CamOut)**
- **Active** block 3
- **Acceleration** block 3
- **Velocity** block 3
- **Position** block 3

- **Execute**
- **Active**
- **Acceleration**
- **Velocity**
- **Position**

**Parameters:**
- **Velocity:** block 0
- **Mode:** block 0
- **Jerk:** block 0
- **Execute:** block 0
- **Deceleration:** block 0
- **Axis Master:** block 0
- **Axis Slave:** block 0
- **Acceleration:** block 0
- **SS:** block 0
- **SO:** block 0
- **RF:** block 0
- **MS:** block 0
- **MO:** block 0
- **MC Stop Block 4**
- **MC Move Velocity Block 2**
- **MC Cam Out Block 3**
- **MC Cam In Block 1**
- **MC Cam Table Select Block 0**

**Graphs:**
- Velocity vs. Time
- Acceleration vs. Time
- Velocity vs. Time
- Position vs. Time
MCP_CamTableSelect – Cam definition

Block Symbol

License: MOTION CONTROL

Function Description

The MCP_CamTableSelect block defines a cam profile. The definition is similar to MC_PositionProfile block, but the time axis is replaced by master position axis. There are also two possible ways for cam profile definition:

1. sequence of values: given sequence of master-slave position pairs. In each master position interval, value of slave position is interpolated by 3rd-order polynomial (simple linear interpolation would lead to steps in velocity at interval border). Master position sequence is in array/parameter mvalues, slave position sequence is in array/parameter svalues. Master position sequence must be increasing.

2. spline: master position sequence is the same as in previous case. Each interval is interpolated by 5th-order polynomial \( p(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \) where beginning of time-interval is defined for \( x = 0 \), end of time-interval holds for \( x = 1 \) and factors \( a_i \) are put in array/parameter svalues in ascending order (e.g. array/parameter svalues contain 6 values for each interval). This method allows to reduce the number of intervals and there is special graphical editor available for interpolating curve synthesis.

For both cases the master position sequence can be equidistantly spaced in time and then the time array includes only first and last point.

Note 1: input CamTable which is defined in PLCOpen specification is missing, because all path data are set in the parameters of the block.

Note 2: parameter svalues must be set as a vector in all cases, e.g. text string must not include a semicolon.

Note 3: incorrect parameter value cSeg (higher then real size of arrays times and/or values) can lead to unpredictable results and in some cases to crash of the whole runtime execution (The problem is platform dependent and currently it is observed only for SIMULINK version).

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uMaster</td>
<td>Master axis reference</td>
<td>reference</td>
</tr>
<tr>
<td>uSlave</td>
<td>Slave axis reference</td>
<td>reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</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>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>Result of the last operation</td>
<td>error</td>
</tr>
<tr>
<td>CamTableID</td>
<td>Cam table reference (connect to MC_CamIn.CamTableID)</td>
<td>reference</td>
</tr>
</tbody>
</table>

### 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>⊕2 long</td>
</tr>
<tr>
<td>cSeg</td>
<td>Number of profile segments</td>
<td>⊕3 long</td>
</tr>
<tr>
<td>Periodic</td>
<td>Indicate periodic cam profile</td>
<td>⊕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>⊕[0 30] double</td>
</tr>
<tr>
<td>svalues</td>
<td>Slave positions or interpolating polynomial coefficients (a0, a1, a2, ...)</td>
<td>⊕[0 100 100 0] double</td>
</tr>
</tbody>
</table>
MC_CombineAxes, MCP_CombineAxes – Combine the motion of 2 axes into a third axis

Block Symbols

Function Description

The MC_CombineAxes block combines a motion of two master axes into a slave axis command. The slave axis indicates synchronized motion state. Following relationship holds:

\[
\text{SlavePosition} = \frac{\text{GearRatioNumeratorM1}}{\text{GearRatioDenominatorM1}} \cdot \text{Master1Position} + \frac{\text{GearRatioNumeratorM2}}{\text{GearRatioDenominatorM2}} \cdot \text{Master2Position}
\]

Negative number can be set in \text{GearRatio}... parameter to obtain the resulting slave movement in form of difference of master axes positions.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uMaster1</td>
<td>First master axis reference</td>
</tr>
<tr>
<td>uMaster2</td>
<td>Second 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>GearRatioNumeratorM1</td>
<td>Numerator for the gear factor for master axis 1</td>
</tr>
<tr>
<td>GearRatioDenominatorM1</td>
<td>Denominator for the gear factor for master axis 1</td>
</tr>
<tr>
<td>GearRatioNumeratorM2</td>
<td>Numerator for the gear factor for master axis 2</td>
</tr>
<tr>
<td>GearRatioDenominatorM2</td>
<td>Denominator for the gear factor for master axis 2</td>
</tr>
</tbody>
</table>
BufferMode Buffering mode

1 ..... Aborting (start immediately)
2 ..... Buffered (start after finish of previous motion)
3 ..... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
4 ..... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
5 ..... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity)
6 ..... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)

RampIn RampIn factor (0 = RampIn mode not used)

Outputs

yMaster1 First master axis reference reference
yMaster2 Second master axis reference reference
ySlave Slave axis reference reference
InSync Slave axis reached the cam profile bool
CommandAborted Algorithm was aborted bool
Busy Algorithm not finished yet bool
Active The block is controlling the axis bool
Error Error occurred bool
ErrorID Result of the last operation error

SyncDistance Position deviation of the slave axis from synchronized position double
MC_GearIn, MCP_GearIn — Engage the master/slave velocity ratio

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 phas- ing): $V_{elS} = V_{elM} \times \frac{\text{RatioNumerator}}{\text{RatioDenominator}}$. Position and acceleration is commanded to be consistent with velocity; position/distance ratio is also locked. This mode of synchronized motion is called electronic gear.

The gear mode is switched off by executing other motion block on slave axis with mode aborting or by executing a MC_GearIn block.

Similarly to the MC_CamIn block, ramp-in mode is activated if initial velocity of slave axis is different from master axis and gearing ratio. Parameters Acceleration, Deceleration, Jerk are used during ramp-in mode.

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uMaster</td>
<td>Master axis reference</td>
<td>reference</td>
</tr>
<tr>
<td>uSlave</td>
<td>Slave axis reference</td>
<td>reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>bool</td>
</tr>
<tr>
<td>RatioNumerator</td>
<td>Gear ratio Numerator</td>
<td>long</td>
</tr>
<tr>
<td>RatioDenominator</td>
<td>Gear ratio Denominator</td>
<td>long</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Maximal allowed acceleration [unit/s$^2$]</td>
<td>double</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration [unit/s$^2$]</td>
<td>double</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk [unit/s^3]</td>
<td>double</td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>1 . . . .  Abort (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 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 commands)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 . . . .  Blending previous (start after finishing the previous motion,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>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 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>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>InGear</td>
<td>Slave axis reached gearing ratio</td>
<td>bool</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
<td>bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>error</td>
</tr>
<tr>
<td></td>
<td>1 . . . .  REXYGEN general error</td>
<td></td>
</tr>
</tbody>
</table>
Example

```
velocity2
30
ratio_numerator
2
ratio_denominator
1
position2
100
mode2
2
mode1
2
jerk1
0
execute2
execute1
direction2
2
deceleration2
20
deceleration1
10
acceleration2
50
acceleration1
25
```

```
MC_MoveAbsolute
block 2
uAxis
Execute
Position
Velocity
Acceleration
Deceleration
Jerk
BufferMode
yAxis
Done
CommandAborted
Busy
Active
Error
ErrorID
```

```
MC_GearIn
block 1
```

```
MC_GearIn = Block 1
MC_MoveAbsolute = Block 2
```

```
Execute
bool
Block 1 execute (GearIn)
Block 2 execute (MoveAbsolute)
0 1 2 3 4 5 ...
```

```
Active
bool
Block 1 active (GearIn)
Block 2 active (MoveAbsolute)
0 1 2 3 4 5 ...
```

```
Acceleration
acceleration
```

```
Velocity
velocity
```

```
Position
position
```
**MC_GearInPos, MCP_GearInPos – Engage the master/slave velocity ratio in defined position**

**Function Description**

*The MC_GearInPos and MCP_GearInPos blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_ version of the block.*

The functional block MC_GearInPos engages a synchronized motion of master and slave axes in such a way that the ratio of velocities of both axes is maintained at a constant value. Compared to MC_GearIn, also the master to slave position ratio is determined in a given reference point, i.e. following relation holds:

$$\frac{\text{SlavePosition} - \text{SlaveSyncPosition}}{\text{MasterPosition} - \text{MasterSyncPosition}} = \frac{\text{RatioNumerator}}{\text{RatioDenominator}}.$$ 

In case that the slave position does not fulfill this condition of synchronicity at the moment of block activation (i.e. in an instant of positive edge of Execute input and after execution of previous commands in buffered mode), synchronization procedure is started and indicated by output StartSync. During this procedure, proper slave trajectory which results in smooth synchronization of both axes is generated with respect to actual master motion and slave limits for Velocity, Acceleration, Deceleration and Jerk (these limits are not applied from the moment of successful synchronization). Parameter setting 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 $\text{MC\_MoveAbsolute}$ is recomputed in every time instant in such a way, that the end
velocity is set to actual velocity of master axis. II. The position, velocity and acceleration
is generated in the same manner as in the synchronized motion and a proper 5th order
interpolation polynomial is added to achieve smooth transition to the synchronized state.
The length of interpolation trajectory is computed in such a way that maximum velocity,
acceleration and jerk do not violate the specified limits (for the interpolation polynomial).
The first algorithm cannot be used for nonzero acceleration of the master axis whereas
the second does not guarantee the compliance of maximum limits for the overall slave
trajectory. Both algorithms are combined in a proper way to achieve the synchronized
motion of both axes.

2. The block parameters (execution of synchronization and velocity/acceleration limits)
have to be chosen so that the slave position is close to $\text{SlaveSyncPosition}$ approx-
imately 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

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{uMaster}$</td>
<td>Master axis reference</td>
</tr>
<tr>
<td>$\text{uSlave}$</td>
<td>Slave axis reference</td>
</tr>
<tr>
<td>$\text{Execute}$</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>$\text{RatioNumerator}$</td>
<td>Gear ratio Numerator</td>
</tr>
<tr>
<td>$\text{RatioDenominator}$</td>
<td>Gear ratio Denominator</td>
</tr>
<tr>
<td>$\text{MasterSyncPosition}$</td>
<td>Master position for synchronization</td>
</tr>
<tr>
<td>$\text{SlaveSyncPosition}$</td>
<td>Slave position for synchronization</td>
</tr>
<tr>
<td>$\text{MasterStartDistance}$</td>
<td>Master distance for starting gear in procedure</td>
</tr>
<tr>
<td>$\text{Velocity}$</td>
<td>Maximal allowed velocity $[\text{unit/s}]$</td>
</tr>
<tr>
<td>$\text{Acceleration}$</td>
<td>Maximal allowed acceleration $[\text{unit/s}^2]$</td>
</tr>
<tr>
<td>$\text{Deceleration}$</td>
<td>Maximal allowed deceleration $[\text{unit/s}^2]$</td>
</tr>
<tr>
<td>$\text{Jerk}$</td>
<td>Maximal allowed jerk $[\text{unit/s}^3]$</td>
</tr>
</tbody>
</table>
**BufferMode** Buffering mode

1. Aborting (start immediately)
2. Buffering (start after finish of previous motion)
3. Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands)
4. Blending high (start after finishing the previous motion, previous motion finishes with the highest 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**: Result of the last operation
  
  i. REXYGEN general error

- **SyncDistance**: Position deviation of the slave axis from synchronized position

**SyncMode** is a long data type.

**Outputs** include:
- **yMaster**: Reference type
- **ySlave**: Reference type
- **StartSync**, **InSync**: Boolean type
- **CommandAborted**: Boolean type
- **Busy**, **Active**, **Error**: Boolean type
- **ErrorID**: Error type
- **SyncDistance**: Double type
Example
Execute

bool
Block 1 execute (MoveAbsolute1)
Block 2 execute (GearIn)
Block 3 execute (MoveAbsolute2)

Active

bool
Block 1 active (MoveAbsolute1)
Block 2 active (GearIn)
Block 3 active (MoveAbsolute2)

InSync

bool
Block 2 InSync (GearIn)

Acceleration

acceleration
Acceleration axis 1
Acceleration axis 2

Velocity

velocity
Velocity axis 1
Velocity axis 2

Position

position
Position axis 1
Position axis 2
**MC_GearOut** – Disengage the master/slave velocity ratio

**Block Symbol**

![Block Symbol](image)

**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: Result of the last operation
- i: REXYGEN general error
Example

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>velocity2</td>
<td>30</td>
</tr>
<tr>
<td>ratio_numerator1</td>
<td>2</td>
</tr>
<tr>
<td>ratio_denominator1</td>
<td>1</td>
</tr>
<tr>
<td>position2</td>
<td>100</td>
</tr>
<tr>
<td>mode2</td>
<td>2</td>
</tr>
<tr>
<td>mode1</td>
<td>2</td>
</tr>
<tr>
<td>jerk3</td>
<td>0</td>
</tr>
<tr>
<td>jerk1</td>
<td>0</td>
</tr>
</tbody>
</table>

Execute

<table>
<thead>
<tr>
<th>Block</th>
<th>Execute</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Block 1 execute (GearIn)</td>
</tr>
<tr>
<td>2</td>
<td>Block 2 execute (MoveAbsolute)</td>
</tr>
<tr>
<td>3</td>
<td>Block 3 execute (GearOut)</td>
</tr>
</tbody>
</table>

Active

<table>
<thead>
<tr>
<th>Block</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Block 1 active (GearIn)</td>
</tr>
<tr>
<td>2</td>
<td>Block 2 active (MoveAbsolute)</td>
</tr>
</tbody>
</table>

Acceleration

<table>
<thead>
<tr>
<th>Axis</th>
<th>Acceleration</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td>50</td>
</tr>
<tr>
<td>slave</td>
<td>10</td>
</tr>
</tbody>
</table>

Velocity

<table>
<thead>
<tr>
<th>Axis</th>
<th>Velocity</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td>150</td>
</tr>
<tr>
<td>slave</td>
<td>200</td>
</tr>
</tbody>
</table>

Position

<table>
<thead>
<tr>
<th>Axis</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td>100</td>
</tr>
<tr>
<td>slave</td>
<td>150</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uMaster</td>
<td>Master axis reference</td>
<td>reference</td>
</tr>
<tr>
<td>uSlave</td>
<td>Slave axis reference</td>
<td>reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
<td>bool</td>
</tr>
<tr>
<td>PhaseShift</td>
<td>Requested phase shift (distance on master axis) for cam</td>
<td>double</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><strong>Jerk</strong></td>
<td>Maximal allowed jerk [unit/s^3]</td>
<td><strong>double</strong></td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td><strong>BufferMode</strong></td>
<td>Buffering mode</td>
<td><strong>long</strong></td>
</tr>
<tr>
<td>1 . . . .</td>
<td>Aborting</td>
<td></td>
</tr>
<tr>
<td>2 . . . .</td>
<td>Buffered</td>
<td></td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th><strong>yMaster</strong></th>
<th>Master axis reference</th>
<th><strong>reference</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ySlave</strong></td>
<td>Slave axis reference</td>
<td><strong>reference</strong></td>
</tr>
<tr>
<td><strong>Done</strong></td>
<td>Algorithm finished</td>
<td><strong>bool</strong></td>
</tr>
<tr>
<td><strong>CommandAborted</strong></td>
<td>Algorithm was aborted</td>
<td><strong>bool</strong></td>
</tr>
<tr>
<td><strong>Busy</strong></td>
<td>Algorithm not finished yet</td>
<td><strong>bool</strong></td>
</tr>
<tr>
<td><strong>Active</strong></td>
<td>The block is controlling the axis</td>
<td><strong>bool</strong></td>
</tr>
<tr>
<td><strong>Error</strong></td>
<td>Error occurred</td>
<td><strong>bool</strong></td>
</tr>
<tr>
<td><strong>ErrorID</strong></td>
<td>Result of the last operation</td>
<td><strong>error</strong></td>
</tr>
<tr>
<td></td>
<td>i . . . . REXYGEN general error</td>
<td></td>
</tr>
</tbody>
</table>
Example

- Velocity:
  - Velocity axis 1 master: 0, 50, 100, 150, 200
  - Velocity axis 2 slave: 0, 50, 100, 150, 200

- Position:
  - Position axis slave without phasing: 0, 150, 300, 450, 600
  - Position axis slave with phasing: 0, 150, 300, 450, 600

- Acceleration:
  - Acceleration axis 1 master: 0, 50, 100, 150, 200
  - Acceleration axis 2 slave: 0, 50, 100, 150, 200

- Active:
  - Block 1 active (GearIn): 0, 1, 2, 3, 4
  - Block 2 active (MoveAbsolute): 0, 1, 2, 3, 4
  - Block 3 active (PhasingAbsolute): 0, 1, 2, 3, 4

- Execute:
  - Block 1 execute (GearIn): 0, 1, 2, 3, 4
  - Block 2 execute (MoveAbsolute): 0, 1, 2, 3, 4
  - Block 3 execute (PhasingAbsolute): 0, 1, 2, 3, 4
MC_PhasingRelative, MCP_PhasingRelative – Phase shift in synchronized motion (relative coordinates)

Block Symbols

Function Description

The MC_PhasingRelative and MCP_PhasingRelative blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP_version of the block.

The MC_PhasingRelative introduces an additional phase shift in master-slave relation defined by an electronic cam (MC_CamIn) or electronic gear (MC_GearIn). The functionality of this command is very similar to MC_MoveSuperimposed (additive motion from 0 to PhaseShift position with respect to maximum velocity acceleration and jerk). The only difference is that the additive position/velocity/acceleration is added to master axis reference position in the functional dependence defined by a cam or gear ratio for the computation of slave motion instead of its direct summation with master axis movement. The relative value of final phase shift with respect to previous value is specified by PhaseShift parameter. Note: The motion command is analogous to rotation of a mechanical cam by angle PhaseShift.

Inputs

- 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
- Velocity: Maximal allowed velocity [unit/s]
- Acceleration: Maximal allowed acceleration [unit/s^2]
- Deceleration: Maximal allowed deceleration [unit/s^2]
Jerk         Maximal allowed jerk [unit/s³]                 double
BufferMode  Buffering mode                               long
            1      Aborting
            2      Buffered

Outputs

yMaster     Master axis reference                         reference
ySlave      Slave axis reference                          reference
Done        Algorithm finished                            bool
CommandAborted Algorithm was aborted                     bool
Busy        Algorithm not finished yet                     bool
Active      The block is controlling the axis             bool
Error       Error occurred                                bool
ErrorID     Result of the last operation                   error
            i      REXYGEN general error
Chapter 20

MC_COORD – Motion control - coordinated movement blocks

Contents

RM_AxesGroup – Axes group for coordinated motion control  . . . 570
RM_Feed – MC Feeder ................................................. 573
RM_Gcode – CNC motion control ............................. 574
MC_AddAxisToGroup – Adds one axis to a group .............. 576
MC_UngroupAllAxes – Removes all axes from the group .... 577
MC_GroupEnable – Changes the state of a group to GroupEnable 578
MC_GroupDisable – Changes the state of a group to GroupDisabled 579
MC_SetCartesianTransform – Sets Cartesian transformation . . 580
MC_ReadCartesianTransform – Reads the parameter of the cartesian transformation ........................................... 582
MC_GroupSetPosition, MCP_GroupSetPosition – Sets the position of all axes in a group ........................................... 583
MC_GroupReadActualPosition – Read actual position in the selected coordinate system .................................... 585
MC_GroupReadActualVelocity – Read actual velocity in the selected coordinate system ........................................ 586
MC_GroupReadActualAcceleration – Read actual acceleration in the selected coordinate system ......................... 587
MC_GroupStop – Stopping a group movement ................. 588
MC_GroupHalt – Stopping a group movement (interruptible) . 591
MC_GroupInterrupt, MCP_GroupInterrupt – Read a group interrupt ................................................................. 596
MC_GroupContinue – Continuation of interrupted movement . . 597
MC_GroupReadStatus – Read a group status .................. 598
MC_GroupReadError – Read a group error ....................... 599
MC_GroupReset – Reset axes errors ............................. 600
MC_MoveLinearAbsolute – Linear move to position (absolute coordinates) ........................................... 601
MC_MoveLinearRelative – Linear move to position (relative to execution point) .......................... 605
MC_MoveCircularAbsolute – Circular move to position (absolute coordinates) .......................... 609
MC_MoveCircularRelative – Circular move to position (relative to execution point) .................. 613
MC_MoveDirectAbsolute – Direct move to position (absolute coordinates) .................................. 617
MC_MoveDirectRelative – Direct move to position (relative to execution point) ....................... 620
MC_MovePath – General spatial trajectory generation ................................................................. 623
MC_GroupSetOverride – Set group override factors .................................................................. 625
<table>
<thead>
<tr>
<th>Aborting</th>
<th>Buffered without Blending</th>
<th>Blending</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Trajectory of TCP" /></td>
<td><img src="image2" alt="Trajectory of TCP" /></td>
<td><img src="image3" alt="Trajectory of TCP" /></td>
</tr>
<tr>
<td><img src="image4" alt="Speed of TCP" /></td>
<td><img src="image5" alt="Speed of TCP" /></td>
<td><img src="image6" alt="Speed of TCP" /></td>
</tr>
</tbody>
</table>
RM_AxesGroup – Axes group for coordinated motion control

Block Symbol Licence: COORDINATED MOTION

Function Description

Note 1: Applicable for all non-administrative (moving) function blocks.
Note 2: In the states GroupErrorStop or GroupStopping, all Function Blocks can be called, although they will not be executed, except MC_GroupReset for GroupErrorStop and any occurring Error – they will generate the transition to GroupStandby or GroupErrorStop respectively.
Note 3: MC_GroupStop.DONE AND NOT MC_GroupStop.EXECUTE
Note 4: Transition is applicable if last axis is removed from the group.
Note 5: Transition is applicable while group is not empty.
Note 6: MC_GroupDisable and MC_UngroupAllAxes can be issued in all states and will change the state to GroupDisabled.

Parameters

<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>long</td>
</tr>
<tr>
<td>AcsCount</td>
<td>Number of axis in ACS</td>
<td>long</td>
</tr>
<tr>
<td>PosCount</td>
<td>Number of position axis</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>
</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>
<tr>
<td></td>
<td>0 .... Disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 .... Standby</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 .... Homing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6 .... Moving</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7 .... Stopping</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 .... Error stop</td>
<td></td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>error</td>
</tr>
<tr>
<td></td>
<td>i .... REXYGEN general error</td>
<td></td>
</tr>
</tbody>
</table>
The State Diagram of AxesGroup

- **GroupMoving**
- **GroupHoming**
- **GroupDisabled**
- **GroupStandby**
- **GroupStopping**
- **GroupErrorStop**

**MC_GroupDisable**

**MC_UngroupAllAxes**

**MC_RemoveAxisFromGroup**

**MC_AddAxisToGroup**

**Note 1 and MC_GroupHalt**

**Note 2**

**Error**

**Note 3**

**Error**

**Note 4**

**Note 5**

**MC_GroupEnable**

**MC_GroupHome**

**MC_GroupStop**

**MC_GroupReset**

**MC_AssAxisToGroup**

**MC_RemoveAxisFromGroup**
adding particular axis to axesgroup
implementation of particular single axis
implementation of axes group

vel4 [actual_velocity4]
vel3 [actual_velocity3]
vel2 [actual_velocity2]
vel1 [actual_velocity1]

vel4 [commanded_velocity4]
vel3 [commanded_velocity3]
vel2 [commanded_velocity2]
vel1 [commanded_velocity1]

commanded_tor4 [commanded_torque4]
commanded_tor3 [commanded_torque3]
commanded_tor2 [commanded_torque2]
commanded_tor1 [commanded_torque1]

commanded_pos4 [commanded_position4]
commanded_pos3 [commanded_position3]
commanded_pos2 [commanded_position2]
commanded_pos1 [commanded_position1]

RM_Axis4
HLD
ActualPos
ActualVelocity
ActualTorque
LIMN
LIMZ
LIMP
axisRef
PhysicalPosition
CommandedVelocity
CommandedAcceleration
CommandedTorque
iState
ErrorID
CommandedPosition

RM_Axis3
HLD
ActualPos
ActualVelocity
ActualTorque
LIMN
LIMZ
LIMP
axisRef
PhysicalPosition
CommandedVelocity
CommandedAcceleration
CommandedTorque
iState
ErrorID
CommandedPosition

RM_Axis2
HLD
ActualPos
ActualVelocity
ActualTorque
LIMN
LIMZ
LIMP
axisRef
PhysicalPosition
CommandedVelocity
CommandedAcceleration
CommandedTorque
iState
ErrorID
CommandedPosition

RM_Axis1
HLD
ActualPos
ActualVelocity
ActualTorque
LIMN
LIMZ
LIMP
axisRef
PhysicalPosition
CommandedVelocity
CommandedAcceleration
CommandedTorque
iState
ErrorID
CommandedPosition

MC_GroupEnable_1234
uAxesGroup
Execute
yAxesGroup
Done
Busy
Error
ErrorID

MC_AddAxisToGroup_O4
uAxesGroup
Execute
yAxesGroup
Done
Busy
Error
ErrorID

MC_AddAxisToGroup_O3
uAxesGroup
Execute
yAxesGroup
Done
Busy
Error
ErrorID

MC_AddAxisToGroup_O2
uAxesGroup
Execute
yAxesGroup
Done
Busy
Error
ErrorID

MC_AddAxisToGroup_O1
uAxesGroup
Execute
yAxesGroup
Done
Busy
Error
ErrorID

MCP_SetKinTransform_Agebot
uAxesGroup
Execute
yAxesGroup
Done
Busy
Error
ErrorID

CNB3
on
CNB2
on
CNB1
on
CNB1
on
CNB
1
RM_Feed – * MC Feeder

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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Axes group reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>String</td>
</tr>
<tr>
<td>VelFactor</td>
<td>(0.01 \leq 100.0 \leq 1.0) double</td>
</tr>
<tr>
<td>Relative</td>
<td>bool</td>
</tr>
<tr>
<td>CoordSystem</td>
<td>(1 \leq 3 \leq 2) long</td>
</tr>
<tr>
<td>BufferMode</td>
<td>(1 \leq 6 \leq 1) long</td>
</tr>
<tr>
<td>TransitionMode</td>
<td>(0 \leq 15 \leq 1) long</td>
</tr>
<tr>
<td>TransitionParameter</td>
<td>double</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxesGroup</td>
<td>Axes group reference</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td>Aux</td>
<td>double</td>
</tr>
</tbody>
</table>
CHAPTER 20. MC_COORD – MOTION CONTROL - COORDINATED MOVEMENT BLOCKS

RM_Gcode – * CNC motion control

Block Symbol

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

Inputs

| uAxesGroup | Axes group reference |
| Exec | The block is activated on rising edge |
| BlockSkip | MILAN |

Parameters

| BaseDir | Directory of the G-code files |
| MainFile | Source file number |
| CoordSystem | Buffering mode |
| BufferMode | Transition mode in blending mode |
| TransitionParameter | Parametr for transition (depends on transition mode) |
| workOffsets | Sets with initial coordinate |

| CoordSystem | long |
| BufferMode | 1 Aborting, 2 Buffered, 3 Blending low, 4 Blending high, 5 Blending previous, 6 Blending next |
| TransitionMode | 1 TMNone, 2 TMStartVelocity, 3 TMConstantVelocity, 4 TMCornrDistance, 5 TMMaxCornerDeviation, 11 Smooth |
| TransitionParameter | double |
| workOffsets | [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0] |
toolOffsets  Sets of tool offset  \[0 0 0\] double
cutterOffsets  Tool radii  \[0 0 0\] double

**Outputs**

- **yAxesGroup**  Axes group reference  reference
- **Done**  Algorithm finished  bool
- **CommandAborted**  Algorithm was aborted  bool
- **Busy**  Algorithm not finished yet  bool
- **Active**  The block is controlling the axis  bool
- **Error**  Error occurred  bool
- **ErrorID**  Result of the last operation  error
- **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 Licence: COORDINATED MOTION

Function Description

The function block MC_AddAxisToGroup adds one uAxis to the group in a structure uAxesGroup. Axes Group is implemented by the function block RM_AxesGroup. The input uAxis must be defined by the function block RM_Axis from the MC_SINGLE library.

Note 1: Every IdentInGroup is unique and can be used only for one time otherwise the error is set.

Inputs

- **uAxesGroup** Axes group reference reference
- **uAxis** Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed) reference
- **Execute** The block is activated on rising edge bool
- **IdentInGroup** The order of axes in the group (0 = first unassigned) long

Outputs

- **yAxesGroup** Axes group reference reference
- **yAxis** Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed) reference
- **Done** Algorithm finished bool
- **Busy** Algorithm not finished yet bool
- **Error** Error occurred bool
- **ErrorID** Result of the last operation error
  
  i ..... REXYGEN general error
MC_UngroupAllAxes – Removes all axes from the group

Block Symbol

Licence: COORDINATED MOTION

Function Description

The function block MC_UngroupAllAxes removes all axes from the group uAxesGroup. After finalization the state is changed to "GroupDisabled".

Note 1: If the function block is execute in the group state "GroupDisabled", "GroupStandBy" or "GroupErrorStop" the error is set and the block is not execute.

Inputs

| uAxesGroup | Axes group reference |
| Execute | The block is activated on rising edge |

Outputs

| yAxesGroup | Axes group reference |
| Done | Algorithm finished |
| Busy | Algorithm not finished yet |
| Error | Error occurred |
| ErrorID | Result of the last operation |

  
  i . . . . . REXYGEN general error
**MC_GroupEnable** – Changes the state of a group to GroupEnable

**Block Symbol**

**Licence:** COORDINATED MOTION

**Function Description**

The function block `MC_GroupEnable` changes the state for the group `uAxesGroup` from "GroupDisabled" to "GroupStandby".

**Inputs**

- `uAxesGroup` Axes group reference
- `Execute` The block is activated on rising edge

**Outputs**

- `yAxesGroup` Axes group reference
- `Done` Algorithm finished
- `Busy` Algorithm not finished yet
- `Error` Error occurred
- `ErrorID` Result of the last operation

`i ..... REXYGEN general error`
**MC_GroupDisable** – Changes the state of a group to **GroupDisabled**

**Function Description**

The function block `MC_GroupDisable` changes the state for the group `uAxesGroup` to "GroupDisabled". If the axes are not standing still while issuing this command the state of the group is changed to "Stopping". It is mean stopping with the maximal allowed deceleration. When stopping is done the state of the group is changed to "GroupDisabled".

**Inputs**

- **uAxesGroup**: Axes group reference
- **Execute**: The block is activated on rising edge

**Outputs**

- **yAxesGroup**: Axes group reference
- **Done**: Algorithm finished
- **Busy**: Algorithm not finished yet
- **Error**: Error occurred
- **ErrorID**: Result of the last operation

... REXYGEN general error
MC_SetCartesianTransform – Sets Cartesian transformation

Block Symbol Licence: COORDINATED MOTION

Function Description

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

Inputs

| 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      | Result of the last operation | error |

i ..... REXYGEN general error
Cartesian Machine Coordinate System

\[ z' = z \]
\[ x = x' \]
\[ y = y' \]
\[ y'' = y' \]
\[ x' = x'' \]
\[ y'' = y'' \]
\[ z'' = z'' \]
\[ y''' = y''' \]
\[ z''' = z''' \]
\[ x''' = x''' \]

\[ x \]
\[ y \]
\[ z \]

Trans

RotZ

RotY

RotX

TransX

TransY

RotZ
MC_ReadCartesianTransform — Reads the parameter of the cartesian transformation

**Block Symbol**

![Block Symbol](image)

**Function Description**

The function block **MC_ReadCartesianTransform** reads the parameter of the cartesian transformation that is active between the MCS and PCS. The parameters are valid only if the output **Valid** is true which is achieved by setting the input **Enable** on true. If more than one transformation is active, the resulting cartesian transformation is given.

**Inputs**

- **uAxesGroup** Axes group reference
- **Enable** Block function is enabled

**Outputs**

- **yAxesGroup** Axes group reference
- **Valid** Output value is valid
- **Busy** Algorithm not finished yet
- **TransX** X-component of translation vector
- **TransY** Y-component of translation vector
- **TransZ** Z-component of translation vector
- **RotAngle1** Rotation angle component
- **RotAngle2** Rotation angle component
- **RotAngle3** Rotation angle component
- **Error** Error occurred
- **ErrorID** Result of the last operation

... REXYGEN 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
- **CommandAborted**: Algorithm was aborted
- **Error**: Error occurred
<table>
<thead>
<tr>
<th>ErrorID</th>
<th>Result of the last operation</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>..... REXYGEN general error</td>
<td></td>
</tr>
</tbody>
</table>
MC_GroupReadActualPosition – Read actual position in the selected coordinate system

Block Symbol

License: COORDINATED MOTION

Function Description

The function block MC_GroupReadActualPosition returns the actual position in the selected coordinate system of an axes group. The position is valid only if the output Valid is true which is achieved by setting the input Enable on true.

Inputs

- **uAxesGroup** Axes group reference
- **Enable** Block function is enabled
- **CoordSystem** Reference to the coordinate system used
  - 1 ... ACS
  - 2 ... MCS
  - 3 ... PCS

Outputs

- **yAxesGroup** Axes group reference
- **Valid** Output value is valid
- **Busy** Algorithm not finished yet
- **Error** Error occurred
- **ErrorID** Result of the last operation
  - i ... REXYGEN general error
- **Position** xxx
MC_GroupReadActualVelocity — Read actual velocity in the selected coordinate system

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

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Axes group reference</td>
</tr>
<tr>
<td>Enable</td>
<td>Block function is enabled</td>
</tr>
<tr>
<td>CoordSystem</td>
<td>Reference to the coordinate system used</td>
</tr>
<tr>
<td></td>
<td>1 . . . . ACS</td>
</tr>
<tr>
<td></td>
<td>2 . . . . MCS</td>
</tr>
<tr>
<td></td>
<td>3 . . . . PCS</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxesGroup</td>
<td>Axes group reference</td>
</tr>
<tr>
<td>Valid</td>
<td>Output value is valid</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td></td>
<td>i . . . . REXYGEN general error</td>
</tr>
<tr>
<td>Velocity</td>
<td>xxx</td>
</tr>
</tbody>
</table>
MC_GroupReadActualAcceleration – Read actual acceleration in the selected coordinate system

Function Description

The function block `MC_GroupReadActualAcceleration` returns the actual velocity in the selected coordinate system of an axes group. The position is valid only if the output `Valid` is true which is achieved by setting the input `Enable` on true.

Inputs

- **uAxesGroup** Axes group reference
- **Enable** Block function is enabled
- **CoordSystem** Reference to the coordinate system used
  - 1 ...... ACS
  - 2 ...... MCS
  - 3 ...... PCS

Outputs

- **yAxesGroup** Axes group reference
- **Valid** Output value is valid
- **Busy** Algorithm not finished yet
- **Error** Error occurred
- **ErrorID** Result of the last operation
  - i ...... REXYGEN general error
- **Acceleration** xxx
MC_GroupStop – Stopping a group movement

Block Symbol

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

Inputs

| uAxesGroup | Axes group reference | reference |
| Execute    | The block is activated on rising edge | bool |
| Deceleration | Maximal allowed deceleration [unit/s²] | double |
| Jerk       | Maximal allowed jerk [unit/s³] | double |

Outputs

| yAxesGroup | Axes group reference | reference |
| Done       | Algorithm finished   | bool |
| CommandAborted | Algorithm was aborted | bool |
| Busy       | Algorithm not finished yet | bool |
| Active     | The block is controlling the axis | bool |
| Error      | Error occurred       | bool |
| ErrorID    | Result of the last operation | error |

i ..... REXYGEN general error
MC_GroupHalt – Stopping a group movement (interruptible)

Block Symbol

Function Description

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

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Axes group reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Maximal allowed deceleration ([\text{unit}/s^2])</td>
</tr>
<tr>
<td>Jerk</td>
<td>Maximal allowed jerk ([\text{unit}/s^3])</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxesGroup</td>
<td>Axes group reference</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>Active</td>
<td>The block is controlling the axis</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
<tr>
<td>i . . . . .</td>
<td>REXYGEN general error</td>
</tr>
</tbody>
</table>
MC_GroupInterrupt, MCP_GroupInterrupt – Read a group interrupt

Block Symbols

**Licence: COORDINATED MOTION**

Function Description

*The MC_GroupInterrupt and MCP_GroupInterrupt blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the MCP version of the block.*

The function block `MC_GroupInterrupt` interrupts the on-going motion and stops the group from moving, however does not abort the interrupted motion (meaning that at the interrupted FB the output `CommandAborted` will not be Set, `Busy` is still high and `Active` is reset). It stores all relevant track or path information internally at the moment it becomes active. The `uAxesGroup` stays in the original state even if the velocity zero is reached and the `Done` output is set.

Note 1: This function block is complementary to the function block `MC_GroupContinue` which execution the `uAxesGroup` state is reset to the original state (before `MC_GroupInterrupt` execution)

**Inputs**

- **uAxesGroup**: Axes group reference
- **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
- **Busy**: Algorithm not finished yet
- **CommandAborted**: Algorithm was aborted
- **Error**: Error occurred
- **ErrorID**: Result of the last operation
  
i ..... REXYGEN general error
MC_GroupContinue – Continuation of interrupted movement

Block Symbol

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

<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>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxesGroup</td>
<td>Axes group reference</td>
</tr>
<tr>
<td>Done</td>
<td>Algorithm finished</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>CommandAborted</td>
<td>Algorithm was aborted</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ErrorID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>REXYGEN general error</td>
</tr>
</tbody>
</table>
MC_GroupReadStatus – Read a group status

Block Symbol

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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>reference</td>
<td>Axes group reference</td>
</tr>
<tr>
<td>Enable</td>
<td>bool</td>
<td>Block function is enabled</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yAxesGroup</td>
<td>reference</td>
<td>Axes group reference</td>
</tr>
<tr>
<td>Valid</td>
<td>bool</td>
<td>Output value is valid</td>
</tr>
<tr>
<td>Busy</td>
<td>bool</td>
<td>Algorithm not finished yet</td>
</tr>
<tr>
<td>GroupMoving</td>
<td>bool</td>
<td>State GroupMoving</td>
</tr>
<tr>
<td>GroupHoming</td>
<td>bool</td>
<td>State GroupHoming</td>
</tr>
<tr>
<td>GroupErrorStop</td>
<td>bool</td>
<td>State ErrorStop</td>
</tr>
<tr>
<td>GroupStandby</td>
<td>bool</td>
<td>State Standby</td>
</tr>
<tr>
<td>GroupStopping</td>
<td>bool</td>
<td>State Stopping</td>
</tr>
<tr>
<td>GroupDisabled</td>
<td>bool</td>
<td>State Disabled</td>
</tr>
<tr>
<td>ConstantVelocity</td>
<td>bool</td>
<td>Constant velocity motion</td>
</tr>
<tr>
<td>Accelerating</td>
<td>bool</td>
<td>Accelerating</td>
</tr>
<tr>
<td>Decelerating</td>
<td>bool</td>
<td>Decelerating</td>
</tr>
<tr>
<td>InPosition</td>
<td>bool</td>
<td>Symptom achieve the desired position</td>
</tr>
<tr>
<td>Error</td>
<td>bool</td>
<td>Error occurred</td>
</tr>
<tr>
<td>ErrorID</td>
<td>error</td>
<td>Result of the last operation</td>
</tr>
</tbody>
</table>

REXYGEN general error
MC_GroupReadError – Read a group error

Block Symbol

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

<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>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>error</td>
</tr>
<tr>
<td>GroupErrorID</td>
<td>Result of the last operation</td>
<td>error</td>
</tr>
</tbody>
</table>
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 reference |
| Execute    | The block is activated on rising edge bool |

Outputs

| yAxesGroup | Axes group reference reference |
| Done       | Algorithm finished bool |
| Busy       | Algorithm not finished yet bool |
| Error      | Error occurred bool |
| ErrorID    | Result of the last operation error |

i ..... REXYGEN general error
MC_MoveLinearAbsolute – Linear move to position (absolute coordinates)

Block Symbol

License: COORDINATED MOTION

Function Description

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

Inputs

- **uAxesGroup**: Axes group reference
- **Execute**: The block is activated on rising edge
- **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
- **BufferMode**: Buffering mode
  - 1: Aborting
  - 2: Buffered
  - 3: Blending low
  - 4: Blending high
  - 5: Blending previous
  - 6: Blending next
CHAPTER 20. MC_COORD – MOTION CONTROL - COORDINATED MOVEMENT BLOCKS

TransitionMode  Transition mode in blending mode  long
1 . . . . . . TMNone
2 . . . . . . TMStartVelocity
3 . . . . . . TMConstantVelocity
4 . . . . . . TMCornerDistance
5 . . . . . . TMMaxCornerDeviation
11 . . . . Smooth

TransitionParameter  Parametr for transition (depends on transition mode)  double

Outputs

yAxesGroup  Axes group reference  reference
Done  Algorithm finished  bool
CommandAborted  Algorithm was aborted  bool
Busy  Algorithm not finished yet  bool
Active  The block is controlling the axis  bool
Error  Error occurred  bool
ErrorID  Result of the last operation  error
    i . . . . REXYGEN general error
Sequence of two complete motions (Done>Execute)
CHAPTER 20. 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

- **Velocity [rad/s]**

### Position AxesGroup

- **Position [rad]**
  - **commanded position x,y**
MC_MoveLinearRelative – Linear move to position (relative to execution point)

Block Symbol

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

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Axes group reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>Distance</td>
<td>Array of coordinates (relative distances and orientations)</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>Jerk</td>
<td>Maximal allowed jerk [unit/s^3]</td>
</tr>
<tr>
<td>CoordSystem</td>
<td>Reference to the coordinate system used</td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reference</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td></td>
</tr>
<tr>
<td>long</td>
<td></td>
</tr>
</tbody>
</table>

MC_MoveLinearRelative
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 reference
Done Algorithm finished bool
CommandAborted Algorithm was aborted bool
Busy Algorithm not finished yet bool
Active The block is controlling the axis bool
Error Error occurred bool
ErrorID Result of the last operation error
i ...... REXYGEN general error
McCoord - Motion Control - Coordinated Movement Blocks

1. Execute - MC_MoveLinearRelative -- Function Block 1

2. Active - MC_MoveLinearRelative -- Function Block 1

3. Done - MC_MoveLinearRelative -- Function Block 1

4. Execute - MC_MoveLinearRelative -- Function Block 2

5. Active - MC_MoveLinearRelative -- Function Block 2

6. Done - MC_MoveLinearRelative -- Function Block 2

Velocity Axes Group:

Position Axes Group:

commanded position x,y

x-axis

y-axis
MC_MoveCircularAbsolute – Circular move to position (absolute coordinates)

Block Symbol

![Block Symbol Diagram]

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>CircMode</td>
<td>Specifies the meaning of the input signals AuxPoint and CircDirection</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>1 ...... BORDER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ...... CENTER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 ...... RADIUS</td>
<td></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>
</tbody>
</table>
### CHAPTER 20. MC_COORD – MOTION CONTROL - COORDINATED MOVEMENT BLOCKS

<table>
<thead>
<tr>
<th>PathChoice</th>
<th>Choice of path</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Clockwise</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CounterClockwise</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Velocity</th>
<th>Maximal allowed velocity [unit/s]</th>
<th>double</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Acceleration</th>
<th>Maximal allowed acceleration [unit/s²]</th>
<th>double</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Jerk</th>
<th>Maximal allowed jerk [unit/s³]</th>
<th>double</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>CoordSystem</th>
<th>Reference to the coordinate system used</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACS</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>MCS</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PCS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BufferMode</th>
<th>Buffering mode</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aborting</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Buffered</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Blending low</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Blending high</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Blending previous</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Blending next</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TransitionMode</th>
<th>Transition mode in blending mode</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TMNone</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TMStartVelocity</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>TMConstantVelocity</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>TMCornerDistance</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>TMMaxCornerDeviation</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Smooth</td>
<td></td>
</tr>
</tbody>
</table>

| TransitionParameter | Parametr for transition (depends on transition mode) | double |

### Outputs

<table>
<thead>
<tr>
<th>yAxesGroup</th>
<th>Axes group reference</th>
<th>reference</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Done</th>
<th>Algorithm finished</th>
<th>bool</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>CommandAborted</th>
<th>Algorithm was aborted</th>
<th>bool</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Busy</th>
<th>Algorithm not finished yet</th>
<th>bool</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Active</th>
<th>The block is controlling the axis</th>
<th>bool</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Error</th>
<th>Error occurred</th>
<th>bool</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>ErrorID</th>
<th>Result of the last operation</th>
<th>error</th>
</tr>
</thead>
</table>

| i            | REXYGEN general error           |       |
MC_MoveCircularAbsolute - Example

Starting point 1

AuxPoint1

EndPoint1 = Starting point 2

AuxPoint2

AuxPointArray 1

AuxPointArray 2

EndPointArray 1

EndPointArray 2
Execute MC_MoveCircularAbsolute -- Function Block 1

Active MC_MoveCircularAbsolute -- Function Block 1

Done MC_MoveCircularAbsolute -- Function Block 1

Execute MC_MoveCircularAbsolute -- Function Block 2

Active MC_MoveCircularAbsolute -- Function Block 2

Done MC_MoveCircularAbsolute -- Function Block 2

Velocity AxesGroup

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

Block Symbol

Licence: COORDINATED MOTION

Function Description

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

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uAxesGroup</td>
<td>Axes group reference</td>
</tr>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>CircMode</td>
<td>Specifies the meaning of the input signals AuxPoint and CircDirection</td>
</tr>
<tr>
<td>AuxPoint</td>
<td>Next coordinates to define circle (depend on CircMode)</td>
</tr>
<tr>
<td>EndPoint</td>
<td>Target axes coordinates position</td>
</tr>
</tbody>
</table>

```plaintext
CircMode = BORDER
CircMode = CENTER
CircMode = RADIUS
```
### 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
- ACS
- MCS
- PCS

### BufferMode
Buffering mode
- Aborting
- Buffered
- Blending low
- Blending high
- Blending previous
- Blending next

### TransitionMode
Transition mode in blending mode
- TMNone
- TMStartVelocity
- TMConstantVelocity
- TMCornerDistance
- TMMaxCornerDeviation
- Smooth

### TransitionParameter
Parameter 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 Result of the last operation
- i .... REXYGEN general error
MC_MoveCircularRelative - Example

Starting point 1

AuxPoint1

AuxPointArray_1

EndPoint1=
Starting point 2

AuxPointArray_2

EndPointArray_1

AuxPointArray_2

EndPointArray_1

AuxPoint2

EndPoint2
Execute - MC_MoveCircularRelative — Function Block 1

Active - MC_MoveCircularRelative — Function Block 1

Done - MC_MoveCircularRelative — Function Block 1

Execute - MC_MoveCircularRelative — Function Block 2

Active - MC_MoveCircularRelative — Function Block 2

Done - MC_MoveCircularRelative — Function Block 2

Velocity AxesGroup

Position AxesGroup

Time [s]

Position [rad]

Velocity [rad/s]

commanded position x

commanded position y

actual position x

commanded position y

actual position y
MC_MoveDirectAbsolute – Direct move to position (absolute coordinates)

Block Symbol Licence: COORDINATED MOTION

Function Description

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

Inputs

| uAxesGroup  | Axes group reference reference |
| Execute     | The block is activated on rising edge bool |
| Position    | Array of coordinates (positions and orientations) reference |
| CoordSystem | Reference to the coordinate system used long |
|             | 1 ..... ACS |
|             | 2 ..... MCS |
|             | 3 ..... PCS |
| BufferMode  | Buffering mode long |
|             | 1 ..... Aborting |
|             | 2 ..... Buffered |
|             | 3 ..... Blending low |
|             | 4 ..... Blending high |
|             | 5 ..... Blending previous |
|             | 6 ..... Blending next |
| TransitionMode | Transition mode in blending mode long |
|             | 1 ..... TMNone |
|             | 2 ..... TMStartVelocity |
|             | 3 ..... TMConstantVelocity |
|             | 4 ..... TMCornerDistance |
|             | 5 ..... TMMaxCornerDeviation |
|             | 11 ..... Smooth |
| TransitionParameter | Parametr for transition (depends on transition mode) double |
Outputs

\textbf{yAxesGroup}  Axes group reference

\textbf{Done}  Algorithm finished

\textbf{CommandAborted}  Algorithm was aborted

\textbf{Busy}  Algorithm not finished yet

\textbf{Active}  The block is controlling the axis

\textbf{Error}  Error occurred

\textbf{ErrorID}  Result of the last operation

\textbf{i}  REXYGEN general error
MC_MoveDirectRelative – Direct move to position (relative to execution point)

Block Symbol

Function Description

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

Inputs

<table>
<thead>
<tr>
<th>uAxesGroup</th>
<th>Axes group reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>The block is activated on rising edge</td>
</tr>
<tr>
<td>Distance</td>
<td>Array of coordinates (relative distances and orientations)</td>
</tr>
<tr>
<td>CoordSystem</td>
<td>Reference to the coordinate system used</td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
</tr>
<tr>
<td>TransitionMode</td>
<td>Transition mode in blending mode</td>
</tr>
<tr>
<td>TransitionParameter</td>
<td>Parametr for transition (depends on transition mode)</td>
</tr>
</tbody>
</table>

- uAxesGroup
- Execute
- Distance
- CoordSystem
- BufferMode
- TransitionMode
- TransitionParameter
**Outputs**

- **yAxesGroup**: Axes group reference
- **Done**: Algorithm finished
- **CommandAborted**: Algorithm was aborted
- **Busy**: Algorithm not finished yet
- **Active**: The block is controlling the axis
- **Error**: Error occurred
- **ErrorID**: Result of the last operation

<i>..... REXYGEN general error</i>
MC_MovePath – General spatial trajectory generation

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>Variable</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>TotalTime</td>
<td>Time [s] for whole move</td>
</tr>
<tr>
<td>RampTime</td>
<td>Time [s] for acceleration/deceleration</td>
</tr>
<tr>
<td>CoordSystem</td>
<td>Reference to the coordinate system used</td>
</tr>
<tr>
<td>BufferMode</td>
<td>Buffering mode</td>
</tr>
<tr>
<td>TransitionMode</td>
<td>Transition mode in blending mode</td>
</tr>
<tr>
<td>TransitionParameter</td>
<td>Parametr for transition (depends on transition mode)</td>
</tr>
</tbody>
</table>
**CHAPTER 20. MC_COORD – MOTION CONTROL - COORDINATED MOVEMENT BLOCKS**

RampIn  
RampIn factor (0 = RampIn mode not used)  

**Parameters**

- **pc**  
  Control-points matrix  
  \[
  \begin{bmatrix}
  0.0 & 1.0 & 2.0 \\
  0.0 & 1.0 & 1.0 \\
  0.0 & 1.0 & 0.0
  \end{bmatrix}
  \]

- **pk**  
  Knot-points vector  
  \[
  \begin{bmatrix}
  0.0 & 0.0 & 0.0 & 0.0 & 0.5 & 1.0 & 1.0 \\
  \end{bmatrix}
  \]

- **pw**  
  Weighting vector  
  \[
  \begin{bmatrix}
  1.0 & 1.0 & 1.0 \\
  \end{bmatrix}
  \]

- **pv**  
  Polynoms for feedrate definition  
  \[
  \begin{bmatrix}
  0.0 & 0.05 & 0.95 & 0.0 & 0.1 & 1.0; 0.0 & 0.0 & 0.0 & 0.1 & 0.0 & -0.1; -0.05 & 0.0 & 0.05; 0.0 & 0.0 \\
  \end{bmatrix}
  \]

- **pt**  
  Knot-points (time [s]) for feedrate  
  \[
  \begin{bmatrix}
  0.0 & 1.0 & 10.0 & 11.0 \\
  \end{bmatrix}
  \]

- **user**  
  Only for special edit  
  \[
  \begin{bmatrix}
  0.0 & 1.0 & 2.0 & 3.0 \\
  \end{bmatrix}
  \]

**Outputs**

- **yAxesGroup**  
  Axes group reference  
  Reference

- **Done**  
  Algorithm finished  
  Bool

- **CommandAborted**  
  Algorithm was aborted  
  Bool

- **Busy**  
  Algorithm not finished yet  
  Bool

- **Active**  
  The block is controlling the axis  
  Bool

- **Error**  
  Error occurred  
  Bool

- **ErrorID**  
  Result of the last operation  
  Error

- **i ..... REXXYGEN general error**
MC_GroupSetOverride – Set group override factors

Block Symbol

Licence: COORDINATED MOTION

Function Description

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

Inputs

<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>
<tr>
<td>VelFactor</td>
<td>Velocity multiplication factor</td>
<td>double</td>
</tr>
<tr>
<td>AccFactor</td>
<td>Acceleration/deceleration multiplication factor</td>
<td>double</td>
</tr>
<tr>
<td>JerkFactor</td>
<td>Jerk multiplication factor</td>
<td>double</td>
</tr>
</tbody>
</table>

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>diff</td>
<td>Deadband (difference for recalculation)</td>
<td>⊕0.05  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>Enabled</td>
<td>Signal that the override faktor are set successfully</td>
<td>bool</td>
</tr>
<tr>
<td>Busy</td>
<td>Algorithm not finished yet</td>
<td>bool</td>
</tr>
<tr>
<td>Error</td>
<td>Error occurred</td>
<td>bool</td>
</tr>
<tr>
<td>ErrorID</td>
<td>Result of the last operation</td>
<td>error</td>
</tr>
<tr>
<td>i .....</td>
<td>REXYGEN general error</td>
<td></td>
</tr>
</tbody>
</table>
Appendix A

Licensing options

From the licensing point of view, there are several versions of the RexCore runtime module to provide maximum flexibility for individual projects. The table below compares the individual variants.

The function blocks are divided into several licensing groups. The STANDARD function blocks are always available, the other groups require activation by a corresponding licence.

<table>
<thead>
<tr>
<th></th>
<th>RexCore DEMO</th>
<th>RexCore Starter</th>
<th>RexCore Plus</th>
<th>RexCore Professional</th>
<th>RexCore Ultimate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function blocks</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STANDARD</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>ADVANCED</td>
<td>●</td>
<td>–</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>REXLANG</td>
<td>●</td>
<td>–</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>MOTION CONTROL</td>
<td>●</td>
<td>–</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>COORDINATED MOTION</td>
<td>●</td>
<td>–</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>AUTOTUNING</td>
<td>–</td>
<td>–</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>MATRIX</td>
<td>●</td>
<td>–</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td><strong>I/O drivers</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic I/O drivers</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Additional I/O drivers</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

(●... included, ○... optional, –... not available)

See Appendix B for details about licensing of individual function blocks.
Appendix B

Licensing of individual function blocks

To maximize flexibility for individual projects, function blocks of the REXYGEN system are divided into several licensing groups. The table below shows the groups the function blocks belong to. See Appendix A for detailed information about the individual licensing options.

<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STANDARD</td>
</tr>
<tr>
<td>ABS_</td>
<td>•</td>
</tr>
<tr>
<td>ABSROT</td>
<td></td>
</tr>
<tr>
<td>ACD</td>
<td>•</td>
</tr>
<tr>
<td>ADD</td>
<td>•</td>
</tr>
<tr>
<td>ADDHEXD</td>
<td>•</td>
</tr>
<tr>
<td>ADDOCT</td>
<td>•</td>
</tr>
<tr>
<td>ADDQUAD</td>
<td>•</td>
</tr>
<tr>
<td>AFLUSH</td>
<td>•</td>
</tr>
<tr>
<td>ALB</td>
<td>•</td>
</tr>
<tr>
<td>ALBI</td>
<td>•</td>
</tr>
<tr>
<td>ALN</td>
<td>•</td>
</tr>
<tr>
<td>ALNI</td>
<td>•</td>
</tr>
<tr>
<td>AND_</td>
<td>•</td>
</tr>
<tr>
<td>ANDHEXD</td>
<td>•</td>
</tr>
<tr>
<td>ANDOCT</td>
<td>•</td>
</tr>
<tr>
<td>ANDQUAD</td>
<td>•</td>
</tr>
<tr>
<td>ANLS</td>
<td>•</td>
</tr>
<tr>
<td>ARC</td>
<td>•</td>
</tr>
<tr>
<td>ARLY</td>
<td>•</td>
</tr>
<tr>
<td>ASW</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>ATMT</td>
<td>•</td>
</tr>
<tr>
<td>AVG</td>
<td>•</td>
</tr>
<tr>
<td>AVS</td>
<td>•</td>
</tr>
<tr>
<td>•</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>BDHEXD</td>
<td>•</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>BMOCT</td>
<td>•</td>
</tr>
<tr>
<td>BPF</td>
<td>•</td>
</tr>
<tr>
<td>CanItem</td>
<td>•</td>
</tr>
<tr>
<td>CanRecv</td>
<td>•</td>
</tr>
<tr>
<td>CanSend</td>
<td>•</td>
</tr>
<tr>
<td>CDELTSSM</td>
<td>ADVANCED</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>ADVANCED</td>
</tr>
<tr>
<td>DATE_</td>
<td>•</td>
</tr>
<tr>
<td>DATETIME</td>
<td>•</td>
</tr>
<tr>
<td>DDELTSSM</td>
<td>ADVANCED</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>DFIR</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>ADVANCED</td>
</tr>
<tr>
<td>EAS</td>
<td>•</td>
</tr>
<tr>
<td>EATMT</td>
<td>ADVANCED</td>
</tr>
<tr>
<td>Function block name</td>
<td>Licensing group</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td>STANDARD</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>
<tr>
<td>FNXY</td>
<td>•</td>
</tr>
<tr>
<td>FOPDT</td>
<td>•</td>
</tr>
<tr>
<td>FRID</td>
<td></td>
</tr>
<tr>
<td>From</td>
<td>•</td>
</tr>
<tr>
<td>GAIN</td>
<td>•</td>
</tr>
<tr>
<td>GETPA</td>
<td>•</td>
</tr>
<tr>
<td>GETPB</td>
<td>•</td>
</tr>
<tr>
<td>GETPI</td>
<td>•</td>
</tr>
<tr>
<td>GETPR</td>
<td>•</td>
</tr>
<tr>
<td>GETPS</td>
<td>•</td>
</tr>
<tr>
<td>Goto</td>
<td>•</td>
</tr>
<tr>
<td>GotoTagVisibility</td>
<td>•</td>
</tr>
<tr>
<td>GRADS</td>
<td></td>
</tr>
<tr>
<td>HMI</td>
<td>•</td>
</tr>
<tr>
<td>HTTP</td>
<td></td>
</tr>
<tr>
<td>HTTP2</td>
<td></td>
</tr>
<tr>
<td>I3PM</td>
<td></td>
</tr>
<tr>
<td>IADD</td>
<td>•</td>
</tr>
<tr>
<td>IDIV</td>
<td>•</td>
</tr>
<tr>
<td>IMOD</td>
<td>•</td>
</tr>
<tr>
<td>IMUL</td>
<td>•</td>
</tr>
<tr>
<td>INFO</td>
<td>•</td>
</tr>
<tr>
<td>INHEXD</td>
<td>•</td>
</tr>
<tr>
<td>INOCT</td>
<td>•</td>
</tr>
<tr>
<td>Import</td>
<td>•</td>
</tr>
<tr>
<td>INQUAD</td>
<td>•</td>
</tr>
<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>
</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>ISSW</td>
<td>●</td>
</tr>
<tr>
<td>ISUB</td>
<td>●</td>
</tr>
<tr>
<td>ITOI</td>
<td>●</td>
</tr>
<tr>
<td>ITOS</td>
<td>●</td>
</tr>
<tr>
<td>KDER</td>
<td></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>
<tr>
<td>LPF</td>
<td>●</td>
</tr>
<tr>
<td>MC_AccelerationProfile</td>
<td></td>
</tr>
<tr>
<td>MC_AddAxisToGroup</td>
<td></td>
</tr>
<tr>
<td>MC_CamIn</td>
<td></td>
</tr>
<tr>
<td>MC_CamOut</td>
<td></td>
</tr>
<tr>
<td>MC_CombineAxes</td>
<td></td>
</tr>
<tr>
<td>MC_GearIn</td>
<td></td>
</tr>
<tr>
<td>MC_GearInPos</td>
<td></td>
</tr>
<tr>
<td>MC_GroupContinue</td>
<td></td>
</tr>
<tr>
<td>MC_GroupDisable</td>
<td></td>
</tr>
<tr>
<td>MC_GroupEnable</td>
<td></td>
</tr>
<tr>
<td>MC_GroupHalt</td>
<td></td>
</tr>
<tr>
<td>MC_GroupInterrupt</td>
<td></td>
</tr>
<tr>
<td>MC_GroupReadActualAcceleration</td>
<td></td>
</tr>
<tr>
<td>MC_GroupReadActualPosition</td>
<td></td>
</tr>
<tr>
<td>MC_GroupReadActualVelocity</td>
<td></td>
</tr>
<tr>
<td>MC_GroupReadError</td>
<td></td>
</tr>
<tr>
<td>MC_GroupReadStatus</td>
<td></td>
</tr>
<tr>
<td>MC_GroupReset</td>
<td></td>
</tr>
<tr>
<td>MC_GroupSetOverride</td>
<td></td>
</tr>
<tr>
<td>MC_GroupSetPosition</td>
<td></td>
</tr>
<tr>
<td>MC_GroupStop</td>
<td></td>
</tr>
<tr>
<td>MC_Halt</td>
<td></td>
</tr>
<tr>
<td>MC_HaltSuperimposed</td>
<td></td>
</tr>
<tr>
<td>MC_Home</td>
<td></td>
</tr>
<tr>
<td>MC_MoveAbsolute</td>
<td></td>
</tr>
<tr>
<td>MC_MoveAdditive</td>
<td></td>
</tr>
<tr>
<td>MC_MoveCircularAbsolute</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>MC_MoveCircularRelative</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_MoveContinuousAbsolute</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveContinuousRelative</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveDirectAbsolute</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_MoveDirectRelative</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_MoveLinearAbsolute</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_MoveLinearRelative</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_MovePath</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_MovePath_PH</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_MoveRelative</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_MoveSuperimposed</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_PhasingVelocity</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_PhasingAbsolute</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_PhasingRelative</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_PositionProfile</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_Power</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_ReadActualPosition</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_ReadAxisError</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_ReadBoolParameter</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_ReadCartesianTransform</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_ReadParameter</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_ReadStatus</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_Reset</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_SetCartesianTransform</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_SetOverride</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_Stop</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_TorqueControl</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_UngroupAllAxes</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MC_VelocityProfile</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_WriteBoolParameter</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MC_WriteParameter</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_AccelerationProfile</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_CamIn</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_CamTableSelect</td>
<td>MOTION CONTROL</td>
</tr>
<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</td>
</tr>
<tr>
<td>MCP_GroupInterrupt</td>
<td>COORDINATED MOTION</td>
</tr>
</tbody>
</table>

The list continues on the next page...
### APPENDIX B. LICENSING OF INDIVIDUAL FUNCTION BLOCKS

<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STANDARD</strong></td>
<td><strong>Other</strong></td>
</tr>
<tr>
<td>MCP_GroupSetOverride</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MCP_GroupSetPosition</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MCP_GroupStop</td>
<td>COORDINATED MOTION</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</td>
</tr>
<tr>
<td>MCP_MoveCircularRelative</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MCP_MoveContinuousAbsolute</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_MoveContinuousRelative</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>MCP_MoveDirectAbsolute</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MCP_MoveDirectRelative</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MCP_MoveLinearAbsolute</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MCP_MoveLinearRelative</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MCP_MovePath</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>MCP_MovePath_PH</td>
<td>COORDINATED MOTION</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</td>
</tr>
<tr>
<td>MCP_SetKinTransform_Arm</td>
<td>COORDINATED MOTION</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><strong>MCU</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MDL</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MDLI</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MID</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MINMAX</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MODULE</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MP</strong></td>
<td></td>
</tr>
<tr>
<td>MqttPublish</td>
<td>MQTTDRV</td>
</tr>
<tr>
<td>MqttSubscribe</td>
<td>MQTTDRV</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>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>ORHEXD</td>
<td>•</td>
</tr>
<tr>
<td>ORQCT</td>
<td>•</td>
</tr>
<tr>
<td>ORQUAD</td>
<td>•</td>
</tr>
<tr>
<td>OSCALL</td>
<td>•</td>
</tr>
<tr>
<td>OUTHEXD</td>
<td>•</td>
</tr>
<tr>
<td>OUTOCT</td>
<td>•</td>
</tr>
<tr>
<td>Outport</td>
<td>•</td>
</tr>
<tr>
<td>OUTQUAD</td>
<td>•</td>
</tr>
<tr>
<td>OUTRHEXD</td>
<td>•</td>
</tr>
<tr>
<td>OUTROCT</td>
<td>•</td>
</tr>
<tr>
<td>OUTRQUAD</td>
<td>•</td>
</tr>
<tr>
<td>OUTRSTD</td>
<td>•</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>•</td>
</tr>
<tr>
<td>PIDE</td>
<td>•</td>
</tr>
<tr>
<td>PIDS</td>
<td>•</td>
</tr>
<tr>
<td>PIDU</td>
<td>•</td>
</tr>
<tr>
<td>PIDUI</td>
<td>•</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>•</td>
</tr>
<tr>
<td>PWM</td>
<td>•</td>
</tr>
<tr>
<td>QFC</td>
<td>•</td>
</tr>
</tbody>
</table>

*The list continues on the next page.*
## APPENDIX B. LICENSING OF INDIVIDUAL FUNCTION BLOCKS

<table>
<thead>
<tr>
<th>Function block name</th>
<th>Licensing group</th>
</tr>
</thead>
<tbody>
<tr>
<td>QFD</td>
<td>STANDARD ADVANCED</td>
</tr>
<tr>
<td>QTASK</td>
<td>·</td>
</tr>
<tr>
<td>RDC</td>
<td>STANDARD ADVANCED</td>
</tr>
<tr>
<td>REC</td>
<td>·</td>
</tr>
<tr>
<td>REGEXP</td>
<td>·</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</td>
</tr>
<tr>
<td>RM_Axis</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>RM_AxisOut</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>RM_AxisSpline</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>RM_Feed</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>RM_Gcode</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>RM_GroupTrack</td>
<td>COORDINATED MOTION</td>
</tr>
<tr>
<td>RM_Track</td>
<td>MOTION CONTROL</td>
</tr>
<tr>
<td>RS</td>
<td>·</td>
</tr>
<tr>
<td>RTOI</td>
<td>·</td>
</tr>
<tr>
<td>RTOS</td>
<td>·</td>
</tr>
<tr>
<td>RTOV</td>
<td>·</td>
</tr>
<tr>
<td>S1OF2</td>
<td>STANDARD ADVANCED</td>
</tr>
<tr>
<td>SAI</td>
<td>STANDARD ADVANCED</td>
</tr>
<tr>
<td>SAT</td>
<td>·</td>
</tr>
<tr>
<td>SC2FA</td>
<td>AUTOTUNING</td>
</tr>
<tr>
<td>SCU</td>
<td>·</td>
</tr>
<tr>
<td>SCUV</td>
<td>·</td>
</tr>
<tr>
<td>SEL</td>
<td>·</td>
</tr>
<tr>
<td>SELHEXD</td>
<td>·</td>
</tr>
<tr>
<td>SELOCT</td>
<td>·</td>
</tr>
<tr>
<td>SELQUAD</td>
<td>·</td>
</tr>
<tr>
<td>SELSOCT</td>
<td>·</td>
</tr>
<tr>
<td>SELU</td>
<td>·</td>
</tr>
<tr>
<td>SETPA</td>
<td>·</td>
</tr>
<tr>
<td>SETPB</td>
<td>·</td>
</tr>
<tr>
<td>SETPB</td>
<td>·</td>
</tr>
<tr>
<td>SEPI</td>
<td>·</td>
</tr>
<tr>
<td>SETPR</td>
<td>·</td>
</tr>
<tr>
<td>SETPS</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>SG</td>
<td>⬤</td>
</tr>
<tr>
<td>SGI</td>
<td>⬤</td>
</tr>
<tr>
<td>SGSILP</td>
<td></td>
</tr>
<tr>
<td>SHIFTOCT</td>
<td>⬤</td>
</tr>
<tr>
<td>SHLD</td>
<td>⬤</td>
</tr>
<tr>
<td>SILO</td>
<td>⬤</td>
</tr>
<tr>
<td>SILOS</td>
<td>⬤</td>
</tr>
<tr>
<td>SINT</td>
<td>⬤</td>
</tr>
<tr>
<td>SLEEP</td>
<td>⬤</td>
</tr>
<tr>
<td>SMHCC</td>
<td></td>
</tr>
<tr>
<td>SMHCCA</td>
<td></td>
</tr>
<tr>
<td>SMTP</td>
<td></td>
</tr>
<tr>
<td>SOPDT</td>
<td>⬤</td>
</tr>
<tr>
<td>SPIKE</td>
<td></td>
</tr>
<tr>
<td>SQR</td>
<td>⬤</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>SWR</td>
<td>⬤</td>
</tr>
<tr>
<td>SWU</td>
<td>⬤</td>
</tr>
<tr>
<td>SWMR</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>TRNDV</td>
<td>⬤</td>
</tr>
<tr>
<td>TSE</td>
<td>⬤</td>
</tr>
<tr>
<td>UTOI</td>
<td>⬤</td>
</tr>
<tr>
<td>VDEL</td>
<td>⬤</td>
</tr>
<tr>
<td>VIN</td>
<td></td>
</tr>
<tr>
<td>VOUT</td>
<td></td>
</tr>
<tr>
<td>VTOR</td>
<td>⬤</td>
</tr>
<tr>
<td>WSCH</td>
<td>⬤</td>
</tr>
<tr>
<td>WWW</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>ZV4IS</td>
<td>STANDARD</td>
</tr>
<tr>
<td></td>
<td>ADVANCED</td>
</tr>
</tbody>
</table>
Appendix C

Error codes of the REXYGEN system

Success codes

0 ........ Success
-1 ...... False
-2 ...... First value is greater
-3 ...... Second value is greater
-4 ...... Parameter changed
-5 ...... Success, no server transaction done
-6 ...... Value too big
-7 ...... Value too small
-8 ...... Operation in progress
-9 ...... REXYGEN I/O driver warning
-10 ...... No more archive items
-11 ..... Object is array
-12 ..... Closed
-13 ..... End of file
-14 ..... Parameter may be incorrect

General failure codes

-100 ..... Not enough memory
-101 ..... Assertion failure
-102 ..... Timeout
-103 ..... General input variable error
-104 ..... Invalid configuration version
-105 ..... Not implemented
-106 ..... Invalid parameter
-107 ..... COM/OLE error
-108 ..... REXYGEN Module error - some driver or block is not installed or licensed
APPENDIX C. ERROR CODES OF THE REXYGEN SYSTEM

-109 .... REXYGEN I/O driver error
-110 .... Task creation error
-111 .... Operating system call error
-112 .... Invalid operating system version
-113 .... Access denied by operating system
-114 .... Block period has not been set
-115 .... Initialization failed
-116 .... REXYGEN configuration is being changed
-117 .... Invalid target device
-118 .... Access denied by REXYGEN security mechanism
-119 .... Block or object is not installed or licensed
-120 .... Checksum mismatch
-121 .... Object already exists
-122 .... Object doesn’t exist
-123 .... System user doesn’t belong to any REXYGEN group
-124 .... Password mismatch
-125 .... Bad user name or password
-126 .... Target device is not compatible
-127 .... Resource is locked by another module and can not be used
-128 .... String is not valid in UTF8 codepage
-129 .... Start of executive not allowed

Class registration, symbol and validation error codes

-200 .... Class not registered
-201 .... Class already registered
-202 .... Not enough space for registry
-203 .... Registry index out of range
-204 .... Invalid context
-205 .... Invalid identifier
-206 .... Invalid input flag
-207 .... Invalid input mask
-208 .... Invalid object type
-209 .... Invalid variable type
-210 .... Invalid object workspace
-211 .... Symbol not found
-212 .... Symbol is ambiguous
-213 .... Range check error
-214 .... Not enough search space
-215 .... Write to read-only variable denied
-216 .... Data not ready
-217 .... Value out of range
-218 .... Input connection error
-219 .... Loop of type UNKNOWN detected
-220 .... REXLANG compilation error
Stream and file system codes

-300 .... Stream overflow
-301 .... Stream underflow
-302 .... Stream send error
-303 .... Stream receive error
-304 .... Stream download error
-305 .... Stream upload error
-306 .... File creation error
-307 .... File open error
-308 .... File close error
-309 .... File read error
-310 .... File write error
-311 .... Invalid format
-312 .... Unable to compress files
-313 .... Unable to extract files

Communication errors

-400 .... Network communication failure
-401 .... Communication already initialized
-402 .... Communication finished successfully
-403 .... Communication closed unexpectedly
-404 .... Unknown command
-405 .... Unexpected command
-406 .... Communication closed unexpectedly, probably 'Too many clients'
-407 .... Communication timeout
-408 .... Target device not found
-409 .... Link failed
-410 .... REXYGEN configuration has been changed
-411 .... REXYGEN executive is being terminated
-412 .... REXYGEN executive was terminated
-413 .... Connection refused
-414 .... Target device is unreachable
-415 .... Unable to resolve target in DNS
-416 .... Error reading from socket
-417 .... Error writing to socket
-418 .... Invalid operation on socket
-419 .... Reserved for socket 1
-420 .... Reserved for socket 2
-421 .... Reserved for socket 3
-422 .... Reserved for socket 4
-423 .... Reserved for socket 5
-424 .... Unable to create SSL context
-425 .... Unable to load certificate
### APPENDIX C. ERROR CODES OF THE REXYGEN SYSTEM

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-426</td>
<td>SSL handshake error</td>
</tr>
<tr>
<td>-427</td>
<td>Certificate verification error</td>
</tr>
<tr>
<td>-428</td>
<td>Reserved for SSL 2</td>
</tr>
<tr>
<td>-429</td>
<td>Reserved for SSL 3</td>
</tr>
<tr>
<td>-430</td>
<td>Reserved for SSL 4</td>
</tr>
<tr>
<td>-431</td>
<td>Reserved for SSL 5</td>
</tr>
<tr>
<td>-432</td>
<td>Relay rejected</td>
</tr>
<tr>
<td>-433</td>
<td>STARTTLS rejected</td>
</tr>
<tr>
<td>-434</td>
<td>Authentication method rejected</td>
</tr>
<tr>
<td>-435</td>
<td>Authentication failed</td>
</tr>
<tr>
<td>-436</td>
<td>Send operation failed</td>
</tr>
<tr>
<td>-437</td>
<td>Receive operation failed</td>
</tr>
<tr>
<td>-438</td>
<td>Communication command failed</td>
</tr>
<tr>
<td>-439</td>
<td>Receiving buffer too small</td>
</tr>
<tr>
<td>-440</td>
<td>Sending buffer too small</td>
</tr>
<tr>
<td>-441</td>
<td>Invalid header</td>
</tr>
<tr>
<td>-442</td>
<td>HTTP server responded with error</td>
</tr>
<tr>
<td>-443</td>
<td>HTTP server responded with redirect</td>
</tr>
<tr>
<td>-444</td>
<td>Operation would block</td>
</tr>
<tr>
<td>-445</td>
<td>Invalid operation</td>
</tr>
<tr>
<td>-446</td>
<td>Communication closed</td>
</tr>
<tr>
<td>-447</td>
<td>Connection cancelled</td>
</tr>
</tbody>
</table>

#### Numerical error codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-500</td>
<td>General numeric error</td>
</tr>
<tr>
<td>-501</td>
<td>Division by zero</td>
</tr>
<tr>
<td>-502</td>
<td>Numeric stack overflow</td>
</tr>
<tr>
<td>-503</td>
<td>Invalid numeric instruction</td>
</tr>
<tr>
<td>-504</td>
<td>Invalid numeric address</td>
</tr>
<tr>
<td>-505</td>
<td>Invalid numeric type</td>
</tr>
<tr>
<td>-506</td>
<td>Not initialized numeric value</td>
</tr>
<tr>
<td>-507</td>
<td>Numeric argument overflow/underflow</td>
</tr>
<tr>
<td>-508</td>
<td>Numeric range check error</td>
</tr>
<tr>
<td>-509</td>
<td>Invalid subvector/submatrix range</td>
</tr>
<tr>
<td>-510</td>
<td>Numeric value too close to zero</td>
</tr>
</tbody>
</table>

#### Archive system codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-600</td>
<td>Archive seek underflow</td>
</tr>
<tr>
<td>-601</td>
<td>Archive semaphore fatal error</td>
</tr>
<tr>
<td>-602</td>
<td>Archive cleared</td>
</tr>
<tr>
<td>-603</td>
<td>Archive reconstructed from saved vars</td>
</tr>
<tr>
<td>-604</td>
<td>Archive reconstructed from normal vars</td>
</tr>
<tr>
<td>-605</td>
<td>Archive check summ error</td>
</tr>
<tr>
<td>-606</td>
<td>Archive integrity error</td>
</tr>
</tbody>
</table>
-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
Appendix C. Error codes of the Rexxygen system

RexVision-related errors

-1000 . . . Result is not evaluated
-1001 . . . The searched object/pattern can not be found
-1002 . . . The search criterion returned more corresponding objects

FMI standard related errors

-1100 . . . FMI Context allocation failure
-1101 . . . Invalid FMU version
-1102 . . . FMI XML parsing error
-1103 . . . FMI Model Exchange kind required
-1104 . . . FMI Co-Simulation kind required
-1105 . . . Could not create FMU loading mechanism
-1106 . . . Instantiation of FMU failed
-1107 . . . Termination of FMU failed
-1108 . . . FMU reset failed
-1109 . . . FMU Experiment setup failed
-1110 . . . Entering FMU initialization mode failed
-1111 . . . Exiting FMU initialization mode failed
-1112 . . . Error getting FMU variable list
-1113 . . . Error getting FMU real variable
-1114 . . . Error setting FMU real variable
-1115 . . . Error getting FMU integer variable
-1116 . . . Error setting FMU integer variable
-1117 . . . Error getting FMU boolean variable
-1118 . . . Error setting FMU boolean variable
-1119 . . . Doing a FMU simulation step failed
-1120 . . . FMU has too many inputs
-1121 . . . FMU has too many outputs
-1122 . . . FMU has too many parameters
Bibliography


Index

REXYGEN Compiler
  compiler, 30
REXYGEN Compiler compiler, 22
TODO
  SRTF DLOG, 36

ABS_, 65, 629
absolute
  position sensor, 101
absolute value, 65
ABSROT, 101, 629
ACD, 271, 629
ADD, 66, 67, 629
ADDHEXD, 67, 629
addition, 66
  extended, 74
  integer, 83
ADDOCT, 66, 67, 97, 629
ADDQUAD, 67, 629
AFLUSH, 277, 629
alarm
  Boolean value, 267
  numerical value, 269
ALB, 267, 629
ALBI, 267, 629
algebraic loop, 30
ALM, 269, 629
ALNI, 269, 629
analog input
  safety, 129
AND_, 232, 233, 629
ANDHEXD, 233, 629
ANDOCT, 232, 233, 629
ANDQUAD, 233, 629
ANLS, 150, 629
application
  of the REXYGEN System, 22
ARC, 20, 23, 268, 270, 272, 275, 277, 278, 629
architecture
  open, 31
archiv, 266
archive, 20
  backed-up memory, 266
  configuration, 20
  disk, 266
  RAM memory, 266
archiving
  delta criterion, 271
ARLY, 163, 629
ASW, 103, 629
ATMT, 14, 234, 241, 298, 307, 311, 630
automaton
  finite-state, 234, 241
average
  moving, 105
AVG, 105, 630
AVS, 14, 106, 630
band
  frequency transmission, 107
band-pass filter, 107
bandwidth, 120
BDHEXD, 236, 241, 630
BDOCT, 236, 241, 630
Bessel filter, 120
binary number
  transformation, 248
binary sequence
  generator, 152, 154
BINS, 152, 154, 630
BIS, 152, 154, 156, 630
INDEX

BITOP, 237, 630
block
description, 15
description format, 15
execution, 35
inputs, 15
mathematic, 14
matrix, 14
modeling, 14
outputs, 15
parameters, 15
symbol, 15
vector, 14
blocks
analog signal processing, 14
data archiving, 14
generators, 14
input-output, 13
logic control, 14
parameter-related, 14
regulation, 14
special, 15
BMHEXD, 238, 241, 630
BMDCT, 238, 241, 630
Boolean complementation, 249
BPF, 107, 630
Butterworth filter, 120
CanItem, 464, 465, 630
CanRecv, 464, 465, 630
CanSend, 467, 630
CDELSSM, 320, 630
circuit
flip-flop Reset-Set, 252
flip-flop Set-Reset, 253
CMP, 108, 630
CNA, 338, 630
CNB, 68, 630
CNDR, 109, 630
CNE, 69, 630
CNI, 70, 630
CNR, 71, 630
CNS, 280, 630
coefficient
relative damping, 107, 120
comparator, 108
compatibility
REXYGEN and Simulink, 30
compensator
lead, 170
lead-lag, 171
nonlinearity, 109
simple nonlinearity, 122
compiler
REXYGEN Compiler, 22, 30
compression, 271
CONCAT, 281, 630
conditioner
nonlinear, 109
configuration
REXYGEN System, 22
archives, 22
computation task, 22
input-output drivers, 22
modules, 22
constant
Boolean, 68
integer, 70
logic, 68
real, 71
control
motion, 15
sequential, 234, 241
control unit
manual, 172
controller
fuzzy logic, 164
PID, 187
PID with gain scheduling, 179
PID with input-defined parameters, 190
PID with relay autotuner, 174
PID with static error, 177
step, 211, 214
with frequency autotuner, 204
conversion
real to integer, 94
COUNT, 17, 239, 630
counter
INDEX

controlled, 239

CSSM, 323, 630

data
remote connection, 435
data storing, 273, 276
data types, 16
DATE, 258, 259, 630
DATETIME, 258, 259, 261, 630
DDELSSM, 325, 630
dead time, 329, 333
DEL, 111, 630
delay
transport, 112
variable, 143
with initialization, 111
DDEL, 112, 630
delta criterion, 271
demultiplexer
bitwise, 236
denominator, 75
DER, 113, 630
derivation, 113, 118
detection
edge, 244
deviation
standard, 115
DFIR, 630
DIF, 72, 630
difference, 72
Display, 44, 630
DIV, 73, 630
division
extended, 75
integer, 87, 88
remainder, 88
two signals, 73
DLL library, 31
driver
REXYGEN System, 27
 rio file extension, 27
configuration data, 27
input-output, 13
input/output, 27
input/output with tasks, 40
user manual, 29
drivers
REXYGEN system, 13
DSSM, 327, 630

EAS, 74, 630
EATMT, 241, 630
edge detection, 244
EDGE, 135, 244, 631
element
three state, 230
EMD, 75, 631
EPC, 37, 426, 631
error
fatal, 33
EVAR, 115, 631
EXEC, 20, 22, 27–29, 31, 33, 34, 38–40, 631
executive
configuration, 13, 19
real-time, 22
RexCore program, 13
external program, 426

feedback loop, 30
filter
band-pass, 107
Bessel, 120
Butterworth, 120
low-pass, 120
moving average, 105
nonlinear, 139
spike, 139
filtering, 113, 118
digital, 33
FIND, 282, 631
finite-state machine, 234, 241
first order system, 333
FLCU, 14, 164, 631
flip-flop circuit
Reset-Set, 252
Set-Reset, 253
FNX, 76, 631
FNXY, 78, 631
FOPDT, 171, 333
FOPDT, 329, 631
frequency transmission band, 107
FRID, 166, 631
From, 45, 48, 49, 631
function
operating system, 37
single variable, 76
two variables, 78
fuzzy logic, 164
GAIN, 80, 631
gain, 80
generator
binary sequence, 152, 154
piecewise linear function, 150
signal, 159
time function, 193
GETPA, 296, 631
GETPB, 298, 631
GETPI, 298, 446, 631
GETPS, 300, 631
Goto, 45–47, 49, 631
GotoTagVisibility, 48, 49, 631
GRADS, 81, 631
HMI, 24, 631
HTTP, 429, 631
HTTP2, 431, 631
hysteresis, 108
I3PM, 168, 631
IADD, 83, 631
identification
three parameter model, 168
IDIV, 87, 631
IMOD, 88, 631
IMUL, 85, 631
INFO, 26, 631
INHEXD, 54, 631
INOC, 54, 631
Inport, 50, 52, 631
INQUAD, 54, 631
INSTD, 45, 54, 55, 631
INTE, 116, 138, 631
integer
division, 87
integer number
transformation, 248
integer signal
switching, 246
integrator
controlled, 116
simple, 138
interpolation
linear, 89
INTSM, 245, 247, 631
IODRV, 23, 27, 45, 47, 631
IOTASK, 29, 36, 40, 296, 298, 305, 307, 320, 323, 631
ISSW, 246, 632
ISUB, 84, 632
ITOI, 248, 632
ITOS, 283, 632
KDER, 118, 632
LC, 170, 632
least squares method, 113
LEN, 284, 632
LIN, 89, 632
linear
interpolation, 89
linear function
generator, 150
LLC, 171, 333, 632
logical sum, 250
loop
algebraic, 30
feedback, 30
low-pass filter, 120
LPB, 13, 30, 103, 632
LPF, 120, 632
LSM, 113
maximum, 121
MB_DASUM, 339
MB_DAXPY, 340
MB_DCOPY, 342
PAGE DIMENSIONS: 595.3x841.9

INDEX

MCP_HaltSuperimposed, 483, 634
MCP_Home, 484, 634
MCP_MoveAbsolute, 486, 634
MCP_MoveAdditive, 489, 634
MCP_MoveCircularAbsolute, 634
MCP_MoveCircularRelative, 634
MCP_MoveContinuousAbsolute, 498, 634
MCP_MoveContinuousRelative, 501, 634
MCP_MoveDirectAbsolute, 634
MCP_MoveDirectRelative, 634
MCP_MoveLinearAbsolute, 634
MCP_MoveLinearRelative, 634
MCP_MovePath, 634
MCP_MovePath_PH, 634
MCP_MoveRelative, 492, 634
MCP_MoveSuperimposed, 495, 634
MCP_MoveVelocity, 505, 634
MCP_PhasingAbsolute, 562, 634
MCP_PhasingRelative, 565, 634
MCP_PositionProfile, 509, 634
MCP_SetCartesianTransform, 634
MCP_SetKinTransform_Arm, 634
MCP_Stop, 522, 634
MCP_StopOverride, 524, 634
MCP_TorqueControl, 526, 634
MCP_VelocityProfile, 529, 634
MCU, 172, 229, 634
MDL, 330, 331, 634
MDLI, 331, 634
mean value, 115
minimum, 121
MINMAX, 121, 634
ML_DGEBAK, 366
ML_DGEBAL, 368
ML_DGEBRD, 370
ML_DGECON, 372
ML_DGEES, 374
ML_DGEEV, 376
ML_DGEEHRD, 378
ML_DGELQF, 380
ML_DGELSD, 382
ML_DGEQRF, 384
ML_DGESDD, 386
ML_DLACPY, 388
ML_DLANGE, 388
ML_DLASET, 390
ML_DTRSYL, 394
model
first order, 329
FOPDT, 171, 333
process, 330, 331
second order, 333
state space
continuous, 323
continuous with time delay, 320
discrete, 327
discrete with time delay, 325
modulation
pulse width, 199
MODULE, 23, 27, 31, 634
module, 31
extending the REXYGEN System, 31
extension, 27
motion control, 15, 106
moving average, 105
MP, 156, 634
MPC, 195
MqttPublish, 458, 634
MqttSubscribe, 460, 634
MUL, 90, 635
multiplexer
bitwise, 238
multiplication
by a constant, 80
extended, 75
two signals, 90
MVD, 332, 635
MX_AT, 396
MX_ATSET, 397
MX_CNADD, 398
MX_CNMUL, 399
MX_CTODPA, 400
MX_DIM, 402
MX_DIMSET, 403
MX_DSAGET, 404
MX_DSAREF, 406
MX_DSASET, 408
negation, 249
nonlinear transformation
  simple, 122
NOT_, 249, 635
NSCL, 122, 635

OPC server, 438
operating system, 37
operation
  binary, 93
  bitwise, 237
operator
  relational, 93
optimization
  gradient based, 81
OR_, 250, 251, 635
order
  driver execution, 27
  driver initialization, 27
  module execution, 31
  module initialization, 31
  of task execution, 38
  of task initialization, 38
ORHEXD, 251, 635
OROCT, 250, 251, 635
ORQUAD, 251, 635
OScALL, 37, 428, 635
OUTHEXD, 55, 57, 635
OUTOCT, 55, 57, 465, 467, 635
Outport, 50, 53, 635
output
  pulse, 192
  three state, 230
output saturation, 202
OUTQUAD, 55, 57, 465, 467, 635

OUTRHEXD, 57, 635
OUTROCT, 57, 635
OUTRQUAD, 57, 635
OUTRSTD, 58, 635
OUTSTD, 47, 54, 55, 58, 635
overhead
  control system core, 22

PARA, 301, 635
parameter
  tick, 22
  input-defined, 302
  remote, 296, 298, 305, 307
  remote acquisition, 296, 298
PARB, 302, 635
PARI, 302, 635
PARR, 137, 302, 635
PARS, 304, 635
path
  full, 35
period
  of quick task execution, 33
  of task execution, 38
PID
  autotuning, 181
  controller, 187
  with gain scheduling, 179
  with input-defined parameters, 190
  with moment autotuner, 181
  with relay autotuner, 174
  with static error, 177
PIDAT, 14, 174, 635
PIDE, 177, 635
PIDGS, 14, 179, 635
PIDMA, 14, 181, 438, 635
PIDU, 174, 177, 179, 181, 187, 190, 229, 438, 472, 635
PIDUI, 190, 635
PJROCT, 286, 635
PJSOCT, 287, 635
POL, 91, 635
polynomial
  evaluation, 91
position sensor
<table>
<thead>
<tr>
<th>Term</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolute</td>
<td>101</td>
</tr>
<tr>
<td>POUT</td>
<td>192, 635</td>
</tr>
<tr>
<td>PRBS</td>
<td>157, 635</td>
</tr>
<tr>
<td>prediction</td>
<td>113</td>
</tr>
<tr>
<td>predictive control</td>
<td>195</td>
</tr>
<tr>
<td>PRGM</td>
<td>193, 635</td>
</tr>
<tr>
<td>priority</td>
<td></td>
</tr>
<tr>
<td>dependency on the operating system</td>
<td>22</td>
</tr>
<tr>
<td>logic</td>
<td>22</td>
</tr>
<tr>
<td>logical</td>
<td>27, 33</td>
</tr>
<tr>
<td>of tasks</td>
<td>38</td>
</tr>
<tr>
<td>process</td>
<td></td>
</tr>
<tr>
<td>model</td>
<td>330</td>
</tr>
<tr>
<td>model with variable parameters</td>
<td>331</td>
</tr>
<tr>
<td>product</td>
<td></td>
</tr>
<tr>
<td>Boolean</td>
<td>232, 233</td>
</tr>
<tr>
<td>logical</td>
<td>232, 233</td>
</tr>
<tr>
<td>program</td>
<td></td>
</tr>
<tr>
<td>REXYGEN Studio</td>
<td>27</td>
</tr>
<tr>
<td>REXYGEN Diagnostics, Enable check-box</td>
<td>35</td>
</tr>
<tr>
<td>REXYGEN Diagnostics, Halt/Run button</td>
<td>35</td>
</tr>
<tr>
<td>REXYGEN Diagnostics, Reset button</td>
<td>35</td>
</tr>
<tr>
<td>RLY</td>
<td>163, 201, 636</td>
</tr>
<tr>
<td>RM_AxesGroup</td>
<td>570, 576, 599, 636</td>
</tr>
<tr>
<td>RM_Axis</td>
<td>422, 471, 472, 484, 513–517, 524, 535, 537, 576, 636</td>
</tr>
<tr>
<td>RM_AxisOut</td>
<td>535, 537, 636</td>
</tr>
<tr>
<td>RM_AxisSpline</td>
<td>537, 636</td>
</tr>
<tr>
<td>RM_Feed</td>
<td>573, 636</td>
</tr>
<tr>
<td>RM_Gcode</td>
<td>574, 636</td>
</tr>
<tr>
<td>RM_GroupTrack</td>
<td>636</td>
</tr>
<tr>
<td>RM_Track</td>
<td>539, 636</td>
</tr>
<tr>
<td>root</td>
<td></td>
</tr>
<tr>
<td>square</td>
<td>96</td>
</tr>
<tr>
<td>RS</td>
<td>252, 636</td>
</tr>
<tr>
<td>pulse width modulation</td>
<td>199</td>
</tr>
<tr>
<td>PWM</td>
<td>180, 185, 189, 191, 199, 218, 222, 223, 635</td>
</tr>
<tr>
<td>QFC</td>
<td>59, 60, 445, 635</td>
</tr>
<tr>
<td>QFD</td>
<td>57–60, 445, 636</td>
</tr>
<tr>
<td>QTASK</td>
<td>22, 23, 29, 33, 36, 38, 320, 323, 431, 636</td>
</tr>
<tr>
<td>quotient</td>
<td>73</td>
</tr>
<tr>
<td>integer</td>
<td>87</td>
</tr>
<tr>
<td>rate limiter</td>
<td>125</td>
</tr>
<tr>
<td>rate monotonic scheduling</td>
<td>23</td>
</tr>
<tr>
<td>RDC</td>
<td>15, 435, 636</td>
</tr>
<tr>
<td>RDFT</td>
<td>123</td>
</tr>
<tr>
<td>real-time</td>
<td></td>
</tr>
<tr>
<td>executive</td>
<td>19, 22</td>
</tr>
<tr>
<td>REC</td>
<td>92, 636</td>
</tr>
<tr>
<td>reciprocal value</td>
<td>92</td>
</tr>
<tr>
<td>REGEXP</td>
<td>288, 636</td>
</tr>
<tr>
<td>REL</td>
<td>93, 636</td>
</tr>
<tr>
<td>relative damping coefficient</td>
<td>107, 120</td>
</tr>
<tr>
<td>relay</td>
<td></td>
</tr>
<tr>
<td>advance</td>
<td>163</td>
</tr>
<tr>
<td>with hysteresis</td>
<td>201</td>
</tr>
<tr>
<td>remote</td>
<td></td>
</tr>
<tr>
<td>data connection</td>
<td>435</td>
</tr>
<tr>
<td>parameter</td>
<td>296, 298</td>
</tr>
<tr>
<td>REPLACE</td>
<td>291, 636</td>
</tr>
<tr>
<td>REXLANG</td>
<td>15, 439, 636</td>
</tr>
<tr>
<td>RLIM</td>
<td>125, 636</td>
</tr>
<tr>
<td>RLY</td>
<td>163, 201, 636</td>
</tr>
<tr>
<td>RM_AxesGroup</td>
<td>570, 576, 599, 636</td>
</tr>
<tr>
<td>RM_Axis</td>
<td>422, 471, 472, 484, 513–517, 524, 535, 537, 576, 636</td>
</tr>
<tr>
<td>RM_AxisOut</td>
<td>535, 537, 636</td>
</tr>
<tr>
<td>RM_AxisSpline</td>
<td>537, 636</td>
</tr>
<tr>
<td>RM_Feed</td>
<td>573, 636</td>
</tr>
<tr>
<td>RM_Gcode</td>
<td>574, 636</td>
</tr>
<tr>
<td>RM_GroupTrack</td>
<td>636</td>
</tr>
<tr>
<td>RM_Track</td>
<td>539, 636</td>
</tr>
<tr>
<td>root</td>
<td></td>
</tr>
<tr>
<td>square</td>
<td>96</td>
</tr>
<tr>
<td>RS</td>
<td>252, 636</td>
</tr>
</tbody>
</table>
INDEX

RTOI, 94, 636
RTOS, 292, 636
RTOV, 421, 426, 427, 636

S1OF2, 126, 636
safety analog input, 129
safety selector, 126
SAI, 126, 128, 129, 636
sample and hold, 137
SAT, 202, 636
SC2FA, 204, 636
schedule
weekly, 262
SCU, 180, 185, 189, 191, 211, 636
SCUV, 180, 185, 187, 189, 191, 214, 636
SEL, 132, 636
selector
active controller, 217
analog signal, 126, 132
safety, 126
with ramp, 142
SELHEXD, 133, 636
SELOCT, 133, 636
SELQUAD, 132, 133, 636
SELSOCT, 293, 636
SELU, 217, 422, 636
sensor
absolute position, 101
sequence
pseudo-random binary, 157
sequential control, 234, 241
SETPA, 305, 636
SETPB, 307, 636
SETP1, 307, 447, 636
setpoint, 193
SETPR, 307, 311, 636
SETPS, 309, 636
SG, 159, 437, 637
SGI, 159, 637
SGSLP, 310, 314, 637
SHIFTQCT, 135, 637
SHLD, 137, 302, 637
signal generator, 159
SILO, 312, 314, 637
SILOS, 316, 637
simulation
parameters, 34
real-time, 34
Simulink, 34
SINT, 116, 138, 637
SLEEP, 13, 34, 637
SMHCC, 218, 637
SMHCCA, 222, 637
SMTP, 433, 637
SOPDT, 333, 637
SPIKE, 129–131, 139, 637
SQR, 95, 637
SQRT_, 96, 637
square root, 96
square value, 95
SR, 253, 637
SRTF, 35, 637
SSW, 141, 422, 438, 637
step controller
with position feedback, 211
with velocity output, 214
STOR, 294, 637
SUB, 67, 97, 637
SubSystem, 46, 48, 52, 54, 55, 637
subsystem
archiving, 265
execution, 35
subtraction
extended, 74
two signals, 97
sum, 66
Boolean, 251
logical, 250, 251
switch
integer signals, 246
simple, 141
unit, 229
with automatic selection of input, 103
SWR, 142, 422, 637
SWU, 217, 229, 637
SWVMR, 422, 637
system
   first order, 171, 329
   second order, 333
TASK, 22, 23, 29, 33, 36, 38, 320, 323, 637
task
   driver-triggered, 29
   execution, 35
   execution period, 38
   priority, 38
   quick, 33
   quick, execution period, 33
   standard, 38
TIME, 259, 261, 637
time delay, 112, 329, 333
variable, 143
timer
   system, 29
   weekly, 262
TIMER_, 254, 637
TIODRV, 23, 29, 40, 637
trajectory
   time-optimal, 106
transformation
   binary number, 248
   integer number, 248
transport delay, 112
trend
   recording, 273, 276
TRND, 17, 273, 276, 437, 637
TRNDV, 276, 637
TSE, 211, 214, 230, 637
type
   input, 16
   output, 16
   parameter, 16
   types
      of variables, 16
user programmable block, 439
UTOI, 637
value
   default, 16
   maximal, 16
   mean, 115
   minimal, 16
   reciprocal, 92
   substitute, 73, 75, 76, 78, 87, 88, 92, 96
valve
   motor driven, 332
   servo, 332
variance, 115
VDEL, 143, 637
VIN, 57, 58, 60, 61, 445, 637
VOUT, 59, 62, 445, 637
VTOR, 123, 423, 427, 637
weekly
   schedule, 262
WSCH, 262, 637
WWW, 42, 637
ZV4IS, 144, 638